Responsive Security for Stored Data [*]

Abstract

We present the design of a distributed store that offers various levels of security guarantees while tolerating a limited number of nodes that are compromised by an adversary. The store uses secret sharing schemes to offer security guarantees namely availability, confidentiality and integrity. However, a pure secret sharing scheme could suffer from performance problems and high access costs. We integrate secret sharing with replication for better performance and to keep access costs low. The tradeoffs involved between availability and access cost on one hand and confidentiality and integrity on the other are analyzed. Our system differs from traditional approaches such as state machine or quorum based replication that have been developed to tolerate Byzantine failures. Unlike such systems, we augment replication with secret sharing and demonstrate that such a hybrid scheme offers additional flexibility that is not possible with current schemes.

1 Introduction

Many applications need to store long-term data and retrieve it for later use. These applications range from single user applications storing personal information to multiuser collaborative applications that allow users to share stored data. Furthermore, these applications may run on computers that range from desktops to mobile and hand held devices. For example, the **Aware Home** [1] that has been built at the Georgia Institute of Technology is an information rich environment where a number of devices capture information about the residents and their activities. Such information, which could be of sensitive nature, is stored and accessed by several applications. Several characteristics of computers that execute such applications make them unsuitable for storing sensitive information. First, the devices may be resource poor and may not be able to store long-term data. Second, they can be easily stolen or compromised and hence cannot be trusted with storing data that has confidentiality and integrity requirements. Third, when data size becomes large, storage management is expensive and prone to errors. As ubiquitous applications become common, the need to store data securely will arise in environments that span the home and community.

It would be desirable to provide a data repository service for applications that store and access sensitive information. The kind of data that would be stored in the data storage service imposes several requirements on its design. First,

[*] This work was supported in part by NSF grants CCR-9988212 and ITR-0081276.

the data has to be highly availabile and quickly accessible to distributed clients. Second, access to private or sensitive data should be controlled. In particular, confidentiality and integrity of the stored data should not be compromised. Third, these requirements should be met even when some number of servers in the secure store are compromised by an adversary.

We present the design and analysis of a distributed data store that can meet both security and performance requirements. Our design integrates two well known techniques, namely replication and secret sharing to achieve this goal. Our key contribution is an architecture that provides desirable levels of security guarantees and performance by exploiting the natural tradeoffs possible between the two conflicting goals. We present protocols that allow data to be read and written, when some number of nodes can be compromised, using replication and secret sharing. To provide additional security and better performance, we integrate periodic share renewal as well as update dissemination among replicas. We present an analysis of the system that demonstrates the tradeoffs that are offered by the design.

The rest of the paper is organized as follows. In the next section, we desribe the two approaches, replication and secret sharing briefly and argue why a combination of the two offers a better solution. In section 3, we discuss related work that has been done in the past. In section 4, we describe the architecture of the secure store and dicuss the related protocols. We do a simple analysis of our system in section 5 and show the tradeoffs permitted by our design for various security and performance requirements. We conclude with a discussion about future work in the last section.

2 Approach

There are two distinct approaches for building a distributed storage service. One is the well known replication technique [6, 7]. Second approach is based on secrect sharing schemes [3, 5]. Each of these techniques focuses on optimizing different set of requirements. While replication enhances availability and increases performance by exploiting data locality and keeping access costs low, schemes based on secret sharing offer better data confidentiality when some number of nodes are compromised. We propose a scheme that integrates the two and offers the benefits of both approaches.

In a pure replication scheme, the servers are replicas of each other, storing copies of the same set of data items. When such a scheme is used, data has to be encrypted before storing it at servers to offer data confidentiality. A client which does the encryption, would hide the key from the servers, so that a compromised server cannot retrieve any information from the data stored in it. When the data so encrypted is shared among a dynamic set of clients, key management becomes an important issue. Whenever a client leaves the set of authorized clients, data has to be re-encrypted using a new key and the new key has to be distributed to the remaining clients. Furthermore, keys have finite lifetime and data has to be periodically re-encrypted using a new key. All replicated copies have to be renewed. Even if one of the servers is compromised and is under the control of an adversary, such a server could retain the old data encrypted with the old key. Thus data content could be leaked over time. Thus, even if data is encrypted with a key that only authorized clients know, storing all the information in encrypted form entirely at any server site would lead to a possibility of information leakage.

A (b, k) secret sharing scheme [3] transforms a data item into k pieces (called data shares or fragments) such that any b shares do not give any information about the data content and any $b + 1$ can be used to reconstruct the original data value. This scheme can be used to guarantee both data confidentiality and data availability when number of compromised nodes is not more than b. Secret sharing schemes offer confidentiality through access control at the servers as opposed to encryption schemes that are based on problems that are hard to compute.

Secret sharing schemes, at the expense of higher communication and computational cost, eliminate the problem of key management. Some secret sharing schemes also allow periodic renewal of shares by the servers without client participation [5]. If the adversary is limited to compromising no more than b nodes in any time-interval of certain length, say T_v units, by doing share renewal at a faster rate (more than once every T_v units), no information would be leaked to an adversary ever. Thus, secret sharing schemes are capable of guaranteeing lifetime secrecy of data content.

While it offers better confidentiality, a pure secret sharing scheme would result in high access cost. For example, consider a system of n servers. Let us assume that not more than b servers are compromised. Consider transforming a data item into n shares using a (b, n) scheme . Writing such a data item would involve contacting all n servers. Reading would involve contacting a minimum of $b + 1$ servers and upto $2b + 1$ servers when some servers are compromised. When n is very large, write cost could be significantly high even when the number of compromised servers is small.

