Future Internet **2015**, *7*, 445-464; doi:10.3390/fi7040445

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

The Future Internet: A World of Secret Shares

William J. Buchanan *, David Lanc, Elochukwu Ukwandu, Lu Fan, Gordon Russell and Owen Lo

Centre for Distributed Systems and Security, Edinburgh Napier University, Edinburgh EH10 5DT, UK; E-Mails: david_lanc@payfont.com (D.L.); e.ukwandu@napier.ac.uk (E.U.); l.fan@napier.ac.uk (L.F.); g.russell@napier.ac.uk (G.R.); o.lo@napier.ac.uk (O.L.)

* Author to whom correspondence should be addressed; E-Mail: w.buchanan@napier.ac.uk; Tel.: +44-131-455-2759.

Academic Editors: Steven Furnell and Nathan Clarke

Received: 27 April 2015 / Accepted: 12 October 2015 / Published: 24 November 2015

Abstract: The Public Key Infrastructure (PKI) is crumbling, partially due to the lack of a strong understanding of how encryption actually works, but also due to weaknesses in its implementation. This paper outlines an Internet storage technique using secret sharing methods which could be used to overcome the problems inherent with PKI, while supporting new types of architectures incorporating such things as automated failover and break-glass data recovery. The paper outlines a novel architecture: SECRET, which supports a robust cloud-based infrastructure with in-built privacy and failover. In order to understand the performance overhead of SECRET, the paper outlines a range of experiments that investigate the overhead of this and other secret share methods.

Keywords: secret shares; PKI; cloud computing

1. Introduction

Cloud Computing has seen one of the most radical shifts within Information Technology, and the shift is most apparent both in the move from migrating private networks to virtualized networks, as well as the move from private cloud systems onto public ones. Unfortunately these moves have not really changed the methods of providing security and robustness, and instead often rely on the addition of

encryption keys or the implementation of multi-factor authentication techniques. Additionally, the public cloud still suffers from doubts around large-scale outage risks and other security concerns.

One of the major risks in scaling existing systems into the Cloud is the current reliance on PKI (Public Key Infrastructure) to secure information, where there are risks around the possible existence of back-doors, front-doors, and a general lack of true understanding of encryption methods. Many encryption methods use a key pair, such as with Rivest Shamir and Adleman (RSA) algorithm, to protect the symmetric key which is then used to encrypt data into the Cloud. While most common methods are considered free of major vulnerabilities, provided the key size is sufficiently large to make it safe from brute force attacks, it is the loss of the private key which can cause major data loss concerns. Many public cloud-based systems are at risk of major data loss through private key loss (and thus data loss), especially through the growth of APTs (Advanced Persistent Threats), certificate cracking, and insider threats.

The Future Internet thus requires in-built data protection, either with a strong protective shell, such as that provided by sticky policies, or with sharing methods which break data into secure fragments, and which have a strong enforcement policy to rebuild the data shares without relying on traditional encryption.

2. Founding Principles around Secure Data Storage

Protecting data in Cloud-based storage systems is a key challenge [1], and existing architectures often fail to properly control access rights to such data. The insider threat, especially from the likes of a System Administrator, and from unforeseen implementation issues, causes Cloud-based systems to be flawed from many viewpoints. There are thus some key driving forces to ensure that data is protected in future Cloud-based infrastructures, including:

- **Keyless encryption method**. While the methods employed in modern encryption are often theoretically watertight, it is the actual implementation in real-life systems which are flawed. Most systems still rely on digital certificates holding a key-pair, and these certificates are all too easily stolen or compromised through brute force attacks. The Superfish compromise highlights a weakness introduced by the software developers, where a digital certificate with both the public and private key was distributed with the software, and which also had a default password based on the name of the company who developed the software [2].
- **Self-destruct**. Some techniques support the concept of a self-destructing data system. With this, all the stored information and their copies, as well as decryption keys, will become unusable after a user-specified time period and without any user involvement. SeDas [3] causes sensitive information, such as account numbers, passwords and notes to irreversibly self-destruct, without any action on the user's part. This technique is applicable in a multiple server-based system, admittedly incurring an operational overhead.
- **Break-glass data recovery.** Many systems have strict access control policies for data, but in some circumstances it is important that access to data can be granted in emergency situations. This is typical in life-threatening scenarios, such as in health and social care. The loss of an encryption key can also cause major problems in data recovery, which would hinder a break-glass requirement.

- **In-built failover protection.** For instance, data which needs redundancy can be replicated and encoded across multiple cloud-based storage systems using an *any k-from-n* sharing policy. So for example, with a storage system striped across three Cloud storage platforms, a policy of 2-from-3 can support a single Cloud storage systems failure, while still allowing the original data to be recovered.

This paper looks at a range of techniques for tackling these issues using a *secret sharing* technique, where the information to be made secret is encoded and distributed across multiple data shares, where each share of the information is stored on a different cloud provider.

3. Related Methods

3.1. Basic Data Striping and RAID Storage Systems

RAID stands for Redundant Array of Independent Disks. It is a well-known method of combining several hard disk drives into one logical unit, so as to address the fault-tolerance and performance limitations of an individual hard disk drive. The key notions behind a RAID storage system are Data Striping and various levels of Redundancy. The striping process takes large data blocks and splits them into blocks of a defined size, such as 4 KB, which are then spread across each of the disks in the array. This can increase the performance of a storage system as a n-disk array (*i.e.*, n-way striping) provides n times the read and write speed improvement of a single disk (although in practice the actual performance achieved tends to be less than n times due to control overheads). However, the trade-off is reliability, as the failure of any individual disk would result in the failure of the entire array. Formally, assume that each independent hard disk drive has an identical rate of failure r, then the overall failure rate of an array of n disks is:

$$1 - (1 - r)^n$$

Basic data striping over an array of n-disks without redundancy is referred to as RAID0, which is only suitable to situations where the highest I/O performance is desired, whilst the resulting increased probability of data loss can be tolerated. In other situations, reliability of data may outweigh system performance, and thus instead of striping the data over n disks, the same data is duplicated (or mirrored) to n disks. This strategy is referred to as RAID1, which is suitable to ensure the reliability of critical data. In comparison to RAID0, the overall failure rate of a RAID1 array of n disks becomes:

$$r^n$$

However, when the same set of data is duplicated over n disks, it would result in a storage overhead of $n - 1$ disks.

RAID0 and RAID1 can be combined into a RAID01 or a RAID10 configuration. The former means a mirrored configuration of multiple striped sets (*i.e.*, mirror of stripes); and the latter means a stripe across a number of mirrored sets (*i.e.*, stripe of mirrors). Both strategies are able to provide very good performance and reliability, whereas both are expensive solutions as considerable amount of hard disk storage is committed to maintaining redundancy information.

