An Approach for Fault Tolerant and Secure Data Storage in Collaborative Work Environments[*]

Arun Subbiah and Douglas M. Blough
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, USA
{arun,dblough}@ece.gatech.edu

ABSTRACT

We describe a novel approach for building a secure and fault tolerant data storage service in collaborative work environments, which uses perfect secret sharing schemes to store data. Perfect secret sharing schemes have found little use in managing generic data because of the high computation overheads incurred by such schemes. Our proposed approach uses a novel combination of XOR secret sharing and replication mechanisms, which drastically reduce the computation overheads and achieve speeds comparable to standard encryption schemes. The combination of secret sharing and replication manifests itself as an architectural framework, which has the attractive property that its dimension can be varied to exploit tradeoffs amongst different performance metrics. We evaluate the properties and performance of the proposed framework and show that the combination of perfect secret sharing and replication can be used to build efficient fault-tolerant and secure distributed data storage systems.

Categories and Subject Descriptors: H.3.4 [Information Storage and Retrieval]: Systems and Software – *distributed systems, performance evaluation*; C.4 [Performance of Systems]: Fault Tolerance, Reliability, availability, and serviceability

General Terms: Security, Algorithms, Design, Performance

Keywords: Byzantine fault tolerance, collaborative environments, confidentiality, distributed data storage, replication, secret sharing

1. INTRODUCTION

The storage of sensitive information has been studied extensively in various contexts ranging from cryptographic keys [19, 4] to generic data [9]. This paper considers the problem of storing sensitive information at a storage service realized using a distributed set of storage servers. The sensitive data must be stored at these servers such that data confidentiality, integrity, and availability requirements are met even when some storage servers are compromised.

[*]This research was supported by the National Science Foundation under Grant CCR-0208655.

The traditional approach for realizing a secure and fault-tolerant storage service is to encrypt the data for confidentiality, and store the encrypted data using replication-based techniques for fault tolerance. This approach has the benefit of being computationally and storage efficient. In collaborative work environments, multiple clients may be authorized to access the encrypted data. To facilitate easy sharing of the encrypted data amongst authorized clients, the cryptographic keys used to encrypt the sensitive data must also be stored at the storage service. Obviously, these keys must be stored at the storage service in a secure and fault tolerant manner without using additional keys. A possible approach given in [9] is to store the key using perfect secret sharing schemes [19, 4]. Perfect secret sharing schemes encode data (in this case, the cryptographic keys) into *shares* such that only certain valid combinations of shares can be used to reconstruct the encoded data, while invalid combinations of shares give no information on the encoded data. By storing these shares at different servers, the encoded data is kept confidential as long as not enough servers are compromised. Confidentiality is achieved without any additional encryption, thus avoiding the need for the storage and management of additional cryptographic keys. Perfect secret sharing schemes have the additional property that the shares can be changed, or "renewed", distributively such that the encoded data still remains the same. This process of share renewal, when performed often, can provide strong data confidentiality. The security of such a scheme relies on the inability of an adversary to compromise a sufficient number of servers in the time between two consecutive share renewals.

The above approach has the drawback that the security of the encrypted data relies solely on maintaining the secrecy of the cryptographic keys. An adversary can find the key through vulnerabilities elsewhere in the system, such as in the applications used by the clients. The release of a cryptographic key to an adversary will give away the confidentiality of *all* the sensitive data encrypted using that key. Our proposed approach to overcome this drawback is to store the sensitive information itself using perfect secret sharing. Thus, the security gained by using perfect secret sharing schemes to store cryptographic keys has been transferred to the sensitive data directly. Moreover, disclosure of some of the sensitive data to an adversary will still not affect the confidentiality of the rest of the sensitive data stored at the storage service.

Unlike private-key encryption schemes, however, most perfect secret sharing schemes are computationally expensive. Verifiable secret sharing schemes [6] are typically used with perfect secret sharing schemes to detect incorrect shares that may be returned by faulty or compromised servers, and also to detect incorrect secret sharing during writes. Such techniques further increase the computation time during the encoding and decoding of data. We solve these problems by 1) using XOR secret sharing for fast computations, and 2) using replication-based schemes to detect incorrect shares that may be returned by faulty or malicious servers. This combination of secret sharing and replication manifests itself as an architectural framework, where servers are arranged in the form of a rectangle or a grid. The proposed architectural framework, which we call *GridSharing*, has the useful property that its dimensions can be varied to trade off several performance metrics.

In collaborative environments, it can be expected that there will be changes in the list of clients authorized to read or update the sensitive data. When using the traditional approach, changes in the access list will require re-encrypting the stored data with a new cryptographic key, which may be cumbersome. For fine grained access list management, each file or document stored at the data storage service would require a unique key. The number of keys could then become large and unmanageable. If the sensitive data is itself stored using secret sharing techniques, such expensive operations during changes in the access list are avoided.

Our contributions are as follows: We describe a novel approach for building a secure and fault tolerant data storage service that uses a combination of perfect secret sharing techniques and replication to provide data confidentiality, integrity, and availability. Perfect secret sharing schemes have found little use in managing sensitive data because of the high computation overheads such schemes incur especially when supplemented with mechanisms to achieve Byzantine fault tolerance. Our proposed approach uses a novel combination of XOR secret sharing and replication mechanisms, which drastically reduce the computation overheads and achieve speeds comparable to standard encryption schemes. The combination of secret sharing and replication manifests itself as an architectural framework, whose dimension can be varied to exploit tradeoffs amongst different performance metrics. We evaluate the properties and performance of the proposed framework to show that the combination of perfect secret sharing and replication can be used to build efficient fault-tolerant and secure distributed data storage systems for collaborative work environments.

2. RELATED WORK

Several works [2, 5, 14, 21, 3, 12, 24, 7] have emerged recently that consider the problem of providing secure distributed data storage services. The confidentiality of the stored data is provided either by encrypting the data with a key and storing the key also at the store using secret sharing [19, 4], or secret sharing the data itself, or a combination of both.

Most works use imperfect secret sharing schemes, such as erasure codes (e.g. Rabin's IDA [18] algorithm), where the knowledge of fewer than the threshold number of shares can reveal some information on the encoded data. Such coding algorithms are thus not information-theoretic secure, but allow savings in storage space. Given enough time, an adversary may compromise enough servers to learn the encoded data. Thus, to provide long-term confidentiality, the secret sharing scheme should allow share renewal, where the shares are changed in a distributed fashion such that the encoded secret is not recovered in the process and is unchanged. To our knowledge, no share renewal scheme for imperfect secret sharing has been developed to date. We instead use perfect secret sharing schemes, which allow share renewal. Perfect secret sharing schemes are also information-theoretic secure, meaning the leakage of an insufficient number of shares to an adversary does not reveal any information on the encoded data.