The cost of write operations could be reduced by allowing a client to write only $2b + 1$ data shares and then generating rest of the shares from the already written ones [5]. Such a generation of each data share involves three or more rounds of $O(b^2)$ messages exchanged between $2b+1$ or more servers. Thus, it would take a lot of time for these new shares to be generated at other servers. If a data item is actively shared between two or more clients contacting different sets of servers in the system, each client would experience a high delay in seeing the writes of the other clients.

Yet another approach is one that combines data replication with key fragmentation. In this scheme, a data item is encrypted and the encrypted version is replicated across servers. The key that is to be shared among clients is fragmented using a secret sharing scheme and distributed across servers. Thus, any client that wants to access a data item would first obtain the key by contacting $b + 1$ servers and then access an encrypted copy of the data item. This is the approach taken by Herlihy et. al. [20]. An extension to this scheme is Krawzcyk's scheme [21] which secret-shares encrypted data using Rabin's information dispersal algorithm [11]. This scheme has the additional property that storage requirement is optimal. We consider this scheme as one of many secret sharing schemes that our proposed system could use.

In this paper, we propose integrating the two approaches, replication and secret sharing, to meet both performance and security requirements of applications. In our approach, servers are divided into c distinct sets, where c is typically $2b + 1$. Alternatively, the number of compromised servers that can be tolerated for a given c is $\lfloor (c-1)/2 \rfloor$. Servers in the same set store replicas of the same data shares. Initially a data item is transformed into c shares using a (b, c) secret sharing scheme and one share is written to each of the c sets, choosing one server from each set. A share is then disseminated to other servers in the same set. A client would contact $b + 1$ servers that store distinct shares to retrieve a data item. Access cost as seen by the client is low since a client needs to contact a maximum of only $2b + 1$ servers for reads and writes, even when n is large. Here, we are interested only in access cost as seen by the client and hence ignore the cost involved in disseminating shares to other servers. As a consequence, only weak forms of consistency are guaranteed for the stored data. Furthermore, higher data locality is possible because of replication of shares. Thus, the system is able to offer better data confidentiality, while offering some of the benefits of replication schemes.

Pure replication and pure secret sharing schemes are extreme cases of the generalized scheme presented here, when c is set to 1 and n respectively. By increasing c from 1 to n, the system goes from a purely relpicated one to a purely secret-shared system. Thus, by increasing c, the system offers better security against compromised servers at the cost of low availability and high access costs . The choice of c depends on the relative importance of security requirements to performance needs. Thus, integration of replication with secret sharing results in the design of a secure store that retains benefits of both these techniques and offers tradeoffs that are not possible with either technique alone.

3 Related work

Replication schemes that tolerate Byzantine faults[1] in a distributed environment has been studied well in the past. Schneider presented a generalized state machine replication approach to fault tolerance [6]. Liskov and Castro gave a practical implementation of the state machine approach for a file system that tolerated Byzantine faults [7]. They eliminated public key operations by using Message Authentication Codes and showed that the overhead of using their file system was only 3% in the normal case. This was implemented on a local area network. The solution does not scale well with large number of servers for the reason that each request involves two or more rounds of communication involving all the servers and that every pair of server nodes needs to share a secret key.

Quorum systems are popular for managing replicated data. Phalanx [2] and Fleet [8] systems were built using a quorum approach to tolerate Byzantine faults. In a quorum scheme, operations are done with sets of servers (quorums) that sufficiently overlap with each other to tolerate some number of malicious failures. Though access cost can be reduced to some extent by weakening the consistency guarantee, it can still be quite high for some class of applications. Alvisi et. al. presented a scheme to dynamically change the threshold value (the number of failures to be tolerated) based on the estimated number of faults perceived [9] and thus avoiding the use of large quorums when the actual number of compromised nodes is small.

Both these approaches do not offer data confidentiality. Encryption schemes have to be used to offer data confidentiality. Alternatively, secret sharing schemes that do not use encryption keys were developed to offer data confidentiality even when some number of nodes are compromised. Shamir gave a simple secret sharing scheme based on polynomial interpolation [3]. A number of other schemes have also been developed [10, 11, 12, 21]. Tompa et. al. [13], Feldman [14] and Pederson [15] and Krawzcyk [22] have also considered malicious corruption of individual shares either by servers or by a client.

More recently Herzberg et. al. developed a proactive secret sharing scheme where servers could proactively recover and renew their shares in a distributed manner to protect information against an adversary who can dynamically compromise nodes [5]. Our discussion here uses techniques presented in this paper, in particular, the share renewal and share recovery protocols.

Herlihy and Tygar developed a scheme where data is encrypted and the key is secret-shared [20]. Krawzcyk gave a computationally secure secret sharing scheme combining secret sharing with encryption and Rabin's information dispersal [21]. Naor and Wool gave a scheme in which access control servers are different from storage servers [23]. However, they considerd the case of benign server faults and malicious clients.

[1]A compromised server can behave in arbitrary manner and is assumed to suffer from a Byzantine failure.

The PASIS project [16] at CMU very closely relates to our work. PASIS considers various secret sharing and other schemes to store data securely in a data repository. However, PASIS does not consider combination of relpication and secret sharing. Other related works include dissemination in non-malicious environment by Demers et al. [17] and in Byzantine environment by Malkhi et al [18].

4 System Architecture and Protocols

Our secure store is implemented by a set of n servers. Clients make read and write requests with subsets of servers by sending a request message to each of the servers (and possibly getting replies for read requests). Each request is authenticated and authorized individually by every server. Hence, we assume the presence of appropriate public key infrastructure. Each client and server node has a private key for which the public key is well known. Besides these keys, clients and servers also negotiate symmetric keys periodically to exchange messages. Thus, we assume all communication channels are secure against eavesdropping, modification and replay attacks. Requests are authorized using access control lists at each server, which are updated securely and in a timely fashion by a system administrator, possibly using a separate service.