Another RAID level, namely RAID5, attempts to mitigate the high costs of RAID01/10. RAID5 utilizes striping and parity techniques to achieve simultaneously a similar I/O performance to RAID0 with a significantly lower failure rate of:

$$1 - (1 - r)^n - nr(1 - r)^{n-1}$$

A parity bit (or check bit) is a bit added to the end of a string of binary code that indicates whether the number of bits in the string with the value 1 is even or odd. In computing and telecommunications, a parity bit is calculated via an XOR sum of all the previous bits, yielding 0 for even parity and 1 for odd parity. Let A and B being two binary sequences. If:

X = A XOR B

then the following will be true:

A = X XOR B
B = X XOR A

Using this property, we can say that the following expressions are also true (and this can be repeated for an infinite amount of terms):

A XOR B XOR C XOR D = X
X XOR B XOR C XOR D = A
A XOR X XOR C XOR D = B
A XOR B XOR X XOR D = C
A XOR B XOR C XOR X = D

Put simply, if we calculate a XOR of a number sequence, we can substitute X for any number in this sequence, recalculate the XOR, and recover the original number. This is the principle for building a RAID5 array, of which an example is shown in Figure 1. Each of the four drives in Figure 1 are divided into four blocks, each belonging to a stripe on the same level across each drive. Each drive and each stripe have a parity block (stored in disk 4) which is the XOR of the other three blocks. So, say Disk 2 failed, then the XOR of A1 with A2 and A4 would get the remaining A3 for the A stripe; B1 with B2 and B3 to get the B4 parity block; *etc.* Generally speaking, by introducing a single parity bit, RAID5 is able to survive the failure of any single hard disk drive in any array which contains a minimum of three drives.

The RAID5 data striping mechanism can be applied to secret sharing of data in the Cloud, where each cloud storage provider represents a disk storing one share of the secret data, but the limitation of this lies in that it always requires $n - 1$ shares to reconstruct the original data, rather than a more desirable and flexible arbitrary k-out-of-n shares.

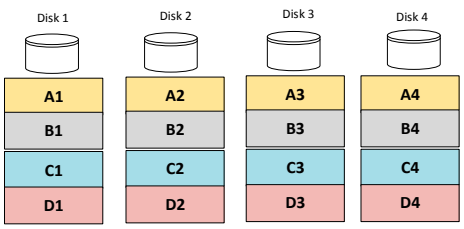


Figure 1. A 4-disk array in RAID5 format.

3.2. Reed-Solomon Error Correcting Code

Reed-Solomon's error correcting code [4] was developed by Irving Reed and Gustave at MIT Lincoln Laboratory in 1960. By adding t check symbols to the data, the RS code can detect any combination of up to t erroneous symbols, and correct up to $t/2$ symbols or t known erasures. RS code is often used to correct burst errors on storage media, such as for CD-ROMs. For example, RAID6 extends RAID5 by using two RS parity blocks to recover from the failure of any two hard disk drives in an array of a minimum of four drives.

RS code is suitable for applying to secret sharing, because the t-out-of-n property of a secret sharing scheme is equivalent to the loss of $(n - t)$ shares, with the positions of the missing shares known in advance. In other words, $(n - t)$ check symbols can be introduced at the point when the shares were created, e.g. use two check symbols to ensure that any three out of the five shares can be combined together to reconstruct the original data.

A secret sharing scheme implemented using the RS code algorithm can be highly efficient, but unfortunately not very secure. A secure secret sharing scheme should distribute shares so that anyone with fewer than t shares has no extra information about the secret than someone with zero shares. For example consider the secret sharing scheme in which the secret phrase "password" is divided into the following shares (plus RS code shares):

"pa------", "--ss----", "----wo--", "------rd"

A person with 0 of these shares may only perhaps know that the password consists of eight letters. For this they would have to guess the password from 208 billion combinations (26^8). With one share, however, they would only have 308 million combinations (as to guess only the six letters—26^6), and so on as more shares became available. RS code is not a secure secret sharing scheme, because it is possible for a player with fewer than t secret-shares to partially identify some of the original data.

4. Basic Data Striping Schemes in Cloud-Based Systems

The major concerns of moving towards Cloud-based storage have been the security and availability of data when accessed. Several researchers have opinioned that a multi-cloud storage platform might improve security and resilience [5–7]. Based on this, to ensure effective data splitting and security of stored data, a secret sharing algorithm such as Shamir secret sharing has been proposed for multi-cloud storage [8–13]. Previous methods split the data into pieces and then reconstruct when needed, while replicating data at different storage areas using encryption to safeguard them. This type of methodology can be costly (in terms of network bandwidth and storage facilities), and not quite safe in scenarios involving poor key management.

Dodis [6] states that a basic secret sharing scheme which lacks redundancy is given by two algorithms: Sharing (*Share*); and Recovery (*Rec*). Share takes a message M and split it into n pieces. Since M is secret, *Share* must introduce randomness (that is, Share is probabilistic); *Rec* is a deterministic algorithm which recreates the message from some or all of the shares:

- **The Sharing Algorithm:**

Share(M) → (S_1, S_2, …. S_n, *pub*). The secrets S_1, S_2, …. S_n are distributed securely among servers 1 through to n, and pub is a public share.

- **The Recovery Algorithm:**

(S'_1, S'_2, …. S'_n, *pub*) = M'. The correctness property of the algorithm defines that, for any message M, Rec(Share(M)) = M

where → depicts randomness.

Secret sharing schemes are important in cryptography and are a key building block for many secure protocols, such as for: Threshold Cryptography; Access Control; Attribute-Based encryption; Byzantine agreement; generalised oblivious transfer; and general protocol for Multi-Party computation [14]. For general algorithms, the following thresholds are useful:

- t_p**—the privacy threshold.** This describes the maximum number of servers that cannot determine the secret, even when they combine their shares together.
- t_f**—the fault-tolerance threshold.** With this, supposing that every server is honest, this is the minimum number of servers you need to recover the secret (if the other servers are absent).
- t_r**—the robustness threshold.** This is arguably the most important, as this is the minimum number of correct shares you need to recover the secret if all other servers are compromised.
- t_s**—the soundness threshold.** This determines the minimum number of correct shares such that it is not possible to recover the wrong secret.
- t_i**—the information rate threshold.** This describes the amount of information that the server needs to keep on a secret and determines the efficiency and idealness of a secret sharing scheme.

4.1. Sharing Methods

There are several relevant methods which could be used for secret sharing and these are outlined in the following sections. The most relevant methods are then evaluated at the end of this paper.

4.1.1. Perfect Secret Sharing Scheme (PSS)

Shamir (1979) provides a good example of a perfect secret sharing [15]. It requires two conditions to be perfect: if, and only if, $t - 1$ shares provide absolutely no information regarding the hidden secret, and when the ratio of the length of the secret to the length of each of the shares (known as the information rate) is equivalent to 1.

Shamir's PSS relies on the idea that on the principle that you can define a straight line with two points, three points for a quadratic equation, and so on, to give t points to define a polynomial of degree $t - 1$. Hence, a method for t-out-of-n secret sharing can thus use a polynomial with a $t - 1$ degree using a secret for the first coefficient, and then random values for the remaining coefficients. Next, find n points on the curve and give one to each of the players. As a result, when at least t out of the n players reveal their points, there is sufficient information to fit a $(t - 1)$th degree polynomial to them, in which the first coefficient is the secret.