When data is stored using secret sharing, it must be possible for a client to identify corrupted shares during reads. Verifiable secret sharing schemes [6] can be used with perfect secret sharing for this purpose, and also to check if the secret sharing was performed correctly during writes. Verifiable secret sharing schemes also allow share renewal. Such schemes are however computationally expensive. Section 4 describes in detail how we avoid using verifiable secret sharing schemes, thereby drastically reducing the computation overheads. Another approach to detect corrupted shares during reads is to store the hash of the shares in a hash vector at all the servers. To our knowledge, no algorithm has been developed for updating the hash vector distributively after share renewal.

Several works have combined replication-based mechanisms and perfect secret sharing [9, 17, 15]. [9] presents a scheme where data is encrypted using a key, and both are stored at the storage servers. The data is stored in replicated form in a quorum, while the key is stored using secret sharing. [17] uses quorum systems and secret sharing to build an authorization service. Quorum properties are used to ensure that sufficient servers agree to authorize a request, but the shares are not replicated at the servers. The paper addresses malicious users, and does not consider compromised servers. The shares are never directly read and written. Thus, [9, 17] consider using perfect secret sharing for some special types of data and not for generic data. Performance during reads and writes is not addressed. [15] uses perfect secret sharing for generic data, while [23] uses perfect secret sharing for archival data. Both these works do not address the problem of high computation overheads.

In [7], a technique called fragmentation-redundancy-scattering is used. The security of this technique relies mainly on securely maintaining the encryption key and the fragmentation key. We instead store data directly using perfect secret sharing schemes.

In CODEX [16], secrets are encrypted using the public key of the data storage service. The private key is maintained at the data storage servers using secret sharing. Due to the use of expensive cryptographic operations, the computation latencies of this approach are expected to be higher than our approach. Also, the secrecy of all stored data rests upon maintaining the secrecy of only one key, which is the service's private key. The scheme is thus not as secure as storing the data directly using perfect secret sharing.

In [24], secret sharing is used to build survivable information storage systems. Tradeoffs possible when using p-m-n threshold schemes are outlined. The description is in terms of how the choice of p, m, and n affect performance. This paper not only explores the trade off space in detail, but also addresses the performance overheads involved in such schemes. However, we consider only perfect secret sharing schemes (which are a special case of p-m-n schemes), since distributed share renewal algorithms (e.g. [10]) have been developed only for these schemes.

3. BACKGROUND

3.1 Secret Sharing Schemes

Secret sharing schemes are techniques where a *secret* is encoded into several fragments, called *shares*, such that certain combinations of shares can together reveal the encoded secret. In *perfect* secret sharing schemes, invalid combinations of shares give no information on the encoded secret. Thus, perfect secret sharing schemes are information-theoretic secure. Perfect secret sharing schemes also allow share renewal, which is the process of distributively changing the shares such that the encoded secret is the same. Frequent share renewal can provide strong data confidentiality.

In perfect *threshold* secret sharing schemes, a secret is encoded into q shares such that any k out of the q shares can be used to recover the encoded secret, while any $(k-1)$ shares give no information on the encoded secret. Such schemes are also called (k, q)-threshold schemes. Shamir's scheme [19] is an example of a

(k, q)-threshold perfect secret sharing scheme, where $k \leq q$.

In the next subsection, we describe Ito, Saito, and Nishizeki's share assignment scheme [11], which realizes any access structure using a (q, q)-threshold secret sharing scheme.

3.2 Ito, Saito, and Nishizeki's Share Assignment Scheme

We describe Ito, Saito, and Nishizeki's share assignment scheme [11] for a threshold access structure. Consider a set of r participants $\{P_1, P_2, ..., P_r\}$ such that any $(m + 1)$ participants can pool their shares to recover the encoded secret. For a secret sharing scheme realizing this access structure, first list the set B consisting of all possible combinations of m participants.
Thus, $B = \{B_1, B_2, ..., B_q\}$, where $q = \binom{r}{m}$.

Next, encode the secret using a (q, q)-threshold secret sharing scheme, where $q = \binom{r}{m}$. Let the shares thus generated be denoted by $s = \{s_1, s_2, ..., s_q\}$, where $q = \binom{r}{m}$. The set of shares assigned to participant P_i is given by the function $g(i) = \{s_j, P_i \notin B_j, 1 \leq j \leq q\}$. Thus, each participant receives $\binom{r-1}{m}$ shares, and each share is stored at $(r - m)$ participants.

For example, consider a set of four participants such that at least three participants must pool their shares to find the encoded secret. Then $r = 4, m = 2$, and the set $B = \{(P_1, P_2), (P_1, P_3), (P_1, P_4), (P_2, P_3), (P_2, P_4), (P_3, P_4)\}$. Next, generate 6 shares of the secret such that all six of them are needed to decode the secret. Denote the six shares by $\{s_1, s_2, s_3, s_4, s_5, s_6\}$.

From the share assignment function g,

Participant P_1 gets shares (s_4, s_5, s_6),
Participant P_2 gets shares (s_2, s_3, s_6),
Participant P_3 gets shares (s_1, s_3, s_5),
Participant P_4 gets shares (s_1, s_2, s_4).

Thus, any two participants can pool their shares to find out only five of the six shares. Without the knowledge of the sixth share, the encoded secret cannot be found out. Any three participants can pool their shares to find out all six shares needed to recover the encoded secret.

4. COMPUTATION OVERHEAD OF PERFECT SECRET SHARING SCHEMES

In this section, we show the high computation overhead of some well known secret sharing schemes, which is the main reason why such schemes are not widely used for distributed data storage. We contrast the computation overheads with that of the Rijndael (AES) symmetric-key encryption algorithm to illustrate this point. We then show that XOR secret sharing combined with replication-and-voting mechanisms has a computational overhead similar to that of Rijndael. All performance measurements reported in this paper were done on an Intel Pentium4 3GHz processor with 256 MB RAM running Linux 2.6.9. The MIRACL [1] library was used to implement all the cryptographic algorithms. In Section 8, we also compare the communication overheads of the techniques developed against encryption and secret sharing.

Shamir's scheme [19] is an example of a (k, q)-threshold perfect secret sharing scheme, where $k \leq q$. Table 1 lists the time taken to compute shares (sharing), and the time taken to compute the secret given enough shares (recovery), for an 8 KB block of data using Shamir's scheme, for a selection of (k, q) values. Secret sharing and recovery are done during writes and reads, respectively, and their overheads are therefore important. For Shamir's scheme, since the computations are done modulo a prime p, the size of this modulus is also a factor in the throughput measurements.