We assume a Byzantine fault model [4] for servers, where a compromised node can behave arbitrarily. Any compromised node can disclose data content of the data shares it holds, corrupt the shares and possibly collude with other compromised nodes. Both system architecture and protocols are designed to tolerate a certain number of Byzantine faults. This number, denoted by b, is referred to as the threshold value of the system in this paper. For the rest of the section, we assume that a threshold value b has been chosen and that in any time interval of length T_v units, not more than b servers are compromised. We refer to the constant T_v as the vulnerability window.

The set of n servers is arranged logically in a two-dimensional matrix as shown in figure 1. The server in i^{th} row and j^{th} column is denoted by S_{ij}. Number of columns is denoted by c and number of rows r with $rc = n$. We assume that c is at least $b + 1$. A data item is fragmented into c shares using a (b, c) secret sharing scheme and is stored as shares at the servers. For a particular data item, servers along a column store copies of the same data share and each column stores a different share. Both security and performance levels change with values chosen for b and c, as we will see in section 5.

The operation of our secure store is characterized by the following three protocols:

1. Read and write protocols that are used by clients to access the data

2. A dissemination protocol which is used by the servers to exchange new data shares

3. A share renewal protocol that is run periodically to generate new data shares for a data item

We do not discuss in this paper how data consistency is guaranteed when clients access shared data items. Our store is capable of offering two kinds of weak consistency guarantees, namely Monotonic Read Consistency and Causal Consistency. We showed in [*] how such consistency can be achieved in a similar system. The read and write protocols presented here can be modified in a similar fashion to offer these consistency guarantees, with some amount of modification on the server side.

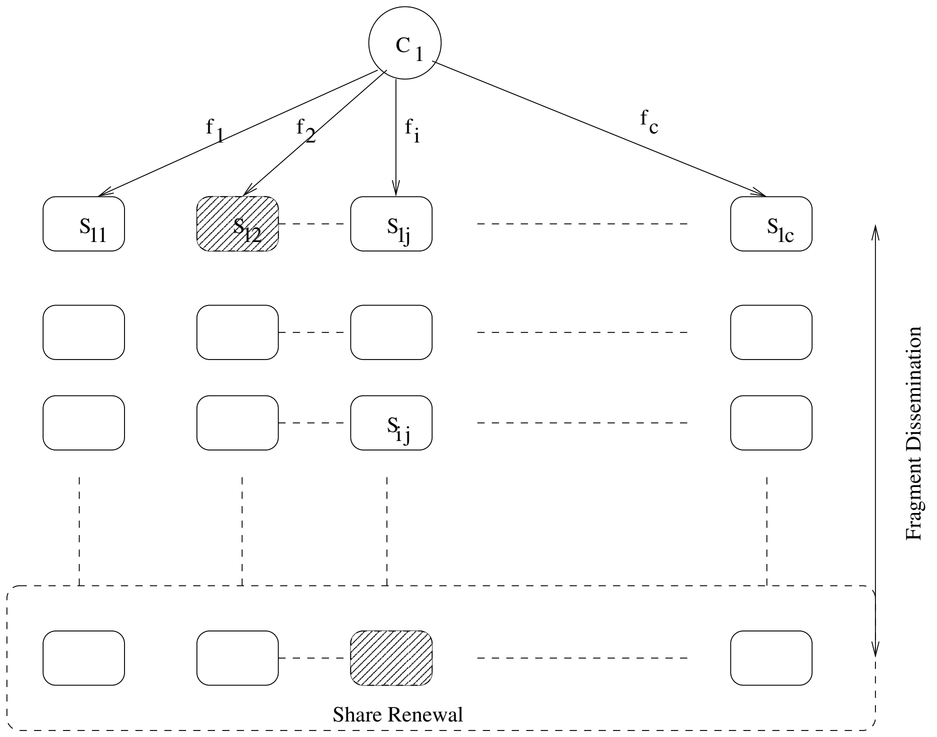


Figure 1. Secure Store Architecture

In this paper, we use a specific verifiable secret sharing scheme due to Feldman [14] for our protocols. The use of this scheme, as discussed in [5], results in (a)easy verification of a share during dissemination, (b)renewal of shares in a purely distributed fashion, and (c)regeneration of lost shares. We could have used any secret sharing scheme that has these properties and we are currently exploring other schemes for faster computation and storage space conservation. We discuss in the following subsections, the protocols for reading and writing, dissemination and share renewal.

4.1 Read and Write Protocols

We do not consider the case of clients being malicious. While malicious clients cannot do any harm to data items for which they do not have access, they can, however, write garbage to the data items for which they have write access. They can also make other clients read inconsistent values or exhaust a server's memory resources. We rely on detecting malicious clients and using authorization and access control mechanisms to stop malicious clients from doing harm to the system. Client side intrusion is addressed in [19] where Strunk et al. have explored the use of self-protecting disks to log, audit and detect a compromise.

Figure 2 shows the write and read protocols when all clients are expected to be non-malicious. In a write operation, a client fragments a data item into c shares using a (b, c) secret sharing schemes. One-way functions $h(x)$ are computed for these shares and concatenated to form a verification string. This is required to let a server know if a share it received in dissemination has been corrupted. Verification string also helps a reading client choose the right set of $b+1$ shares to reconstruct the data. For write, the shares are sent to servers along a row, each receiving a different share along with uid of the object, timestamp, the verification string and the signature of the whole write. Since some servers that are contacted can be compromised ones, the write is repeated with different rows until the number l of rows contacted is such that $c - \lfloor b/l \rfloor$ is at least $b + 1$. Since maximum number of columns in which all l servers contacted are compromised is $\lfloor b/l \rfloor$, writing to l rows ensures that in each of $b + 1$ or more columns, at least one non-malicious server has received the write mesage.