To illustrate this construction technique, consider a concrete example. Assume that a secret value is split into five parts, three of which are needed to reconstruct the original data (*i.e.*, $n = 5$, $t = 3$). The first step is to create a second degree polynomial:

$$y = a_0 \times x^0 + a_1 \times x^1 + a_2 \times x^2$$

The second step is to assign the secret value to the first coefficient (*i.e.*, a_0) and choose random values for the remaining coefficients (*i.e.*, a_1 and a_2). Suppose that the secret value is: 42, $a_1 = 1$ and $a_2 = 2$, then the polynomial becomes:

$$y = 42 + x + 2 \times x^2$$

The third step is to calculate any five (x, y) pairs, for instance:

x		y
1	$42 + 1 + 2 \times 1$	45
2	$42 + 2 + 2 \times 4$	52
3	$42 + 3 + 2 \times 9$	63
4	$42 + 4 + 2 \times 16$	78
5	$42 + 5 + 2 \times 25$	97

Finally, distribute one pair to each player, e.g., Player1 receives (1, 45), Player2 receives (2, 52), and so on. No player can thus tell anything about the original secret, unless at least three players exchange their information and yield equations like:

$$45 = a_0 + a_1 \times 1 + a_2 \times 1^2$$
$$52 = a_0 + a_1 \times 2 + a_2 \times 2^2$$
$$63 = a_0 + a_1 \times 3 + a_2 \times 3^2$$

Hence, the three players together can work out that the secret value a_0 is 42. This method works fine, but from security point of view a player can still obtain more information about the secret with every pair on the polynomial that they find. For example, player Eve finds two pairs (2, 52) and (4, 78). Although these are not enough to reveal the secret value, Eve could combine them together and get:

$$76 = a_0 + a_1 \times 4 + a_2 \times 4^2$$
$$52 = a_0 + a_1 \times 2 + a_2 \times 2^2$$

Therefore, Eve can work out that:

$$a_0 = 26 + 8 \times a_2$$

So, Eve starts to replace a_2 with 0, 1, 2, 3… to find all possible values of a_0. This problem can be fixed by using finite field arithmetic in a field of size r where:

$r > a_i$, $r > N$, $r = p^k$, $p \in$ P where P is the set of primes and k is a positive integer.

Then, calculate the pairs as:

$$y = f(x) \bmod(p)$$

For example, consider the example of $p = 61$:

x	***p***		***y***
1	61	(42 + 1 + 2 × 1) % 61	45
2	61	(42 + 2 + 2 × 4) % 61	52
3	61	(42 + 3 + 2 × 9) % 61	2
4	61	(42 + 4 + 2 × 16) % 61	17
5	61	(42 + 5 + 2 × 25) % 61	36

Shamir's PSS is a promising approach to secret sharing for cloud storage, and provides many advantages, including:

- **Secure**—Anyone with fewer than t shares has no extra information about the secret than someone with zero shares.
- **Extensible**—When n is fixed, new shares can be dynamically added or deleted without affecting the existing shares.
- **Dynamic**—With this it is possible to modify the polynomial and construct new shares without changing the secret.
- **Flexible**—In organisations where hierarchy is important, it is possible to supply each participant different number of shares according to their importance.

However, the PSS scheme also has the following drawbacks:

- **Computational overheads**—The preparation of the polynomial, the generation of shares, and the reconstruction of the secret may consume significant computational resources.
- **Storage and transmission overheads**—Given $t - 1$ shares, no information whatsoever can be determined about the secret. Hence, the final share must contain as much information as the secret itself, which means effectively each share of the secret must be at least as large as the secret itself. The storage and transmission of the shares requires an amount of storage space and network bandwidth equivalent to the size of the secret times the total number of shares.

4.1.2. Share Distribution

An honest Dealer (D) selects n distinct, non-zero elements of $\mathbb{Z}$p (the finite field of size p), denoted by x_i where $i \in [1,n]$ and it is required that $p \geq n + 1$ because in a field of $\mathbb{Z}$p, there can be at most $p - 1$ participants. D spreads the value x_i to P_i and these x_i values are public. The Dealer selects the secret $k \in \mathbb{Z}$p, secretly, and independently, at random, selects $t - 1$ elements of $\mathbb{Z}$p denoted by $a_1, a_2, \ldots a_{t-1}$. For $i \in [1,n]$, dealer computes $y_i = a(x_i)$, where:

$$a(x) = k + \sum_{j=1}^{t-1} a_j x^j \pmod{p} \tag{1}$$

For $i \in [1,n]$, dealer gives the Share $S = y_i$ to P_i.

To recover the information, a group of participants A, which are members of the access structure Γ defined as ($A' \subset A$: A' can reconstruct the secret k) are required to meet to recover the secret. $A \in \Gamma$ and can reconstruct the secret $k = a(0)$ by substituting $x = 0$ into LaGrangian interpolation formula:

$$k = \sum_{j=1}^{t} \left(y_{i_j} \prod_{1 \le n \le t, n \ne j} \frac{x_{i_n}}{x_{i_n} - x_{i_j}} \right) \bmod p \quad (2)$$

4.1.3. Information Dispersal Algorithm (IDA)

Rabin (1989) suggested splitting a secret S into n pieces such that a person can obtain the secret only if $k < n$ of these pieces are available, where k is the threshold. Here, each secret S_i, $i \le n$, is of size $|S|/k$, where $|S|$ is the size of the secret [16]. The total sizes of all the secrets are:

$$(n/k) \times |S|$$

Thus, with Rabin's Information Dispersal Algorithm, the storage complexity of a secret sharing system can be significantly reduced in comparison to Shamir (1979) perfect secret sharing (PSS) scheme. But, the security flaw in this method is that, if the data exhibits some pattern frequently, and that the attacker gets hold of $m < k$ slices, there are great possibilities for the attacker gaining the secret S.

Both the split and combine algorithms operate by performing matrix multiplication over the input. In the case of the split operation, the transform matrix has n rows *(n = number of shares)* and k columns *(k = quorum*), while, in the case of combine operation, the transform matrix has k rows and k columns. Either operation is described simply as the matrix multiplication:

Transform matrix × Input matrix = Output matrix

In the case of the split operation, the Transform Matrix has n rows and t columns, while in the case of combine operation, the Transform Matrix has t rows and t columns. The Input Matrix always has t rows, while the Output Matrix will have n rows for the split operation, or t rows for the combine operation.