Table 1: Computation times for Shamir's scheme (8 KB block)

Prime Length	$(k,q) = (3,5)$		$(k,q) = (6,10)$	
	Sharing	Recovery	Sharing	Recovery
160 bits	4.956 ms	826 μs	14.87 ms	1.446 ms
512 bits	6.192 ms	1.290 ms	20.00 ms	2.064 ms
1024 bits	10.53 ms	2.145 ms	34.65 ms	3.575 ms

Table 2: Computation times for 8 KB block using Shamir's with Feldman's scheme (Feldman's prime length = 1025 bits).

Prime Length	$(k,q) = (3,5)$		$(k,q) = (6,10)$	
	Sharing	Recovery	Sharing	Recovery
160 bits	2.461 s	2.616 s	4.956 s	7.228 s
512 bits	1.037 s	1.097 s	2.090 s	2.795 s
1024 bits	728 ms	747.5 ms	1.464 s	1.809 s

There are two attacks possible when data is stored using secret sharing techniques. One attack is by a faulty client that generates inconsistent shares during writes, i.e., different subsets of k shares out of the q shares will decode to different values. The other attack is when a faulty server returns incorrect or arbitrary shares during reads. Such attacks can be detected using verifiable secret sharing schemes [6]. In such schemes, some common data (called *witnesses*) for all the shares is computed by a client during writes and sent to all the servers. Before storing the shares and the witnesses, the servers check the shares received against the witnesses and arrive at a consensus on the consistency of the shares. During reads, a client will first determine the witnesses and check the validity of each share with the witnesses before proceeding to decode the sensitive data. Verifiable secret sharing schemes significantly increase the computation overheads during the secret sharing (encoding) and secret recovery (decoding) processes. A widely used method for verifiable secret sharing is Feldman's scheme [8]. Table 2 gives the computation times during secret sharing and secret recovery of an 8 KB block of data when Feldman's scheme is used with Shamir's scheme.

For comparison purposes, the throughputs of the AES Rijndael symmetric-key encryption algorithm are given in Table 3.

From Tables 1–3, it is clear that the computation times of Shamir's scheme and Feldman's scheme are far higher than those of symmetric-key encryption and, in fact, this performance is well below what is acceptable for modern data storage systems. The secret recovery computation time for verifiable secret sharing are at least 3000 times slower than the Rijndael decryption times. The above analyses also indicate, in part, why perfect secret sharing techniques have not been adopted for generic data to date. We reduce the computation overheads by using the following two mechanisms:

Mechanism 1: Use a (q, q) perfect secret sharing scheme: When $k = q$, i.e., all the shares are needed to recover the secret, then "inconsistent" secret sharing is not possible. That is, there is no question of different subsets of k shares out of q shares decoding to different values because there is only one such subset, since $k = q$. Hence, verifiable secret sharing schemes can be avoided. Further, a (q, q) perfect secret sharing scheme can be realized using simple bit-wise XOR operations. If each data bit is thought of as a separate secret, then each share is a single bit and XOR of the q shares (or q bits) gives the encoded secret bit. In practice, XOR secret sharing can be implemented with word-wide operations for efficiency. Table 4 lists the computation times during secret sharing and secret recovery for a selection of (q, q) values for XOR secret sharing. Note that XOR secret sharing is also a perfect se-

Table 3: Computation time for AES (CBC mode, 8 KB block).

Key length	Encryption	Decryption
16 bytes	205 μs	205 μs
24 bytes	230 μs	241 μs
32 bytes	282 μs	271 μs

Table 4: Computation times for XOR sharing (8 KB block).

(q, q)	Secret sharing	Secret recovery
$(5, 5)$	333 μs	35 μs
$(10, 10)$	732 μs	60 μs
$(20, 20)$	1.494 ms	140 μs

cret sharing scheme. The only constraint compared to the general (k, q)-threshold scheme with $k < q$ is that all q shares must be recovered to reconstruct the secret. Compared with the computation times using Shamir's scheme (Table 1), the computation times using XOR secret sharing are much lower.

Mechanism 2: Use replication-and-voting to determine incorrect shares during reads: To detect incorrect shares that may be returned by malicious servers during reads, we propose that each share is replicated at enough servers such that if at least a threshold of servers return the same share during a read, then that share is correct and can be used for the secret recovery computation. This is the traditional technique used for managing replicated data, which we apply for each share. If the number of malicious servers is denoted by b, then for each share at least $(2b + 1)$ responses must be received. The value returned by at least $(b + 1)$ servers is the correct value of the share being read.

Table 5 gives the computation times for determining each share from $(2b + 1)$ responses, where b is the number of possibly malicious servers. Note that the numbers are given for each share. Hence, the computation time during secret recovery must now include the product of the time taken to determine each share from $(2b + 1)$ responses and the number of shares. The secret sharing computation time will remain unchanged as no additional shares are generated. The secret sharing and recovery computation times for XOR secret sharing along with voting for $b = 3$ are shown in Table 6. Compared with the computation times of verifiable secret sharing schemes (Table 2), the computation times of XOR secret sharing with voting are much lower, and are in the same order of magnitude as those of the Rijndael encryption algorithm (Table 3).

Summarizing, perfect secret sharing schemes can be used for fault-tolerant and secure distributed data storage by combining them with verifiable secret sharing schemes. Using the computation latency of Rijndael as the benchmark, we have shown that well known verifiable secret sharing techniques such as the combination of Feldman's scheme with Shamir's scheme are too slow to be used for large volumes of data. The computation overheads can be drastically reduced by using instead a (q, q) perfect secret sharing scheme (namely, XOR secret sharing) along with replication-and-voting mechanisms. The computation times are comparable to those of Rijndael. In the rest of the paper, we describe in detail how XOR secret sharing with replication-and-voting mechanisms can be combined, and the benefits of this approach.

5. FAULT AND ADVERSARY MODEL

Since our data storage service must offer availability, integrity, and confidentiality guarantees for the stored data, we identify the following three types of server faults:

Table 5: Computation times for voting out of 2b+1 responses to determine a share of size 8 KB. Measurements reflect the best case where there are no incorrect responses.

b	1	2	3	4	5
Computation Time (μs)	13.75	25	40	50	65

Table 6: Secret sharing and recovery computation times for XOR secret sharing with voting (8 KB block, b = 3).