The one-way function used in Feldman's scheme is g^x where g is a primitive element in the field[2] from which values for x are chosen. This is secure based on the assumption that discrete logarithm in a finite field is difficult to compute. While this works for our solution, this is costly in terms of storage space required (verification string is c times as large as the data item) and in terms of computation (exponentiation) for verification during dissemination. In the next section, after discussing dissemination, we will list the desired properties of an one way function that could be used in our protocols.

While reading, a client sends a read request for the object to servers along a randomly chosen row. It collects $b + 1$ or more shares corresponding to the same timestamp and reconstructs the data value. Finally it verifies the signature before accepting the value. If the read is not successful, the client contacts additional servers. Variations of this protocol are possible by varying the number and choice of servers a client contacts initially and by varying the way additional servers are contacted. For example, a client could contact additional servers one by one looking for the missing shares. We have given a simple protocol here for the sake of clarity.

4.2 Dissemination

Our write protocol writes each share to only one server in a column. To provide better performance and availability, shares written at one set of servers should be disseminated to other servers so that the data is available for access at other servers. Hence, in the secure store, shares are disseminated along columns. Data dissemination for non-malicious environments was studied in [17]. Presence of malicious servers requires a more careful treatment.

Byzantine nodes can modify the data being disseminated and thus can compromise the availability and integrity of the data. There are two approaches to do dissemination in the presence of Byzantine faults. One approach is to attach an unforgeable signature of the writing client to the data being disseminated. We disseminate data shares rather than the data item itself. Since shares change over time, due to periodic share renewal (dicussed in section 4.3), the client has to recompute the signature of the shares. However, it is a desirable feature to do share renewal without requiring client participation, as we will see in section 4.3.

The second approach (referred to as the fan-in approach) is to write a data item to at least $2b + 1$ servers initially and requiring that other servers accept a data item as valid only if they hear from $b + 1$ servers. Malkhi et. al. have analyzed this scheme for dissemination in a Byzantine environment in [18]. The fan-in factor ($b + 1$) determines how fast a data item can be disseminated. A fan-in of 1 corresponds to the case where faults are considered to be non-malicious.

[2]For the purpose of this paper, the data shares and values for data items are assumed to be chosen from the finite field Z_p, for an appropriate prime p.

```
**Write**($x_j, v$) **by client** $C_i$
  1. Let timestamp $ts$ = current clock value concatenated with $uid$(client).
  2. Fragment value $v$ into $c$ shares $v_1$,$v_2$,..$v_c$ using a $(b, c)$ secret sharing scheme.
  3. Compute one-way function of each of the shares, $h(v_i) = g^{v_i}$,
     where g is a chosen primitive element of the field.
  4. Form the verfication string $VS$ and compute signature
       $VS = h(v_1)|h(v_2)...|h(v_c)$. (concatenation).
       $sig = \{uid(x_j), ts, v\}_{K_{c_i}^{-1}}$,
       where $\{\}_{K_{c_i}^{-1}}$ denotes signature using private key of the client.
  5. Choose a row $k$.
     for ($m$ = 1 to $c$)$\{$
       send $\{$"write",uid($x_j$),$ts$,$v_m$,$VS$,$sig\}$ to server $S_{km}$.
     $\}$
  6. Repeat 5 for a different row until $c - \lfloor b/l \rfloor \geq b + 1$ where $l$ is
     the number of rows contacted.
**Read**($x_j$) **by client** $C_i$
  1. Choose a row $k$.
     for $m$ = 1 to $2b + 1$ $\{$
       send $\{$"read",$uid(x_j)\}$ to $S_{km}$.
     $\}$
  2. Receive a list of timestamps from each of the server with the corresponding
     data share and verification string.
  3. A timestamp is said to be "good" if it appears in at least $b + 1$ replies (lists) and
     the corresponding verification strings are the same.
     Let $t_r$ be the highest timestamp among such good timestamps.
  4. If there is no good timestamp, repeat from 1 for a different k.
  5. Pick shares corresponding to this good timestamp. Pick $b + 1$ shares among these
     that are successfully verified by the verification string.
     Reconstruct the data value.
  6. Check if signature is valid. If valid, return the reconstructed data value.
     If signature is not valid, repeat from 1 for a different $k$.
```

Figure 2. Write and Read protocols

We cannot adopt the fan-in approach directly to our scenario for the reason that each share is written to only one server initially. We use a combination of the two approaches mentioned above. We replace signature verification by a set of one-way function verifications. Thus, a server verifies a data share f it receives in dissemination by