The Transform Matrix must have the property that any subset of t rows represent linearly independent basis vectors. If this is not the case then the transform cannot be reversed. To generate a Transform Matrix, a *key* must be defined as a list of distinct elements, *i.e.*,:

$$x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_t$$

Then, the Transform Matrix is created as:

$$\begin{bmatrix} \frac{1}{x_1 + y_1} & \frac{1}{x_1 + y_2} & \cdots & \frac{1}{x_1 + y_t} \\ \frac{1}{x_2 + y_1} & \frac{1}{x_2 + y_2} & \cdots & \frac{1}{x_2 + y_t} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{1}{x_n + y_1} & \frac{1}{x_n + y_2} & \cdots & \frac{1}{x_1 + y_1} \end{bmatrix}$$

4.1.4. Krawczyk's Computational Secret Sharing (CSS)

Hugo Krawczyk [17] proposed the Computational Secret Sharing (CSS) technique (a.k.a. *secret sharing made short*), which combines Rabin's IDA with Shamir's PSS. Data is first encrypted with a randomly generated key, using a symmetric encryption algorithm. Next this data is split into n fragments using Rabin's IDA with a threshold t configured. In this case, the scheme is t times more efficient than

Shamir's PSS. The final step is to use Shamir's PSS to produce shares of the randomly generated symmetric key (which is typically of the order of 64 to 256 bits) and then give one share and one fragment to each shareholder.

A related approach, known as AONT-RS [18], applies an All-Or-Nothing transform (AONT) to the data as a pre-processing step to the IDA. AONT guarantees that any number of shares less than the threshold is insufficient to decrypt the data.

4.1.5. Rabin's and Ben-Or's Publicly Verifiable Secret Sharing (PVSS)

Some applications of a secret sharing scheme, such as e-auction, e-voting and multiparty computation, require public verifiability. Namely, it must be publicly verifiable without revealing the secret or any of its shares that all the n shares are consistently generated from a unique share-generating polynomial such that any t of them can reconstruct the same secret. This capability is crucial to overcome the problem of dishonest share dealers.

Tal Rabin and Michael Ben-Or [19] devised a publicly verifiable secret sharing system that supports dishonest detection for the dealer, or for one-third of the players, even if those players are coordinated by an adaptive attacker who can change strategies in real-time depending on what information has been revealed. The PVSS is essentially a combination of secret sharing and publicly verifiable encryption and works as follows:

(1) The dealer creates shares S1, S2, ..., Sn for each participant P1, P2, ..., Pn respectively;
(2) The dealer encrypts the shares for each participant using their public keys and publishes the ciphertexts Ei(Si);
(3) The dealer also publishes a string ProofD to show that each Ei encrypts Si and the reconstruction protocol will result in the same secret S;
(4) Anybody knowing the public keys for the encryption methods Ei, can verify the shares;
(5) If one or more verifications failed, the dealer fails and the protocol is aborted;
(6) Each participant Pi decrypts their shares Si using Ei(Si);
(7) The participant releases Si plus a string ProofPi to show that the released share is correct;
(8) Dishonest participants who failed to decrypt Si or to generate ProofPi will be excluded;
(9) The secret S can be securely reconstructed from the shares of any qualified set of participants.

Kun Peng [20] carried out a critical survey of five existing PVSS algorithms which claimed strong security and high efficiency. Surprisingly, in practice none of the algorithms can achieve efficiency as high as O(tn) exponentiations, which is widely believed to be feasible.

5. Secret Sharing in a Multi-Cloud Environment

The secret sharing methods show potential in being used in Cloud-based infrastructures, as they inherently preserve the data without the requirement for private keys. Figure 2 outlines an example of a proposed architecture known as SECRET which supports a secret sharing scheme in a multi-cloud environment. It has four main elements:

- **Application platform**. Its function is to: determines the access structure; encodes secrets; sends secrets to the main multi-cloud proxy server for distribution to multi-cloud service providers, as well as keeping the secret shares when recovered.
- **Main multi-cloud proxy server with router**. This splits and distributes encoded shares to multi-cloud based on pre-determined access structure and manages the fail-over protection of shares.
- **Metadata server**. This includes the functionality of: User management; Server management; Session management; and File metadata management [3].
- **Multi-cloud proxy server**. This Gathers shares and reconstruct secret as well manages break-glass data recovery.
- **Sub-routers.** This creates a path between the Cloud Service Provider (CSP) (considered here as front-end) with other Cloud Service providers (considered here as the Back-ends) thereby creating a quick and alternative recovery path for all the shares. For example: R4 connects with R3 + R2 + R1, so R4 is a path for CSP1 + R3 + R2 + R1 + S1, and so on.

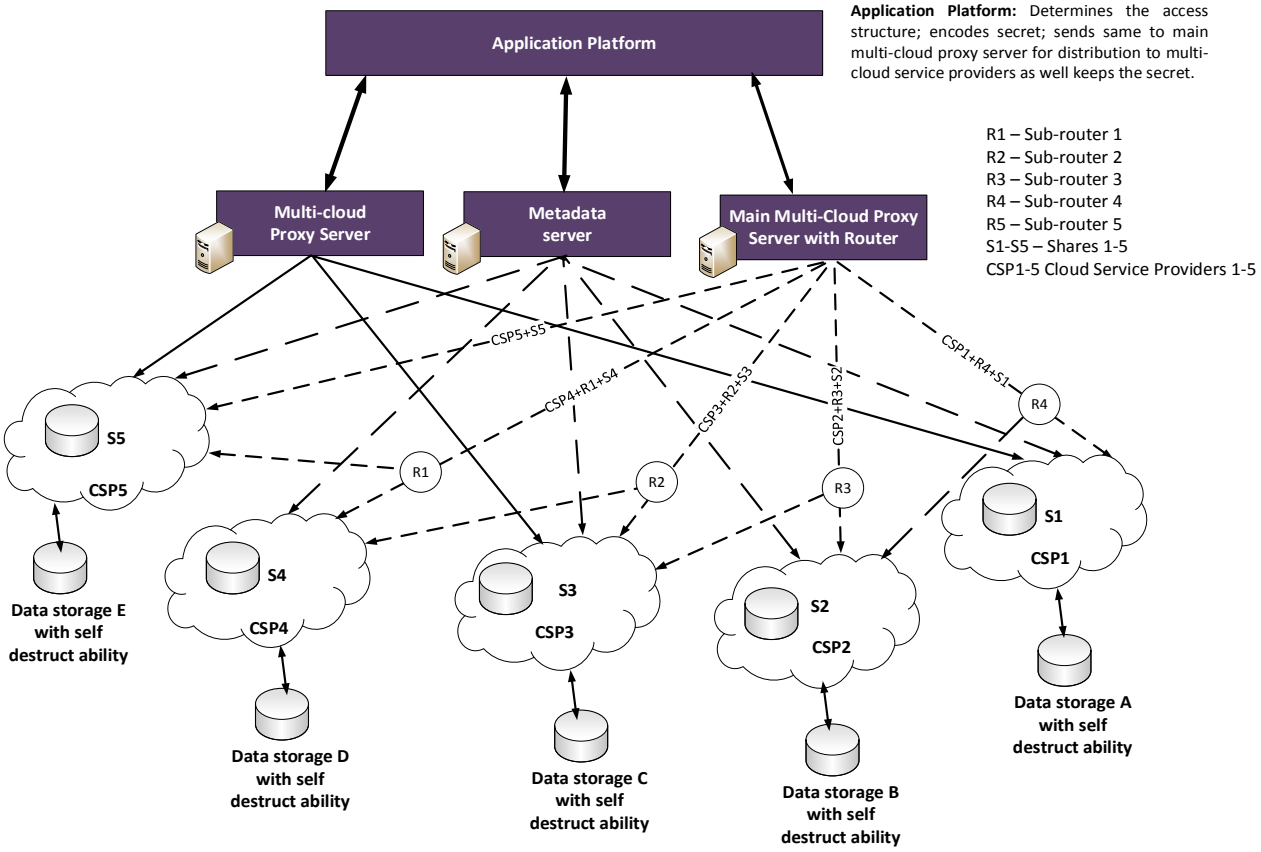


Figure 2. Proposed architecture of a Secret Sharing Scheme in a Multi-cloud environment.