(q, q)	Secret sharing	Secret recovery
$(5, 5)$	333 μs	235 μs
$(10, 10)$	732 μs	460 μs
$(20, 20)$	1.494 ms	940 μs

- **Crash:** A server is said to be *crashed* if it stops performing all computations and neither sends nor receives messages on the network.
- **Byzantine:** A Byzantine-faulty server can deviate arbitrarily from its specified protocol. A Byzantine faulty server can also reveal the shares stored locally and its internal state to an adversary.
- **Leakage-only:** A server is said to exhibit a leakage-only fault if it can reveal its shares and state to an adversary, but executes its specified protocol faithfully.

The proposed fault model allows for direct reasoning about the availability, integrity, and confidentiality properties of the storage service. In availability attacks, such as Denial-of-Service attacks, the resources available for legitimate use of the service are constrained by, for example, limiting network bandwidth and by increasing server loads. Crash faults are a more severe form of attack, where a server stops execution completely and permanently. A storage service that can tolerate a high number of crash faults is also a highly-available storage service, and will be able to tolerate Denial-of-Service attacks to a greater degree. Integrity attacks, in the storage service model we consider, can consist of either compromising servers and altering their behavior or compromising servers and arbitrarily modifying the shares stored in them. Such attacks are represented by Byzantine faults. Confidentiality attacks can be launched only by compromising servers to obtain sufficient shares, as we focus only on the share allocation problem and not on actual protocols. These are modeled by Byzantine and leakage-only faults.

We use the threshold fault model for each of the three types of faults. We assume that not more than c servers can crash, not more than b servers can be Byzantine-faulty, and not more than l servers can exhibit leakage-only faults.

6. COMBINING SECRET SHARING AND REPLICATION

Our approach for a fault tolerant and secure data storage service is to use perfect threshold secret sharing for data confidentiality, and to use replication-based mechanisms to manage each share for crash and Byzantine fault tolerance. We first describe a straightforward method of using this approach, called the *direct approach*, and show that it suffers from requiring a large number of storage servers. We then introduce the *GridSharing* framework, where a tradeoff between the number of servers required and the storage space needed at each server is achieved. This is a worthwhile tradeoff because storage space is cheap.

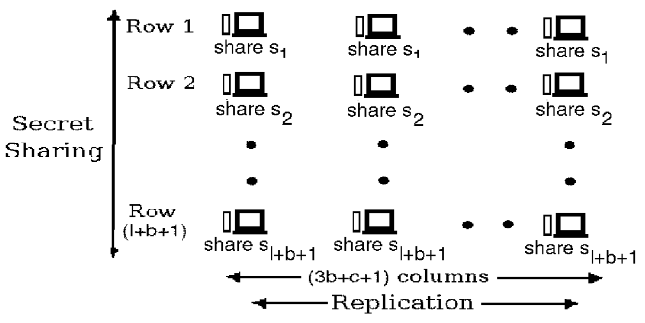

Figure 1: The *Direct Approach*: Servers are arranged in a logical grid having (l+b+1) rows, with at least (3b+c+1) servers in each row. Secret sharing is done across rows, with a distinct share assigned to each row. Shares are replicated along rows.

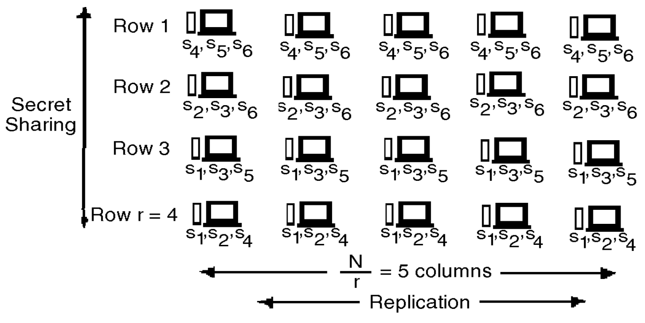

Figure 2: The *GridSharing* framework: N servers are arranged in a logical grid having r rows. Secret sharing is done across rows, and shares are replicated along rows. Setup shown for N = 20, l = 1, b = 1, and c = 6. Note that each server holds 3 shares.

6.1 The Direct Approach

We are interested in designing a share allocation scheme given N storage servers, where not more than l servers can be leakage-only faulty, not more than b servers can be Byzantine faulty, and not more than c servers can crash. The direct solution to this is to use a $(l+b+1, l+b+1)$-threshold perfect secret sharing scheme. Each share is given to a distinct set of x servers. The setup can be envisioned as the N servers arranged in the form of a logical grid with $(l+b+1)$ rows and x columns, as shown in Figure 1.

Servers in the same row store the same shares. Replication of shares is used to achieve crash and Byzantine fault tolerance. Data confidentiality is achieved using secret sharing. The secret sharing is done across rows. Thus, $(l+b+1)$ rows are required, with each share assigned to a distinct row. The compromise of any $(l+b)$ servers will give only up to $(l+b)$ shares to an adversary, but all $(l+b+1)$ shares are needed to recover the secret.

When secrets are read and written, the shares are read and written using replication-based protocols. For the purposes of this and subsequent analyses, we assume the following simple replication protocol. To write a secret S, the user generates $(l+b+1)$ shares such that their bitwise-XOR gives the secret S. The user writes to each server its assigned share. Thus, in the example depicted in Figure 1, the user will write to each server in row 1 the share s_1, to each server in row 2 the share s_2, and so on.

When the secret S is to be read at a later time, the same user or a different user will need to only contact some set of servers to read all the shares. Consider how share s_1 is read in our example. The share s_1 is stored in row 1, which consists of x servers. The user needs to contact only $(2b+1)$ of these servers to determine s_1, since only a maximum of b servers can be Byzantine faulty. The share s_1 returned by at least $(b+1)$ servers must have been returned by at least one server that is not Byzantine-faulty, and therefore should be correct. The user must obtain at least $(2b+1)$ responses to determine s_1, but up to $(c+b)$ servers can fail to return any response. Assuming clients connect to the servers over an asynchronous network so that they are unable to detect server failures, each share must be written to at least $((2b+1)+(c+b)) = (3b+c+1)$ servers for reads to be successful in the presence of b Byzantine failures and c crash failures in the system.

Thus, each share must be stored on at least $(3b+c+1)$ servers. Thus, $x = (3b+c+1)$, which gives $N \geq (l+b+1)(3b+c+1)$. Note that the given description for writes and reads is only an approach for a possible replication-based protocol to manage the shares. We have overlooked the need for the use of timestamps which are common to all the shares. All the shares must be written as part of a single write operation. The approach described is just sufficient to derive a lower bound on the number of servers required to store each share. This lower bound will change based on the assumptions on the system model and the kind of read-write semantics to be realized. The minimum number of servers needed to maintain each share is the only point in the design of the framework that is dependent on the choice of the replication protocol and its underlying assumptions.