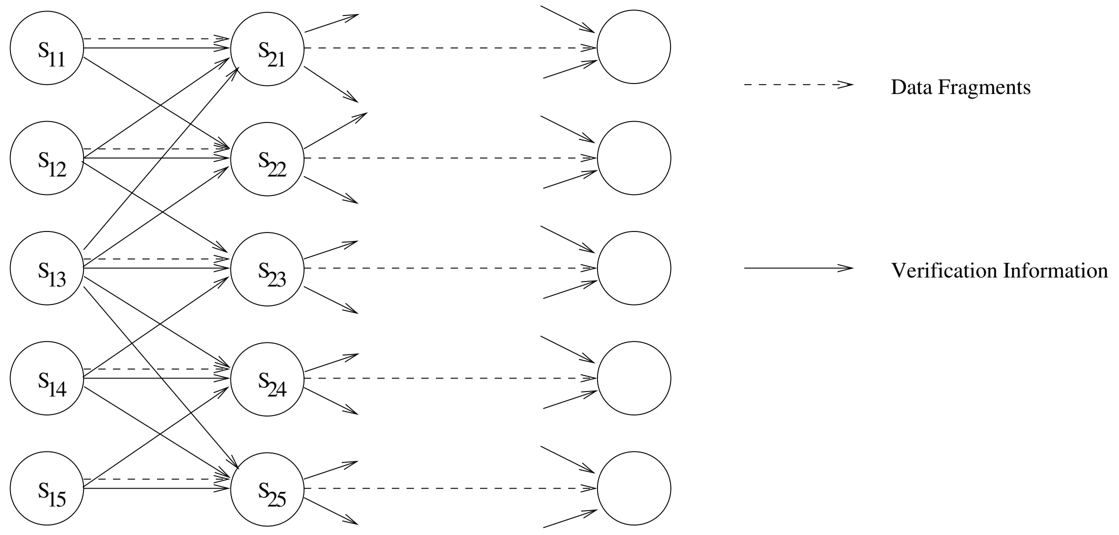


Figure 3. Secure Dissemination

checking it against the one-way function of the share $h(f)$. The verification string, which is a concatenated list of one-way functions of the shares of a data item, should be fully reliable since a corrupted verification string can verify a manipulated share to be correct. Hence, verification string itself is written by the client along with shares and disseminated to other servers securely using the fan-in approach. We require that the verification string be disseminated across columns, among all servers. A server accepts a verification string as valid only if $b+1$ or more servers report the same verification string for a given timestamp. Once a server accepts a list of one-way functions as valid for a timestamp, it accepts the corresponding share by verifying that the one-way function applied on that particular share matches the corresponding part of the verification string. This way, a compromised server cannot modify and disseminate a corrupted share without going undetected, even when colluding with other malicious servers.

Hence, our dissemination protocol works in two parts: (1) dissemination of verification string, (2) dissemination of shares. Figure 4 describes the dissemination protocol. In addition to dissemination of shares and verifiction string, we add a component to detect corrupted shares and regenerate correct shares. Share recovery involves getting secondary shares from $b+1$ or more other servers, each from a different column, to recover its share. This scheme is described in [5]. Once the right share is constructed, this share is disseminated along the column. Share recovery is costly and we expect that, if there are only few malicious servers in the system, share recovery would be done only infrequently, .

During share renewal, the shares of a date item change, but, the new verification string can still be computed securely and reliably, even in the presence of active attackers during the phase of share renewal [5]. This is due to the fact that the one-way function, g^x, is homomorphic. Thus, even after share renewal, we can disseminate the new shares securely as we did before share renewal.

In place of the function g^x for $h(x)$, we could use any one-way function that satisfies the following properties.

1. It should be hard to invert and finding a collision should be computationally infeasible.

Dissemination of verification string

1. During a write operation, client writes VS to all servers it contacts(at least 2b+1).

2. A server accepts a VS as valid only if it hears from the client directly or when b+1 other servers report the same VS for a write.

3. Servers disseminates valid VS to all other servers, both within and across columns. A server cannot serve VS to clients or other servers before validating the VS.

Dissemination of shares

1. A server disseminates the shares it receives to other servers in the same column.
Shares do not have to be verified before disseminating to other servers.

2. A server accepts a share as valid if the share is successfully verified by a valid VS.
If a share is successfully verified, that share is marked as verified.

3. A server can serve a share to clients even before verification.

Detecting corrupted shares and regenerating correct shares

1. A server detects corruption if it receives two or more different shares for the same write.

2. A server can probabilistically suspect corruption of the shares it holds.

3. Upon detection or suspicion, a server uses a valid VS to decide about the validity of the shares.
If the server does not have a valid VS, it pulls VS from other servers.

4. If a server finds a share to be corrupted, and does not have the correct share,
it initiates a share recovery protocol with servers from $b + 1$ or more other columns.

Figure 4. Dissemination protocol

2. It should be computable in share renewal phase. In share renewal phase, new verification string is computed from old verification string and the information communicated to every one participating in the protocol.

3. It should be space efficient, in particular, optimized for the case of a single large data item being broken into smaller blocks before fragmentation for secret sharing.

4. Verification should be fast, optimized for the case of a single large data item, as before.

Properties 3 and 4 are required when we implement secret sharing of a large data item by breaking it into smaller blocks and then fragmenting the blocks further using a secret sharing scheme. The function used for the discussion in this paper, g^x, does not satisfy properties 3 and 4 for the reason that verification string is c times as big as the shares themselves and that verification involves exponentiation. We are currently exploring possible alternative one-way functions that can be used in place of g^x.

4.3 Share Renewal Protocol

We assume that an adversary cannot compromise more than b nodes in any time frame of length T_v . However, this does not prevent an adversary from compromising $b + 1$ or more nodes over a longer period of time and obtain $b + 1$

or more shares to learn the content of a data item. Hence, the shares need to be periodically renewed, in a distributed manner, so that an adversary who obtains b shares before share renewal cannot use them in any manner in future to gain any information, even if he finds additional data shares. The share renewal of a data item is done without the participation of the client that wrote it. This enables share renewal even when the client is offline for an extended period of time. In this paper, we use the share renewal protocol proposed by Herzberg et. al. for Feldman's secret sharing scheme as discussed in [5].

For a given data item, servers belonging to one row initiate a share renewal protocol. At the end of share renewal, the shares are renewed while the data content and other meta-data are retained. The verification string is also updated securely for the new shares. The protocol guarantees that if the number of non-faulty servers is a majority, at the end of the protocol, all non-faulty servers hold valid shares and a copy of the same valid verification string. From then on, the new shares are disseminated as before. No data share is stored at any non-malicious server beyond T_v seconds after it's renewal. A share is erased either when a new share arrives or when the share expires. This is critical to the confidentiality of a data item since share renewal schemes rely on erasing the old shares.