5.1. The SECRET Architecture

SECRET is one of the efforts geared towards accelerating the adoption of multiple cloud storage among enterprises due to its multiple benefits especially as it concerns its pay-as-you-use concept, reduction of capital expenditures in personnel hiring for data management; purchase of huge infrastructure for data storage through data outsourcing; the simplification of wider information sharing

among users due to wider coverage areas and increased in data confidentiality, availability and integrity. In many other similar works like [21,22], the use in exchange of Electronic Health Records were discussed extensively and the prove that such deployment can provide an increased data availability, confidentiality and integrity of the documents stored in the cloud, and so on were provided. SECRET architecture tends to extend these features by adding the possibility of a break-glass data recovery system, near keyless encryption with improved data privacy and security leading to the elimination of threats in poor key management by incorporating Self-Destructive Data System (SeDaS) [3] on the data to be distributed among the cloud service providers. This therefore enhances the security and privacy of data at every stage (in motion, at rest or in use) within and outside the cloud environment.

5.2. How It Works

At the application platform, the data owner determines n and t values (see Section 6) and using both calls up the application to be used, selects algorithm of choice based on our Evaluation in Section 6 after a successful sign in to the system and access level determined. It will be nice to base the choice of algorithm on performance as it relates to security, overhead cost and data size. Figure 2 shows a *3-out-of-5* access structure, while Section 6 use *4-out-of-10* and *2-out-of-5*. The encoded data is sent to the local main multi-cloud proxy server with router for onward dissemination to the CSPs. The proxy splits the encoded data according to a secret-sharing scheme determined access structure, and distributes each share over the Internet to different CSPs. The retrieval process is similar to the storage process as the metadata server helps to keep track of the siblings of the shares. The proxy retrieves enough corresponding shares from the cloud service providers. This retrieval involves authentication to the cloud providers. The retrieved shares are sent back to the application platform software, which decodes them and verifies their authenticity before reconstructing the data.

The design incorporates unique feature in a multi-cloud environment as it uses secret sharing scheme to implement near keyless encryption using SeDaS. This is done by breaking the secret into chunks called (*k-out-of-n*) threshold in such a manner that less than k shares cannot recover the secret, thus using it for data distribution in object storage system. This is also used to implement self-destruct with equal divided shares. The incorporation of a Self-Destructive system solves the problem of cloud user's privacy as there is no way the user's data can be accessed, copied, cached or used without the data owner's consent within a pre-determined time-frame as all data and their copies become destructed or unreadable after a user-specified time, without any user intervention [3]. The self-destructive system defines two modules: a self-destruct method object; and survival time parameter for each secret key part. In this case, a secret sharing algorithm is used to implement share distribution in object storage system so as to ensure safe destruct with equally divided shares. Based on active storage framework, object-based storage interface will be used to store and manage the equal divided shares.

The use and implementation of a threshold system in cloud services are a deliberate act towards implementing failover protection in the model. In normal circumstances all the service providers are used in share storage as well as secret reconstruction, but in an extreme and desperate situation, *2-out-of-5* can be made redundant. That is to say if *2-out-of-5* CSPs fails, data/secret storage and data reconstruction are still possible.

There is also an incorporation of a Break-Glass data recovery system, implemented using one of the Proxy Servers. Just as in Secret Sharing, a trusted dealer can decide to give more shares to a more trusted and reliable participant, which will enable the secret to be reconstructed by fewer participants. SECRET leverages on this concept to implement Break-Glass data recovery by establishing an access to Multi-Cloud proxy Server II, which entails an access to CSPs 1, 3 and 5 and this in turn ensures a quick recovery of shares in order to reconstruct the secret as it is a quick link to all other CSPs and moreover, following the access structure, such access ensures the possibility of reconstructing the secret in an emergency situation. This is an important feature as there could be a period of cloud outage as stated in [23], and in such situation, data recovery could be done from 3-*out-of-5* Cloud service providers being used for data storage. That is to say, if *2-out-of-the-5* cloud service providers fail, data recovery is still possible in such an extreme condition.

Following the above analysis, SECRET therefore shows that the architecture enhances privacy, confidentiality, integrity, availability and security of data at every state (rest, in motion or in use). There is also a reduced overhead and time lags associated with use of Shamir's algorithm by supporting an integration of Rabin's IDA with perhaps Krawczyk's CSS. The proposition of near keyless encryption thereby avoiding the problem associated with key management is also an added advantage.

Therefore, the main advantages of the proposed scheme includes:

- Fast and efficient data/key distribution to multi-cloud service providers.
- Keyless encryption and by extension data security and reduced key management hassles.
- Provide data owner's privacy by implementing SeDas, as it meets all the privacy-preserving goals [3].
- SeDas supports security erasing files and random storage in drives (Cloud, HDD or SSD) respectively [3].
- Backup operational mode in which the function of *5* CSPs can be assumed by *3* CSPs when *2-out-of-the-5* CSPs become unavailable either through failure or scheduled down time.
- Break-glass data recovery.

6. Evaluation

Secret sharing schemes have been used successfully in data splitting and reconstruction, thereby providing data security in a keyless manner. This section outlines an experiment involving three of the main contenders for Secret Sharing schemes in Cloud-based systems: Adi Shamir's Perfect Secret Sharing Scheme (PSS), Hugo Krawczyk's Secret Sharing made short or Computational Secret Sharing scheme (CSS) and Rabin's Information Dispersal Algorithm (IDA), see Tables 1–8. The experiments were conducted in collaboration with a commercial partner, Payfont Limited, that has further validated the results below in a highly scaled, commercial technical infrastructure. The evaluator in this case is the performance overhead at increasing thresholds and data sizes show varied behaviours that depict their strengths and weaknesses at different application scenarios. Figures 3–6 outline the results.

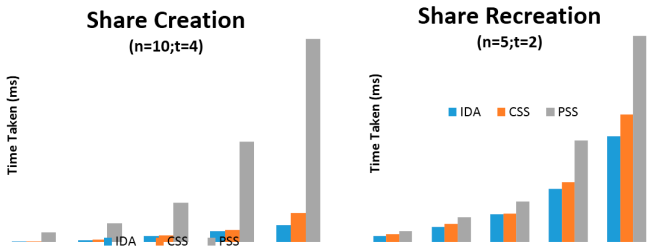


Figure 3. Share creation and recreation against increasing data sizes (n = 5; t =2).