Thus, to tolerate l leakage-only faults, b Byzantine faults, and c crash faults, at least $(l+b+1)(3b+c+1)$ servers are required for this approach. For $l = b = c = 2$, at least 45 servers are required. That is, only up to $6/45 = 13.3\%$ servers can be faulty. This is inefficient in terms of the number of storage servers required. However, the storage blowup at each server is one, as the size of each share is the same as the size of the encoded secret. Also, the bare minimum number of shares are generated, which is $(l+b+1)$. Thus, the computation times during secret sharing (writes) and secret recovery (reads) at the clients are kept as small as possible.

In the next section, we describe the *GridSharing* framework, where we balance the strengths and the weakness of the *direct approach*. We tradeoff the number of storage servers required against the storage blowup at each server and the total number of shares generated for each secret.

6.2 The GridSharing Framework

Similar to the *direct approach*, the *GridSharing* framework consists of N servers, where not more than c servers can crash, not more than b servers can be Byzantine faulty, and not more than l servers can exhibit leakage-only faults. The N servers are arranged in the form of a logical rectangular grid with r rows and $\frac{N}{r}$ columns, where for simplicity it is assumed that N is a multiple of r. The arrangement is depicted in Figure 2.

Servers in the same row store the same shares. Thus, tolerance to crash and Byzantine failures is achieved. Data confidentiality is achieved using secret sharing. The secret sharing is done across rows. Ito et al's [11] share assignment scheme is used to assign shares to the rows. Thus, as per the terminology used in Section 3.2, the r rows are the r participants amongst which shares are distributed. Since up to l servers can be leakage-only faulty (reveal their shares to an adversary) and up to b Byzantine-faulty servers can also do the same, shares from up to $(l+b)$ rows can be disclosed to an adversary. From Section 3.2, an $\left(\binom{r}{l+b}, \binom{r}{l+b}\right)$-threshold perfect secret sharing scheme can be used to tolerate $(l+b)$ faulty servers in r rows.

Figure 2 gives an example where $N = 20$ servers are arranged in a rectangular grid with $r = 4$ rows. If it is necessary to tolerate $b = 1$ Byzantine fault and $l = 1$ leakage-only fault, then a $\left(\binom{4}{2}, \binom{4}{2}\right) =$

$(6,6)$ XOR secret sharing scheme will have to be used. Assume a secret S is encoded into six shares $(s_1, s_2, s_3, s_4, s_5, s_6)$ such that $S = s_1 \oplus s_2 \oplus s_3 \oplus s_4 \oplus s_5 \oplus s_6$. That is, each bit in the secret S is the XOR of the corresponding bits in the shares $s_1, s_2, s_3, s_4, s_5, s_6$. Then according to the share assignment function g given in Section 3.2,

Servers in row 1 get shares (s_4, s_5, s_6),
Servers in row 2 get shares (s_2, s_3, s_6),
Servers in row 3 get shares (s_1, s_3, s_5),
Servers in row 4 get shares (s_1, s_2, s_4).

The choice of Ito et al's share assignment scheme is motivated by the fact that each share is assigned to multiple rows. This is in line with our principle of using replication of shares to achieve Byzantine and crash fault tolerance. Note also that, as in the *direct approach*, shares are replicated along rows. As argued in Section 6.1, each share must be stored on at least $(3b+c+1)$ servers. In the proposed framework, each share is assigned to $(r-(l+b))$ rows, and each row has $\frac{N}{r}$ servers. Thus, each share is stored at $(r-(l+b))\frac{N}{r}$ servers, and this must be at least $(3b+c+1)$. Thus,

$$(r-(l+b))\frac{N}{r} \geq 3b+c+1 \qquad (1)$$

which gives

$$r \geq \frac{N(l+b)}{N-(3b+c+1)} \qquad (2)$$

Inequality 2 gives the smallest number of rows possible for the framework. Thus, r can vary in the range $\left[\frac{N(l+b)}{N-(3b+t+1)}, N\right]$. Also, r must be greater than $(l+b)$, otherwise a Byzantine fault or a leakage-only fault in each row will give the adversary all the shares to recover the encoded data. From Inequality 2, it is obvious that the lower bound on r is greater than $(l+b)$.

For a given l, b, c, and r, Inequality 1 can be rewritten as

$$N \geq \frac{3b+c+1}{1-\frac{l+b}{r}} \qquad (3)$$

to give a lower bound on the number of servers N required. The lower bound is minimized for a given l, b, and c when r is at its maximum value, which is N. Substituting $r = N$ in Inequality 3 gives the following requirement for N for tolerating l leakage-only faults, b Byzantine faults, and c crash faults:

$$N \geq 4b+l+c+1 \qquad (4)$$

Thus, as the number of rows r is increased from $(l+b+1)$ to $(4b+l+c+1)$, the minimum number of servers required will decrease. When $r = (4b+l+c+1)$, the smallest number of servers needed to tolerate b Byzantine, c crash, and l leakage-only faults will be reached. For $r > (4b+l+c+1)$, there will be only one column, the number of servers N will be the same as the number of rows r, and N will increase with r.

7. PERFORMANCE ANALYSIS OF GRID-SHARING

7.1 Performance Metrics

This section defines some performance metrics, whose relation with the fault tolerance and security properties l, b, and c, and the number of rows r, will be described in this section.

- **min(N)**: is the minimum number of servers required for a given l, b, c, and r. This is given by the smallest N satisfying Inequality 3, with N being a multiple of r.
- **#Shares**: The total number of shares generated per secret. For the proposed framework, #Shares $= \binom{r}{l+b}$.
- **Storage Blowup Per Server**: is defined as the ratio of the storage space taken at each server to the size of the data encoded. For the proposed framework, the storage blowup factor is $\binom{r-1}{l+b}$. Since we use the XOR secret sharing scheme, the size of a share is the same as the size of the secret.
- **Secret Sharing and Secret Recovery Computation Times**: The secret sharing computation time is the time taken to generate (#Shares) shares of an 8 KB block of data. The secret recovery computation time is the sum of two components. The first component is the time taken to determine the correct (#Shares) shares from $(2b+1)$ responses for each share, where b is the Byzantine fault tolerance threshold. We assume the best case where there are no incorrect servers when evaluating this component. The second component is the time taken to compute the data block once the correct (#Shares) shares have been determined. The size of the data block and each share are 8 KB. The measurements were taken on a Pentium4 3GHz computer with 256 MB RAM running Linux 2.6.9. All measurements were performed in memory and involved no disk and network I/O.