For a threshold value of b, at least $2b+1$ servers are needed for share renewal. The above protocol can be modified to work even if some among these $2b+1$ servers have copies of the same shares. This would be useful if c is less than $2b+1$. However, for any good choice of r, c and b, we expect the number of columns to be at least $2b+1$. Herzberg et. al. [5] assume a secure broadcast channel for share renewal protocol. In our system, we could dedicate a set of servers for this purpose, on a single shared wire, doing share renewal for different data items.

5 Analysis

In this section, we do an analysis of the secure store based on a probabilistic model and show how the choice of a threshold value and other parameters affect the security and performance of the system. During any continuous time interval of length T_v units, we assume that any server can be compromised with a probability p. Thus, expected number of compromised servers during a time innterval of length T_v would be np. However, a lower or higher value can be chosen for b to tolerate certain number of failures depending on whether better performance or better security is desired. We assume that the probability of compromising one node is independent of the other. Thus, in the analysis, we do not consider the case of related or similar attacks on nodes operating on same OS or run time code. The analysis we provide here is a simplified one and its goal is to provide us with insights into how some parameters affect security and performance levels offered by the system. In particular, it gives us a direction as to what threshold value should be chosen and the degree of replication and secret sharing, given the desired levels of confidentiality, integrity, availability and access cost.

We consider the following security metrics.

- **Availability** : Availability is defined as the probability that a legitimate client can read a data item that has been written successfully.
- **Confidentiality** : Confidentiality is defined as complement of the probability that an adversary can read a data item that has been written successfully.

- **Integrity** : Integrity is defined as complement of the probability that any client could be given changed or corrupted data content when a read on that data item is done.

We assume that the servers are organized in c columns and r rows with $rc = n$, where n is the total number of servers. Furthermore, we assume that a (b, c) scheme is used for secret sharing. With these assumptions, the security metrics can be evaluated as follows:

Availability(α):

$\alpha(b, c, n)$ = probability of finding at least $b + 1$ non faulty servers, one each from a different column.

$$\alpha = \Sigma_{i=b+1}^{c} \binom{c}{i} (1 - p^r)^i * (p^r)^{(c-i)}$$

Confidentiality(κ):

$\kappa(b, c, n)$ = 1 - probability of finding at least $b + 1$ malicious servers, one each from a different column.

$$\kappa = 1 \ - \ \Sigma_{i=b+1}^{c} \binom{c}{i} (1 - q^r)^i * (q^r)^{(c-i)}$$

where $q = 1 - p$.

For our system, integrity is same as confidentiality because of the assumption that any node that is compromised can both disclose data shares and corrupt them. If use of signature for verification of a data content is considered, integrity becomes the probability that a signature can be forged, analyzing which is beyond the scope of this paper. Hence, in rest of the section, we discuss only confidentiality and availability as the primary security metrics.

In addition to these security metrics, for our system, we also define the following performance metrics.

- **Read cost** : Read cost is defined as the expected number of servers a client needs to contact to read a data item successfully. A data item is read successfully when the client is able to collect $b + 1$ distinct shares for the data item.

- **Write cost** : Write cost is defined as the number of servers a client needs to contact to write a data item at a confidence level h. By confidence level, we mean the probability that a write has been successfully completed which happens when at least one non-faulty server from each of $b+1$ or more columns has registered the write.

Both read and write costs are defined in terms of number of servers contacted. We expect the communication cost to be the dominant factor compared to the computation cost and hence discuss only communication costs in this section. This may not be true when we consider large data items. Secret sharing and data reconstruction are costly for such items. Furthermore, computation cost also increases with threshold value. Hence, for large data items, it is important that threshold value is kept as low as possible. Number of messages sent/received during a read or write operation is twice the number of servers contacted to complete the operation.

Read cost is calculated based on the following protocol: A client contacts $2b + 1$ servers, each from a different column. If it has successfully collected $b + 1$ shares, read returns. Otherwise, client contacts additional servers as necessary. If p_r is the probability that a client would find $b + 1$ non-faulty servers among the $2b + 1$ it contacts, then expected number of servers a client needs to contact is approximately $(2b + 1)/p_r$.

When a client does a write, it writes to some number of rows so that the probability of finding at least one non-faulty server from each of $b + 1$ or more columns is greater than or equal to h where h is the confidence level. If $p_w(k)$ is the probability of a write being successful when a client writes to k rows, then write cost is $\mathbf{wc} * c$ where $\mathbf{wc}$ is such that $p_w(\mathbf{wc} - 1) < h$ and $p_w(\mathbf{wc}) \geq h$.

Given the probability of a node being compromised and the total number of servers, there are two parameters that determine the values of the security and performance metrics. These are the threshold level b and the degree of replication which is the number of rows or alternatively the number of columns c.

We calculated the four metrics by varying these two parameters for a system of 45 servers with p = 0.15. Graphs 4(a), 4(c) and 4(e) in figure 5 show availability and confidentiality. Both availability and confidentiality are plotted in logarithmic scale. For a value x plotted in the graph, the corresponding probability (availability or confidentiality) is $1 - 10^{-x}$. Graphs 4(b), 4(d) and 4(f) show read and write costs. For graphs 4(a) and 4(b), number of columns was varied for fixed values of threshold. For graphs 4(c) and 4(d), threshold value was varied for fixed values for number of columns. For graphs 4(e) and 4(f), threshold value was varied and number of columns was set to $2b+1$ accordingly.