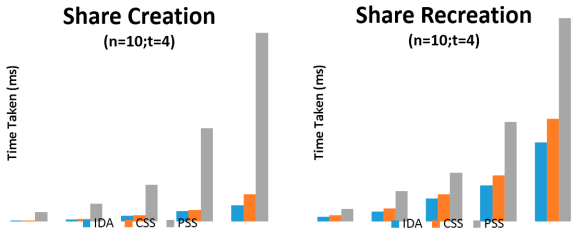
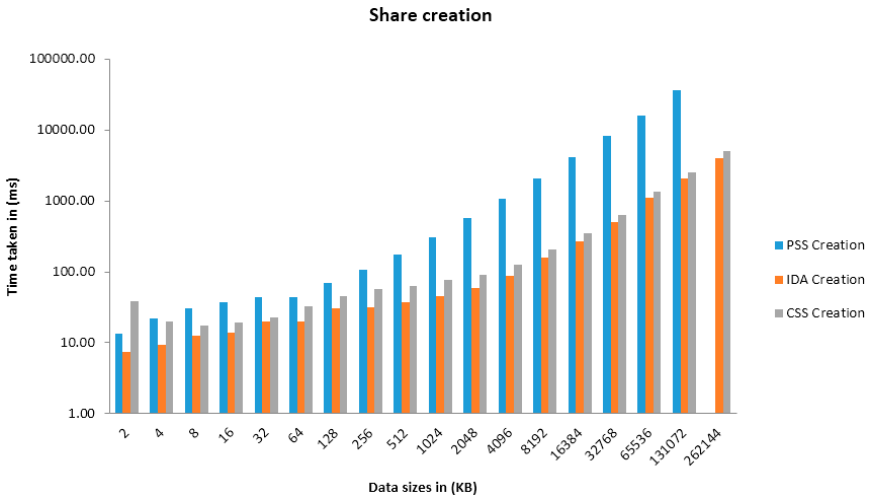
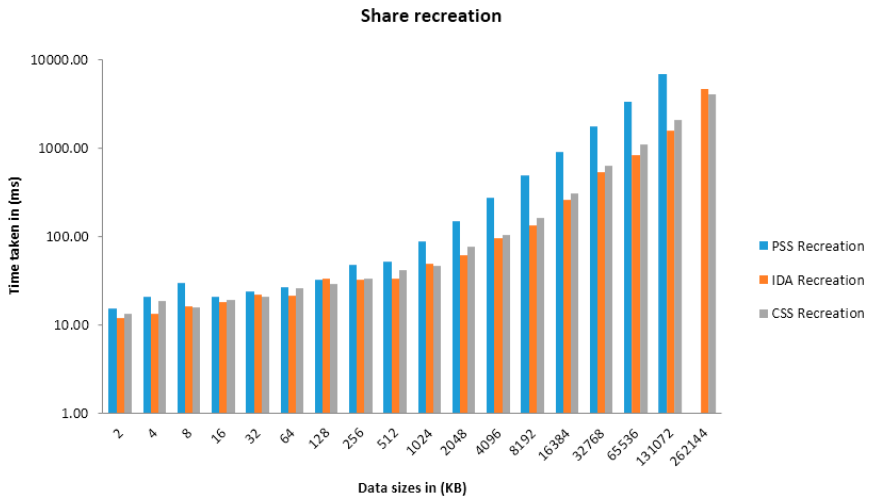


Figure 4. Share creation and recreation against increasing data sizes (n = 10; t = 4).



(**a**)



(**b**)

Figure 5. Graphs of Time taken against Data sizes in share creation (**a**) and recreation (**b**) when n = 10, t = 6.

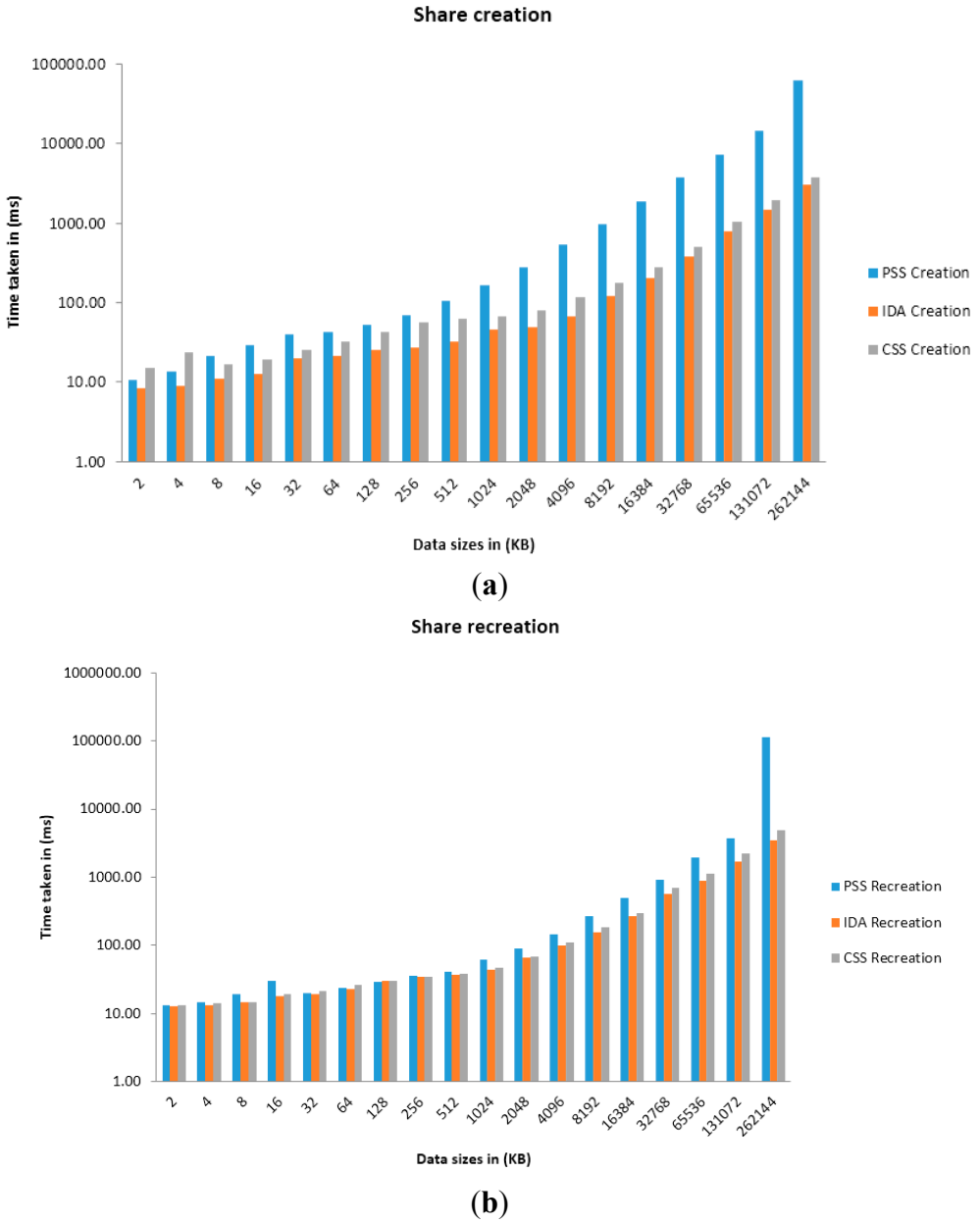


Figure 6. Graph of Time taken against Data sizes in share creation (**a**) and recreation (**b**) when $n = 5$, $t = 3$.