7.2 Effect of Grid Dimension

For given security and fault tolerance thresholds l, b, and c, the performance metrics can be traded off against each other by varying the number of rows r in the framework. The secret sharing and recovery computation times are dependent on #Shares, which is dependent on r and $(l+b)$. The smaller the number of rows r, the fewer the number of shares (#Shares), and the lower the computation times during secret sharing and secret recovery. But if r is increased from $(l+b+1)$ to $(4b+l+c+1)$, from Inequality 3, the minimum number of servers required will decrease. Thus, the number of rows affects $\min(N)$ and the secret sharing and recovery computation times in opposing ways. For $l=2$, $b=2$, and $c=2$, the tradeoff space is given in Table 7.

Table 7 shows that increasing the number of rows from $(l+b+1)$ reduces the minimum number of servers required for that configuration while increasing the number of shares, #Shares, needed to store each secret. The storage capacity required at each server thus increases with r. Increasing #Shares will also increase the computation overheads at the users during the secret sharing and secret recovery processes. The practical range of r is thus limited by the storage blowup and the computation overheads.

When there are five rows in the framework, each row gets a distinct share (which is, the *direct approach*). The number of shares (#Shares) generated is minimum, and the computation times are small. But 45 servers are required for this configuration. By having 7 rows in the framework, the minimum number of servers required is lowered by more than half to 21 servers. For given fault tolerance and security thresholds, having fewer servers implies that a higher percentage of faulty servers is tolerated. Having fewer servers will also increase the manageability of the system. On the other hand, the storage blowup at each server increases by a factor of 15. Since storage cost is cheap, this is a worthwhile tradeoff. The computation times are also at acceptable values when $r=7$. Thus, the choice of the number of rows in the framework can be used to arrive at a suitable tradeoff point between the number of servers required, and the storage blowup and the secret sharing and recovery computation overheads.

Table 7: Effect of increasing number of rows r on performance when l = 2, b = 2, and c = 2

r	min(N)	# Shares	Storage Blowup Per Server	Computation Time	
				Secret Sharing	Secret Recovery
5	45	5	1	333 μs	160 μs
6	30	15	5	1.103 ms	490 μs
7	21	35	15	2.668 ms	1.150 ms
8	24	70	35	5.480 ms	3.020 ms
9	18	126	70	10.31 ms	6.276 ms

Table 8: Effect of increasing l on performance when b = 2, c = 2, and min(N) $\leq$ 35 servers

l	r	min(N)	# Shares	Storage Blowup Per Server	Computation Time	
					Secret Sharing	Secret Recovery
1	5	25	10	4	732 μs	310 μs
2	6	30	15	5	1.103 ms	490 μs
3	7	35	21	6	1.568 ms	706 μs
4	9	27	84	28	6.750 ms	4.084 ms
5	10	30	120	36	9.675 ms	6.120 ms

7.3 Effect of Fault Thresholds Given N Servers

In this section, we assume that 35 data storage servers are available, and investigate the relation between the fault tolerance and security thresholds l, b, and c and the performance metrics. We consider three cases. In each case, we fix two of the thresholds at two servers, and increase the other threshold from one to five servers. Tables 8, 9, and 10 show the three different cases. For each combination of (l, b, c), we fix the number of rows such that the secret recovery computation time is the smallest possible for the given configuration. Since the secret recovery computation time decreases with increasing r, for the given (l, b, c), r is set to the smallest value ($r \geq \frac{N(l+b)}{N-(3b+c+1)}$) such that $\min(N)$ is not more than 35 servers.

From Table 8, increasing the leakage-only fault threshold l leads to a tolerable increase in the storage blowup per server, while the secret sharing and recovery computation times become high for $l \geq 4$ servers. The effect of increasing the Byzantine fault threshold b, as shown in Table 9, has a more adverse effect on performance. The storage blowup per server and the secret sharing and recovery computation times increase rapidly with increasing b. Thus, to achieve a very high performance with 35 servers, only a relatively small number of Byzantine failures can be tolerated.

On the other hand, the framework can accomodate more crash failures without any substantial performance impact, as shown in Table 10. Increasing the crash fault threshold from one to five servers leaves the performance metrics mostly unchanged. The storage blowup at each server is tolerable and the computation throughputs are maintained at acceptable levels.

The examples considered above demonstrate that the framework can tolerate crash failures with little performance impact, leakage-only faults with medium peformance impact, and a limited number of Byzantine faults. The maximum number of faults that can be tolerated is given by Equation 4. Thus, given 35 servers, when $b = 2$ and $c = 2$, up to 24 leakage-only faults can be tolerated; when $l = 2$ and $c = 2$, up to 7 Byzantine faults can be tolerated; and when $l = 2$ and $b = 2$, up to 24 crash faults can be tolerated. However, practical limits on the secret sharing and recovery computation times and the storage blowup at each server are a more severe restriction on the actual range of faults that can be tolerated. Notice that, except for high values for the Byzantine fault threshold b, the secret sharing and recovery computation times are much smaller than the figures given for verifiable secret sharing in Table 2.

7.4 Effect of Fault Thresholds Given Restriction on Secret Recovery Computation Time

Since increasing l, and b in particular, can lead to a substantial increase in secret sharing and secret recovery computation times, as observed in Table 8 and Table 9, we remove the requirement of having only 35 storage servers available, and instead impose the requirement that the secret recovery computation time for 8 KB of data must be less than 1.6 ms. The secret recovery computation time is important when reads are more frequent than writes, which is often the case. A secret recovery computation time of 1.6 ms for 8 KB of data is approximately six and eight times slower than the decryption time using the Rijndael encryption algorithm for key sizes of 32 bytes and 16 bytes respectively, as was shown in Table 3.

Similar to Section 7.3, we consider three cases. In each case, we fix two of the fault thresholds at two servers, and increase the other fault threshold from one to five servers. Tables 11, 12, and 13 show the three different cases. For each combination of (l, b, c), we fix the number of rows r that gives the smallest $\min(N)$ while maintaining the secret recovery computation time to be less than 1.6 ms. Restricting the secret recovery computation time limits the number of shares (#Shares) generated, which in turn keeps the storage blowup at each server reasonable. In Table 11, the minimum number of servers required ($\min(N)$) shows a moderate increase with increasing l. When $l = 5$ servers, a total of 9 ($l + b + c$) servers out of 45 servers are faulty. That is, up to 20% of the servers can be faulty (leakage-only, Byzantine, or crash), which should be acceptable. In Table 12, the minimum number of servers required ($\min(N)$) increases rapidly with the Byzantine fault threshold b. Thus, the proposed framework is suitable for tolerating a small number of Byzantine faults.