We observe the following dependencies :

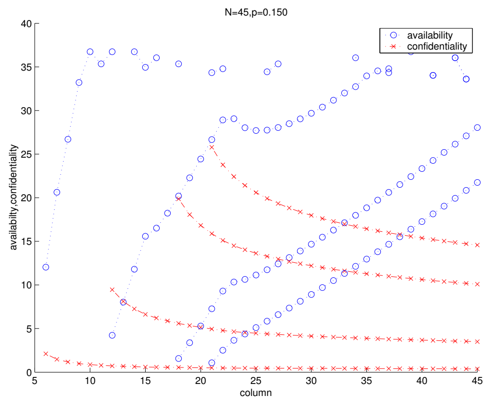
- For a given threshold level, increasing the number of columns (and hence decreasing the number of rows) increases availability and decreases confidentiality. Write cost increases linearly with number of columns but read cost remains almost a constant.

- For a given number of columns (and rows), increasing the threshold value decreases availability and increases confidentiality. Read cost increases linearly with threshold. Write cost remains almost a constant for low threshold levels. As threshold value approaches the number of columns, write cost starts increasing.

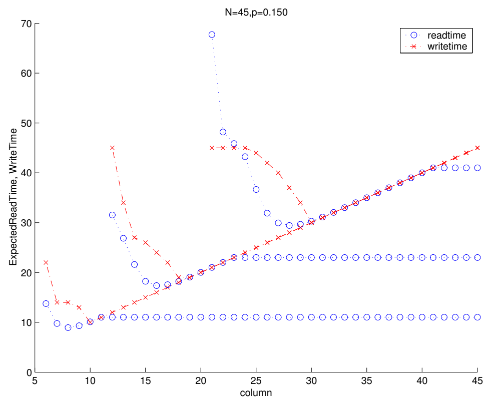
- By choosing c value optimally for every b value, increasing b increases access costs and confidentiality and decreases availability.

We can see from the plots in figure 5 that an hybrid replication and secret sharing scheme provides more flexible design options for the secure store. Given p, the likelihood of a node being compromised, the degree of replication and secret sharing (e.g, number of rows and columns in the logical matrix) depends on the security and performance metrics that need to be optimized. Clearly, when access cost or availability is of paramount importance, pure replication ($r = n, c = 1, b = 0$) is the best option. On the other hand, when confidentiality is the critical metric, pure secret sharing ($r = 1, c = n, b = \lfloor (c-1)/2 \rfloor$) is the best option. There is a wide range of confidentiality, availability and access costs where the deisred levels are achieved when the server nodes are arranged in a certain number of rows and columns. Thus, both replication and secret sharing are essential when certain bounds are placed on the security and performance metrics. For example, with a confidentiality requirement of $\kappa \geq 1 - 10^{-3.5}$, when access cost should not exceed 22 servers, the optimal choice would be $b = 10, c = 21$.

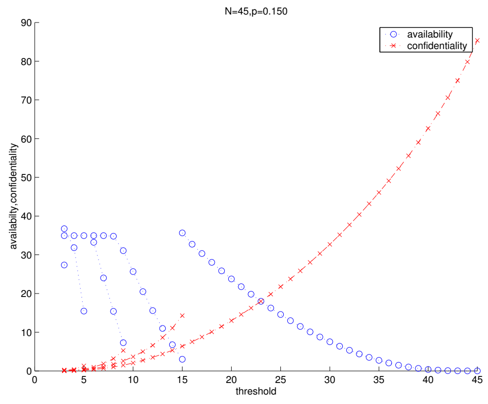
Apart from the flexible tradeoff our system offers between security and performance, one additional benefit in our system is tolerance to related attacks. When a subset of nodes that run same implementation version on same OS are vulnerable to related attacks, by placing them in the same column, effort required to steal information is not any easier.



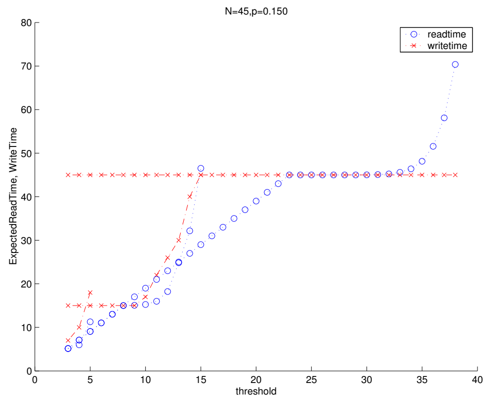
(a)



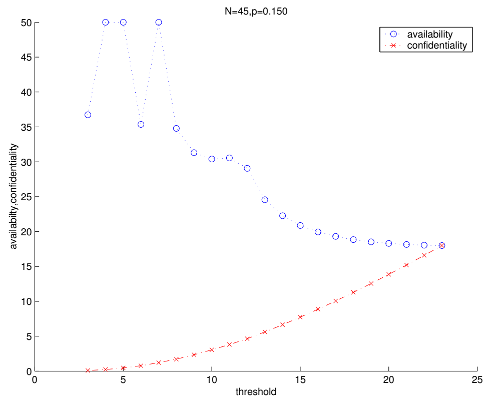
(b)



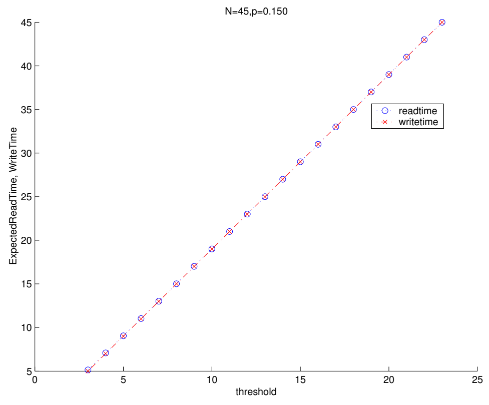
(c)



(d)



(e)



(f)

Figure 5. (a) α,κ as functions of number of columns for various threshold values b = 1,11,17,20, (b) access costs as functions of c for various threshold values b = 1,11,20, (c) α,κ as functions of threshold value for various c values, c = 3,5,9,15,45, (d) access costs as functions of threshold value for various c values c = 5,15,45, (e) α,κ as functions of threshold value for $c = 2b + 1$, (f) access costs as functions of threshold value for $c = 2b + 1$.