Two different experiments were performed on two different machines. The aim of the experiments were to demonstrate the implication data size variations have on the performance of each secret sharing scheme (SSS) algorithm in terms of share creation and share recreation. Data sizes from 1024 KB to 16,384 KB and 2 KB to 262, 144 KB were evaluated on machines (a) and (b) respectively. The data generated are arbitrary due to the fact that the evaluations are not catered for in relation to one specific area where SSS algorithms may be applied in. The test machines are (a) a D-Series 3 specification Microsoft Azure virtual machine which consists of 4 vCores, 14 GB of RAM and a 200 GB SSD; and (b) an HP EliteDesk 800 G1 USDT, X64-based PC running on Intel® Core ™i5-4570S CPU @ 2.90 GHz, 2901 MHz, 4 cores, 4 GB RAM and a 460 GB SATA drive.

Four primary sets of results were presented which use the parameters of ($n = 5$, $t = 2$) and ($n = 10$, $t = 4$) on machine (a) and ($n = 5$, $t = 3$) and ($n = 10$, $t = 6$) on machine (b). The variable n relates to the

number of shares to create while the variable *t* relates to the number of shares required for recreation of the original arbitrary data (using each SSS algorithm). Time taken for share creation and recreation unit is presented in milliseconds. From Figures 3–6 and Tables 1–8, it can be clearly demonstrated that IDA is the fastest algorithm regardless of data size. CSS comes second in terms of time taken for share creation and recreation while PSS comes last. Some significant observations in the results presented is that PSS demonstrates greater issues in regards to scalability as the data size increases in comparison with the other two algorithms and as it gets beyond 131,072 KB on machine (b) shares could no longer be created nor recreated as the machine throws up out of memory exceptions. Additionally, as we increase the parameters from $n = 5$; $t = 2$ to $n = 10$; $t = 4$ and from $n = 5$; $t = 3$; $n = 10$; $t = 6$ it can be demonstrated that only share creation will produce significant increase in performance time.

Although IDA has demonstrated the fastest time in results presented in this paper it would be naive to simply use this algorithm from these results alone. Depending on the context and application, there may be a need to strike a fine balance between ensuring strong security and acceptable level of performance. Thus, ultimately, the decision on which SSS algorithm to use will be most dependent on the use-case scenario at hand.

Table 1. Time taken (ms) of share **creation** ($n = 5$; $t = 2$).

Algorithm	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)
IDA	14.42	31.90	65.83	126.52	274.43
CSS	21.27	43.29	97.25	174.97	380.53
PSS	172.21	304.94	622.16	1065.41	2228.30

Table 2. Time taken (ms) of share **recreation** ($n = 5$; $t = 2$).

Algorithm	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)
IDA	26.35	63.39	116.69	223.60	445.53
CSS	35.01	76.88	121.42	251.43	536.36
PSS	45.69	104.73	171.81	428.29	867.35

Table 3. Time taken (ms) of share **creation** ($n = 10$; $t = 4$).

Algorithm	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)
IDA	19.97	58.90	185.20	333.22	518.00
CSS	26.80	70.81	192.48	367.04	871.62
PSS	298.25	567.11	1169.94	3013.11	6091.91

Table 4. Time taken (ms) of share **recreation** ($n = 10$; $t = 4$).

Algorithm	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)
IDA	21.04	45.11	104.75	163.78	362.32
CSS	26.83	59.10	122.94	210.51	470.08
PSS	56.79	139.10	222.47	455.58	932.91

Table 5. Time take in (ms) for share creation ($n = 10$, $t = 6$).

Algorithm	2 (KB)	4 (KB)	8 (KB)	16 (KB)	32 (KB)	64 (KB)	128 (KB)	256 (KB)	512 (KB)	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)	32768 (KB)	65536 (KB)	131072 (KB)	262144 (KB)
CSS	38.21	20.26	17.21	19.31	23.07	32.86	44.94	56.65	63.39	77.71	90.95	125.22	203.46	350.05	637.28	1337.67	2492.91	5005.75
IDA	7.31	9.47	12.43	13.78	19.98	20.21	30.82	31.14	36.97	44.93	59.78	87.37	156.65	268.90	505.48	1109.95	2042.17	4030.90
PSS	13.44	22.21	30.40	37.04	44.05	44.65	70.71	106.58	175.70	306.61	564.58	1067.90	2078.61	4129.68	8163.94	15999.61	36832.71	-

Table 6. Time take in (ms) for share recreation ($n = 10$, $t = 6$).

Algorithm	2 (KB)	4 (KB)	8 (KB)	16 (KB)	32 (KB)	64 (KB)	128 (KB)	256 (KB)	512 (KB)	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)	32768 (KB)	65536 (KB)	131072 (KB)	262144 (KB)
CSS	13.57	18.60	16.08	19.05	21.23	26.35	29.21	33.62	41.61	46.66	77.40	104.98	162.85	305.54	639.02	1104.15	2103.79	4050.72
IDA	12.03	13.32	16.56	18.49	21.94	21.66	33.58	32.53	34.03	48.91	62.42	96.28	135.73	263.71	535.24	832.55	1595.89	4722.39
PSS	15.44	20.92	29.91	20.73	24.11	26.85	32.96	48.69	52.16	88.83	149.98	272.39	492.71	908.36	1779.57	3341.85	6930.38	-

Table 7. Time take in (ms) for share creation ($n = 5$, $t = 3$).

Algorithm	2 (KB)	4 (KB)	8 (KB)	16 (KB)	32 (KB)	64 (KB)	128 (KB)	256 (KB)	512 (KB)	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)	32768 (KB)	65536 (KB)	131072 (KB)	262144 (KB)
CSS	15.08	23.72	16.70	19.45	25.03	31.80	42.57	55.58	61.69	68.02	79.97	115.65	175.01	277.30	501.91	1040.84	1943.00	3765.50
IDA	8.34	8.97	11.05	12.82	19.59	21.19	24.93	27.22	32.88	45.00	49.39	67.36	119.23	204.49	377.09	792.22	1468.66	3034.31
PSS	10.82	13.51	21.51	28.78	40.01	42.89	52.80	68.67	106.28	166.79	277.36	529.73	971.72	1902.44	3792.03	7311.64	14605.74	63156.81

Table 8. Time take in (ms) for share recreation ($n = 5$, $t = 3$).