In Table 13, the computation throughputs and the storage blowup remain the same with increasing crash fault threshold c for the example considered. With 21 servers, up to two crash faults are tolerated, and with 28 servers, up to 5 crash faults can be tolerated. Note that with 5 crash faults, a total of 9 servers out of 28 servers can be faulty. That is, up to 32% of the servers can be faulty, which is a standard property of replica management protocols that tolerate only Byzantine faults. While in this example most of the faults are crash faults, the number of servers required is reasonable.

Thus, from Tables 11, 12, and 13, low secret recovery computation times can be achieved with acceptable requirements on the number of servers and the storage blowup at each server. As ob-

Table 9: Effect of increasing b on performance when l = 2, c = 2, and min(N) $\leq$ 35 servers

b	r	min(N)	# Shares	Storage Blowup Per Server	Computation Time	
					Secret Sharing	Secret Recovery
1	4	24	4	1	267 μs	80 μs
2	6	30	15	5	1.103 ms	490 μs
3	8	32	56	21	4.315 ms	2.740 ms
4	11	33	462	210	38.88 ms	37.41 ms
5	16	32	11440	6435	3.104 sec	2.319 sec

Table 10: Effect of increasing c on performance when l = 2, b = 2, and min(N) $\leq$ 35 servers

c	r	min(N)	# Shares	Storage Blowup Per Server	Computation Time	
					Secret Sharing	Secret Recovery
1	6	24	15	5	1.103 ms	490 μs
2	6	30	15	5	1.103 ms	490 μs
3	6	30	15	5	1.103 ms	490 μs
4	7	28	35	15	2.668 ms	1.150 ms
5	7	28	35	15	2.668 ms	1.150 ms

served in Section 7.3, the number of servers required for tolerating crash and leakage-only faults is acceptable, while practical considerations will restrict the number of Byzantine faults that can be tolerated. Note that, in all the analyses, the number of rows was manipulated to arrive at the optimum configuration.

8. DISCUSSION

While the *GridSharing* framework aims to decrease the computation overheads incurred during secret sharing and recovery, the storage blowup at each server is increased, which increases the communication overhead during reads and writes. Table 14 shows the computation and communication overheads during the secret sharing (writes) and secret recovery (reads) processes for encryption, verifiable secret sharing (VSS), and the *GridSharing* framework when the fault thresholds l, b, and c are all equal to one.

The total time taken during a write operation is composed of three parts - the secret sharing operation, encryption of the shares to establish secure channels between the client and the servers, and the communication time. Similarly, the total time taken during a read operation consists of the communication time in getting the required number of shares, decrypting the shares from the secure channels, and then recovering the secret. For the communication time, it is assumed that the network bandwidth between the client and the servers is 100 Mbps. Note that our read / write protocol is a very simple one where a client writes to all servers and reads from the required number of servers. Timestamps and the use of Message Authentication Codes for providing message integrity during communication have been overlooked. It is also assumed that the client reliably gives each server its share(s) during writes, thus eliminating the need to implement a reliable broadcast protocol. The figures given serve only to compare between GridSharing, verifiable secret sharing schemes (namely, the combination of Shamir's and Feldman's scheme), and encryption.

The figures given for encryption do not take into account the overheads due to the storage and retrieval of cryptographic keys. The encrypted data need not be re-encrypted to achieve secure channels. The read and write latencies are thus very small. Only the minimum number of servers ($= (3b+c+1)$) are required, and the storage blowup at each server is one. Data storage using encryption-and-replication is hence an attractive option when performance is critical. However, as argued in Section 1, the security of the scheme relies on the secure maintenance of the cryptographic keys.

For VSS, the number of servers required is only 5 ($= (3b + c + 1)$). A $(3, 5)$-threshold Shamir's scheme is used, because up to two servers ($= (l + b)$) can leak shares to an adversary. The write and read latencies of VSS are over 213x and 346x slower than those of encryption. The secret sharing and recovery computation overheads account for over 98% of the total write and read latencies. In GridSharing, the secret sharing and recovery computation overheads are decreased substantially, while the communication overheads are increased. However, the overall write and read latencies for GridSharing are still much less than that of VSS. When the number of rows r is set to 7 in GridSharing, the write and read operations are over 7x and 13x faster than that of VSS, respectively. The number of servers required is only two more than that of VSS, but the storage blowup at each server is 15. Decreasing r in *GridSharing* decreases the read and write latencies and the storage blowup at the expense of requiring more storage servers. When $r = 3$, the write and read latencies are comparable to those of encryption, but three times more storage servers are required.

The increased storage blowup in GridSharing should not be a limitation, as storage space is cheap. The fact that large amounts of inexpensive, surplus storage are available has been exploited in other applications, such as in [20], where the surplus storage space is used to store different versions of objects for subsequent intrusion diagnosis and recovery.

Finally, we would like to note that the communication overheads when using replication-based protocols can be reduced using other techniques. In [13], the use of cryptographic hashes when reading replicated data has been shown to significantly reduce the read latency. [22] investigates the tradeoff between computation and communication overheads for several lossless compression algorithms. Cryptographic hashes and compression algorithms reduce communication overheads while increasing the computation overheads, which reinforces the need for reducing the computation overheads during the secret sharing and recovery processes.

9. CONCLUSION

This paper presents a novel approach for realizing a secure and fault tolerant data storage service in collaborative work environments. Key highlights of our work are:

- Perfect secret sharing schemes provide stronger security than encryption-based techniques, and facilitate easier sharing of data in collaborative work environments.

Table 11: Effect of increasing l on performance when b = 2, c = 2, and secret recovery computation time $\leq$ 1.6 ms

l	r	min(N)	# Shares	Storage Blowup Per Server	Computation Time	
					Secret Sharing	Secret Recovery
1	6	18	20	10	1.494 ms	640 μs
2	7	21	35	15	2.668 ms	1.150 ms
3	7	35	21	6	1.568 ms	706 μs
4	8	40	28	7	2.109 ms	928 μs
5	9	45	36	8	2.742 ms	1.196 ms

Table 12: Effect of increasing b on performance when l = 2, c = 2, and secret recovery computation time $\leq$ 1.6 ms

b	r	min(N)	# Shares	Storage Blowup Per Server	Computation Time	
					Secret Sharing	Secret Recovery
1	6	12	20	10	1.494 ms	415 μs
2	7	21	35	15	2.668 ms	1.150 ms
3	7	42	21	6	1.568 ms	1.02 ms
4	8	64	28	7	2.109 ms	1.60 ms
5	8	144	8	1	592 μs	576 μs

- Verifiable secret sharing schemes are typically used with perfect secret sharing schemes to achieve Byzantine fault tolerance. We show that verifiable secret sharing schemes incur substantial computation overheads, and are much slower than the Rijndael encryption algorithm.