Thus, our analysis demonstrates that our hybrid scheme offers greater flexibility in meeting performance and security goals of a secure store.

6 Conclusion

In this paper, we have presented a design for a distributed store that integrates replication and secret sharing to meet both performance and security needs while tolerating Byzantine faults. By our simple analysis, we have shown that for a probabilistic model of Byzantine failures, our design provides a greater flexibility in meeting security and performance needs compared to either a pure replication or a pure secret sharing scheme. Although we used specific secret sharing and share renewal schemes for the discussion in this paper, the design of our system does not depend on any specific choice for these schemes. We could use any secret sharing or share renewal scheme that satisfies certain requirements, as appropriate for our protocols.

There are several directions in which our research can be pursued further. First, we are currently experimenting with and exploring other possible secret sharing schemes and one-way functions to use in dissemination. We plan to extend the existing system to provide flexible security guarantees, customizable on a per-object basis. Currently, our system provides the same level of security and performance for all objects in the store. Dynamic inclusion and exclusion of servers, secure authorization service, support from intrusion detection systems to realize the assumption we make about Byzantine failures, are some interesting problems to explore. In the future, we also plan to prototype a secure store and evaluate it experimentally.

References

[1] The Aware Home Research Initiative, http://www.cc.gatech.edu/fce/ahri/, 2000.

[2] Malkhi D., Reiter M., “Secure and Scalable Replication in Phalanx”, *Proc. 17th IEEE Symposium on Reliable Distributed Systems*, October 1998.

[3] Shamir A., “How to Share a Secret”, *Communications of the ACM*, 22, 1979, pp.612–613.

[4] Lamport L., Shostak R., Pease M., “The Byzantine Generals Problem”, *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[5] Herzberg A., Jarecki S., Krawczyk H., and Yung M., ”Proactive secret sharing,” *Advances in Cryptology, Crypto ’95*, Lecture Notes in Computer Science, 963, D. Coppersmith, Ed., 1995, pp. 339–352, Springer-Verlag.

[6] Schneider F., “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial”, *ACM Computing Surveys*, Vol. 22, No. 4, December 1990.

[7] Castro M., Liskov B., “Practical Byzantine Fault Tolerance”, *Proc. 3rd Symposium on Operating Systems Design and Implementation, New Orleans*, February 1999.

[8] Scalable and Survivable Data Replicatiion : The Fleet Project, http://www.bell-labs.com/user/reiter/fleet/.

[9] Alvisi L., Malkhi D., Pierce E., Reiter M., Wright R., “Dynamic Byzantine Quorum Systems”, *Proc. International Conference on Dependable Systems and Networks*, June 2000.

[10] Blakley G., ”Safeguarding Cryptographic Keys”, *Proc. Nat’l Computer Conf.*, American Federation of Information Processing Societies, Montvale, N.J., 1979, pp. 313-317.

[11] Rabin M., “Efficient dispersal of information for security, load balancing and fault tolerance, *Journal of the ACM*,36(2):335-348, 1989.

[12] A. De Santis and B. Masucci, ”Multiple Ramp Schemes,” *IEEE Trans. Information Theory*, July 1999, pp. 1720-1728.

[13] Tompa M.and Woll H., ”How to Share a Secret with Cheaters,”, *Journal of Cryptology*, Feb. 1988, pp. 133-138.

[14] Feldman P., “A practical scheme for non-interactive verifiable secret sharing”, in *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, IEEE Press, 1987, 427-437.

[15] Pederson T.P., “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”, *Proc. Crypto’ 91*, LNCS 576, pp. 129-140.

[16] Wylie J.J,, Bigrigg M.W., Strunk J.D, Ganger G.R., Kiliccote H., and Khosla P.K., “Survivable information storage systems”,*IEEE Computer*, 33(8):61-68, August 2000.

[17] Demers A., Greene D., Hauser C., Irish W., Larson J., Shenker S., Sturgis H., Swinehart D, and Terry D., “Epidemic algorithms for replicated database maintenance”, *Proc. 6th Symposium on Principles of Distributed Computing*, pp. 1-12,1987.

[18] Malkhi D., Mansour Y., and Reiter M., “On diffusing updates in a Byzantine environment”, *Proc. 18th IEEE Symposium on Reliable Distributed Systems*, October 1999.

[19] Strunk J.D, Goodson G.R., Scheinholtz Michael L., Craig A. N. Soules, and Ganger G.R., “Self-securing storage: Protecting data in compromised systems”, *Operating Systems Design and Implementation*, San Diego, CA, 23-25, October 2000, pages 165-180, USENIX Association, 2000.

[20] Maurice P. Herlihy and J.D.Tygar, “How to make replicated data secure”, *Crypto ’87*.

[21] Hugo Krawczyk, “Secret sharing made short”, in *Advances in Cryptology – CRYPTO ’93*, D. R. Stinson, ed., Lecture Notes in Computer Science 773 (1994), 136-146.

[22] Hugo Krawczyk, “Distributed fingerprints and secure information dispersal”, *12th ACM Symposium on Principles on Distributed Computing*, Ithaca, NY, 1993.

[23] Moni Naor and Avishai Wool, “Access Control and Signatures via Quorum Secret Sharing”, *IEEE Trans. Parallel and DIstributed Sys* 9(9), 1998.