Algorithm	2 (KB)	4 (KB)	8 (KB)	16 (KB)	32 (KB)	64 (KB)	128 (KB)	256 (KB)	512 (KB)	1024 (KB)	2048 (KB)	4096 (KB)	8192 (KB)	16384 (KB)	32768 (KB)	65536 (KB)	131072 (KB)	262144 (KB)
CSS	13.19	14.12	14.53	19.27	21.63	26.11	29.64	33.98	37.52	47.30	69.14	109.34	181.29	300.20	707.48	1106.48	2215.46	4826.05
IDA	12.69	13.13	14.71	17.82	19.51	22.57	29.58	34.79	37.39	44.34	64.87	98.49	152.63	269.23	561.52	885.50	1711.26	3466.03
PSS	13.18	14.87	19.02	30.01	20.00	23.89	29.25	35.07	40.46	60.76	89.60	145.85	265.34	498.64	906.07	1935.26	3692.24	113498.21

7. Conclusions

The modern cloud-based systems that are commonly produced are often scaled from private platforms into public cloud infrastructures, with the addition of some degree of encryption. As the data is more accessible, many systems now suffer from a loss of the symmetric key which had been used to encrypt the data. Too often a single key is used to encrypt large-scale data storage elements, where a single breach could potentially release vast amounts of sensitive information. The usage of secret share, though, may provide new and better methods of protecting data, and in providing robustness. An important element is the break-glass property, as this allows access to information within a critical incident, while similar studies conducted in [21,22] lend support to the fact that data storage using our multi-provider cloud architecture coupled with secret sharing scheme for data dissemination to these cloud storage devices has great potential in providing increased confidentiality, integrity and availability of data stored in the cloud. In all SECRET tries to lend a voice that using this scheme, a more robust Disaster Recovery system would have been implemented as a failover protection system different from the already existing Redundancy and Backup strategies [24], which we may refer to as Threshold Disaster Recovery System. Moreso, as the founding principles of SECRET, the incorporation of SeDaS for security and privacy tends to deviate from the use of encryption alone for data security and privacy implementation in cloud-based systems. All the same, this paper has shown that IDA and CSS provide the best performance in creating shares, and are ideal companions with the SECRET technique.

Author Contributions

The paper builds on patented dynamic fragmentation for non-linear authentication, authored by David Lanc. David Lanc initially collaborated with Bill Buchanan to investigate the application of dynamic fragmentation to data at rest, eliminating the need for reliance on keys. Initial research and prototyping was performed by Lu Fan and Owen Lo. This was then extended as part of Elochukwu Ukwandu's PhD research, with contributions from his supervisory team William J. Buchanan and Gordon Russell. David Lanc and Payfont Limited have further contributed to this work through commercial research focusing on high-scalability, high-resilience, architecture agnostic applications.

Conflicts of Interest

The authors declare no conflicts of interest.

References

1. Abbadi, I.M. *Cloud Management and Security*; John Wiley & Sons Ltd.: Chichester, UK, 2014; p. 2.
2. Buchanan, W.J. Storing The Keys to your House Under a Plant Pot. Available online: https://www.linkedin.com/pulse/storing-keys-your-house-under-plant-pot-william-buchanan (accessed on 1 April 2015).
3. Zeng, L.; Chen, S.; Wei, Q.; Feng, D. Sedas: A self-destructing data system based on active storage framework. In Proceedings of the 2012 Digest APMRC, Singapore, 31 October–2 November 2012; pp. 1–8.

4. Cipra, B.A. *The Ubiquitous Reed-Solomon Codes*; SIAM: Philadelphia, PA, USA, 1993; 26(1).
5. Morozan, I. Multi-Clouds Database: A New Model to Provide Security in Cloud Computing. Available online: https://www.researchgate.net/publication/273136522 (accessed on 1 April 2015).
6. Dodis, Y. Exposure-Resilient Cryptography. PhD Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, August 2000.
7. Patel, A.; Soni, K. Three Major Security Issues in Single Cloud Environment. *Int. J. Adv. Res. Comput. Sci. Software Eng.* **2014**, *4*, 268–271.
8. Padsala,C.; Palav, R.; Shah, P.; Sonawane, S. Survey of Cloud Security Techniques. *Int. J. Res. Appl. Sci. Eng. Technol.* **2015**, *3*, 47–50.
9. Mounica, D.; Rani, M.C.R. Optimized Multi-Clouds using Shamir Shares. *Int. J. Dev. Comput. Sci. Technol.* **2013**, *1*, 83–87.
10. Bharambe, J.V.; Makhijani, R.K. Secured Data Storage and Retrieval in Multi-Cloud using Shamir's Secret Sharing Algorithm. *Int. J. Comput. Sci. Eng.* **2013**, *2*, 15–19.
11. Mallareddy, A.; Bhargavi, V.; Rani, K.D. A Single to Multi-Cloud Security based on Secret Sharing Algorithm. *Int. J. Res.* **2014**, *1*, 910–915.
12. Padmavathi, M.; Sirisha, D.; Lakshman, R.A. The Security of Cloud Computing System Enabled by Shamir's Secret Sharing Algorithm. *Int. J. Res. Stud. Sci. Eng. Technol.* **2014**, *1*, 103–109.
13. Makkena, T.; Rao, T.P. A Shamir Secret Based Secure Data sharing between Data owners. *Int. J. Eng. Trends Technol.* **2014**, *16*, 362–165.
14. Beimel, A.; Chee, Y.M.; Guo, Z.; Ling, S.; Shao, F.; Tang, Y.; Wang, H.; Xing, C. (Eds.) *Secret-Sharing Schemes: A Survey*; Springer: Berlin Heidelberg, Germany, 2011; pp. 11–46.
15. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613.
16. Rabin, M.O. Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance. *J. ACM* **1989**, *36*, 335–348.
17. Krawczyk, H. Secret Sharing Made Short. In Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, CA, USA, 22–26 August 1993.
18. Resch, J.; Plank, J. AONT-RS: Blending Security and Performance in Dispersed Storage Systems. In Proceedings of the 9th USENIX on File and Storage Technologies, San Jose, CA, USA, 15–17 February 2011.
19. Rabin, T.; Ben-Or, M. Verifiable secret sharing and multiparty protocols with honest majority. In Proceedings of the Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, 14–17 May 1989.
20. Peng, K. Critical survey of existing publicly verifiable secret sharing schemes. *Inf. Secur.* **2012**, *6*, 249–257.
21. Ermakova, T.; Fabian, B. Secret sharing for health data in multi-provider clouds. In Proceedings of the 2013 IEEE 15th Conference on Business Informatics (CBI), Vienna, Austria, 15–18 July 2013; pp. 93–100.
22. Fabian, B.; Ermakova, T.; Junghanns, P. Collaborative and secure sharing of healthcare data in multi-clouds. *Inf. Syst.* **2015**, *48*, 132–150.
23. Srinivasan, S. Building Trust in Cloud Computing: Challenges in the midst of outages. In Proceedings of the Informing Science & IT Education Conference (InSITE), Wollongong, Australia, 30 June–4 July 2014; pp. 305–312.

24. Khoshkholghi, M.A.; Abdullah, A.; Latip, R.; Subramaniam, S.; Othman, M. Disaster recovery in cloud computing: A survey. *Comput. Inf. Sci.* **2014**, *7*, 39–54.