- We use an (n, n)-threshold perfect secret sharing scheme, namely the XOR secret sharing scheme, for confidentiality, and manage each share using replication-based protocols for Byzantine and crash fault tolerance. The computation overheads are reduced drastically when compared to verifiable secret sharing schemes, but additional servers and storage capacities at each server are required. An example where the secret recovery computation time was only up to 6 to 8 times slower than the Rijndael decryption algorithm was given.

- We present an architectural framework, called *GridSharing*, whose dimension can be varied to tradeoff between the number of servers required, and the storage blowup and secret sharing and recovery computation times. This property was shown to be valuable in arriving at optimum configurations for different fault thresholds.

- We introduce a new fault model consisting of crash, Byzantine, and *leakage-only* faults for our analyses. We believe this new fault model will prove to be useful for analyzing works that are common to the areas of fault tolerance and security.

- For secret recovery computation times that are 6 to 8 times slower than Rijndael decryption, we show that our proposed framework provides good fault tolerance to leakage-only and crash faults with acceptable overheads. However, in practice, resource limitations place a restriction on the number of Byzantine server failures that can be tolerated.

- A rough comparison of the overheads, including read and write latencies, between encryption-with-replication, verifiable secret sharing, and *GridSharing* was given. Since *GridSharing* incurs a higher storage blowup at each server, the read and write communication overheads are higher than with VSS. Despite this, GridSharing has lower overall write and read latencies than VSS. Write and read latencies comparable to storing data using private-key encryption schemes can be achieved at the expense of requiring a greater number of storage servers.

10. REFERENCES

[1] The miracl software library. http://indigo.ie/˜mscott/.

[2] A. Adya, R. P. Wattenhofer, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, and M. Theimer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the* 5^{th} *Symposium on Operating Systems Design and Implementation*, 2002.

[3] R. J. Anderson. The eternity service. In *Proceedings of the* 1^{st} *International Conference on Theory and Application of Cryptography (Pragocrypt)*, 1996.

[4] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, 1979.

[5] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos. A prototype implementation of archival intermemory. In *Proceedings of the* 4^{th} *ACM International Conference on Digital Libraries*, 1999.

[6] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the* 26^{th} *IEEE Symposium on Foundations of Computer Science*, 1985.

[7] Y. Deswarte, L. Blain, and J. C. Fabre. Intrusion tolerance in distributed computing systems. In *Proceedings of the* 14^{th} *IEEE Symposium on Security and Privacy*, 1991.

[8] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the* 28^{th} *IEEE Symposium on Foundations of Computer Science*, 1987.

[9] M. Herlihy and J. D. Tygar. How to make replicated data secure. In *Crypto*, 1987.

[10] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Crypto*, 1995.

[11] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings of the IEEE Global Communication Conference*, 1987.

[12] A. Iyengar, R. Cahn, C. Jutla, and J. Garay. Design and implementation of a secure distributed data repository. In *Proceedings of the* 14^{th} *IFIP International Information*

Table 13: Effect of increasing c on performance when l = 2, b = 2, and secret recovery computation time $\leq$ 1.6 ms

c	r	min(N)	# Shares	Storage Blowup Per Server	Computation Time	
					Secret Sharing	Secret Recovery
1	7	21	35	15	2.668 ms	1.150 ms
2	7	21	35	15	2.668 ms	1.150 ms
3	7	28	35	15	2.668 ms	1.150 ms
4	7	28	35	15	2.668 ms	1.150 ms
5	7	28	35	15	2.668 ms	1.150 ms

Table 14: Comparison between encryption, verifiable secret sharing, and GridSharing for 8 KB of data. l = 1, b = 1, and c = 1.

Coding Scheme	min(N)	# Shares	Storage Blowup Per Server	Overhead during Writes				Overhead during Reads			
				Secret Sharing	Encryption (Secure Channels)	Comm. Time	Total	Secret Sharing	Encryption (Secure Channels)	Comm. Time	Total
Encryption	5	1	1	205 μs		3.277 ms	3.482 ms	218.75 μs		1.966 ms	2.185 ms
VSS	5	5	4	728ms	1.23 ms	13.11 ms	742.34 ms	747.5 ms	820 μs	7.864 ms	756.18 ms
GridSharing #rows = 3	15	3	1	180 μs	3.28 ms	9.83 ms	13.29 ms	61.25 μs	2.05 ms	5.898 ms	8.01 ms
GridSharing #rows = 4	12	6	3	430 μs	7.995 ms	23.59 ms	32.02 ms	122.5 μs	4.1 ms	11.796 ms	16.02 ms
GridSharing #rows = 5	10	10	6	732 μs	13.53 ms	39.32 ms	53.58 ms	207.5 μs	6.765 ms	19.66 ms	26.63 ms
GridSharing #rows = 6	12	15	10	1.103 ms	26.65 ms	78.64 ms	106.39 ms	321.25 μs	10.045 ms	29.49 ms	39.86 ms
GridSharing #rows = 7	7	21	15	1.568 ms	24.6 ms	68.81 ms	94.98 ms	469.75 μs	14.76 ms	41.288 ms	56.25 ms

Security Conference, 1998.

[13] L. Kong, D. J. Manohar, A. Subbiah, M. Sun, M. Ahamad, and D. M. Blough. Agile store: Experience with quorum-based data replication techniques for adaptive byzantine fault tolerance. In *Proceedings of the International Symposium on Reliable Distributed Systems*, 2005.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the* 9th *ASPLOS*, 2000.

[15] S. Lakshmanan, M. Ahamad, and H. Venkateswaran. Responsive security for stored data. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):818–828, 2003.

[16] M. A. Marsh and F. B. Schneider. Codex: A robust and secure secret distribution system. *IEEE Transactions on Dependable and Secure Computing*, 1(1):34–47, 2004.

[17] M. Naor and A. Wool. Access control and signatures via quorum secret sharing. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):909–922, 1998.

[18] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 38(2):335–348, 1989.

[19] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[20] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-securing storage: Protecting data in compromised systems. In *Proceedings of the* 4th *Symposium on Operating Systems Design and Implementation*, 2000.

[21] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *Proceedings of the* 9th *Usenix Security Symposium*, 2000.

[22] Y. Wiseman, K. Schwan, and P. Widener. Efficient end to end data exchange using configurable compression. In *Proceedings of the* 24th *International Conference on Distributed Computing Systems*, 2004.

[23] T. M. Wong. *Decentralized recovery for survivable storage systems*. PhD thesis, Carnegie Mellon University, 2004.

[24] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliççöte, and P. K. Khosla. Survivable information storage systems. *IEEE Computer*, 33(8):61–68, 2000.