Secure Group Services for Storage Area Networks

Yongdae Kim
University of Minnesota - Twin Cities
kyd@cs.umn.edu

Fabio Maino
Andiamo Systems
fmaino@andiamo.com

Maithili Narasimha
University of California Irvine
mnarasim@ics.uci.edu

Gene Tsudik
University of California Irvine
gts@ics.uci.edu

Abstract

Storage Area Networks, with their ability to offer high data availability, reliability and scalability, are a promising solution for the large scale storage needs of many enterprises. As with any distributed storage system, a major design challenge for a Storage Area Network (SAN) is to provide data integrity and confidentiality. In this paper, we propose a solution which addresses these core security requirements. In particular, we focus on mechanisms that enable efficient key distribution to allow scalable data sharing. Our scheme uses strong cryptographic techniques to achieve data security and integrity. Further, we delegate the bulk of the cryptographic processing to the SAN entities (e.g., switches, routers or other network elements), thereby removing bottlenecks at the disks and causing minimal inconvenience to the hosts. By recognizing the peer nature of the group of SAN entities, we propose efficient group key mechanisms that do not involve any centralized servers. This allows our scheme to be scalable and be free from any single point of failure or attack.

Keywords: *Storage area networks, secure storage, key management, group key agreement*

1 Introduction

Continued growth and popularity of the Internet fuels increased reliance on e-business which often involves data-intensive applications. Consequently, the amount of information that needs to be stored and managed can become quite intimidating. Traditional centralized servers with SCSI interfaces to peripheral storage devices, which have been the workhorses of the industry, are often unable to meet the storage needs of large organizations. To this end, they are being replaced by network attached disks and, more recently, by *Storage Area Networks* (SAN-s). SAN-s provide efficient any-to-any connectivity between hosts and storage devices and represent a major step in the evolution of network storage.

A critical requirement in any distributed (e.g., storage) system is the security and integrity of stored data. Although this problem has been studied intensively in the past, certain unique features of the SAN setting result in some new security challenges. In this work, we concentrate on safeguarding data (stored on a SAN) from various threats and attacks with the further emphasis on efficient key management. Specifically, we propose a security architecture for preserving privacy and integrity of SAN data. We use on-disk as well as on-wire encryption to protect data from unauthorized insiders and malicious outsiders. Only authorized SAN entities and hosts (through these authorized SAN entities) can gain access to the unencrypted disk contents. We employ cryptographically secure hashing and digital signatures to provide data integrity. Our approach, in addition to providing strong security, offers good systemwide performance since the bulk of the cryptographic operations are relegated to SAN entities (which are typically equipped with powerful processors and/or hardware acceleration to support wire-speed encryption).

This paper is organized as follows: In section 1.1 we discuss prior work in storage security. Section 1.2 outlines our contributions. We describe the details of our system model, trust assumptions and specify design goals in Section 2. Section 3 introduces the cryptographic primitives used in the paper. The actual architecture is introduced in Section 4. Section 5 describes two key management schemes for SAN entities. Section 6 discusses the security and the performance of the proposed system and we conclude in Section 7.

1.1 Prior Work

Security in storage systems has been an area of active research for well over a decade now. Several systems have been proposed and analyzed, e.g., CFS[1], SFS-RO[2], Cepheus[3], and NASD[4]. Prior results vary widely with respect to trust assumptions and security primitives/services offered. For example, one of the earliest systems, Cryptographic File System (CFS) [1], is tailored towards single-user workstations and relies on user-supplied passwords for data encryption. In contrast, Network-Attached Secure Disks (NASD) [4] proposes a distributed system comprising of intelligent disks and uses user supplied keys as proofs of authorization. The recent work of Reidel et. al. [5] investigates the level of security offered by various cryptographic storage systems and provides a framework for evaluating them.

We can separate previously proposed secure storage systems into: those which only focus on protecting data in transit, those that attempt to safeguard the data while stored on disk, and those which provide end-to-end protection (both on-wire and on-disk). Approaches where the underlying storage server is trusted, (e.g. NASD and SFS [6]) focus mainly on securing network traffic and preventing *outsider* attacks. Other approaches like Cepheus and SNAD [7] do not trust the storage servers and, therefore, propose security measures to protect data in transit as well as on disk. We follow the latter approach and aim to provide both on-wire and on-disk data protection.

Many of these storage systems provide mechanisms for efficient group sharing of data. In other words, identical data access permissions are given to groups of users, and any user who can prove group membership is authorized to access data based on the group permissions. Group sharing reduces the total number of keys to be stored and distributed in the system. These group keys are typically used to secure the symmetric keys that are used for data encryption (e.g., *group lockbox* in Cepheus). Systems such as SNAD and NASD rely on centralized group servers to distribute these group keys. Although a centralized server simplifies key distribution, it is a single point of failure and represents an enticing target for attacks. Our system does not require any centralized entities, however, it provides efficient group key sharing. Similar to SNAD, we also store these lockboxes on the storage system itself to enable efficient key retrieval.

1.2 Contributions

The notable features of our approach are as follows:

- We delegate the bulk of the cryptographic operations to the SAN entities (switches, routers or other network devices) essentially freeing the hosts from the cryptographic burden. A host only needs to establish a secure link with the SAN perimeter. (This can be easily achieved using a password-based mechanism such as SRP [8], for example. Also, in some settings, such as the Storage Service Provider model, the perimeter SAN entities are within the security boundary of the data owner). Thus, the level of user inconvenience introduced by our system is minimal.

- Since SAN entities are entrusted with all cryptographic duties, key management only involves these (relatively) few entities, as opposed to involving all possible hosts in the system.

- We recognize the peer nature of the SAN entities that are authorized to virtualize a secure volume. Exploiting this feature, we propose two different key management approaches: one based on a simple Public Key Infrastructure (PKI). The second, more novel, approach is based on peer group *key agreement* techniques.

2 System Model

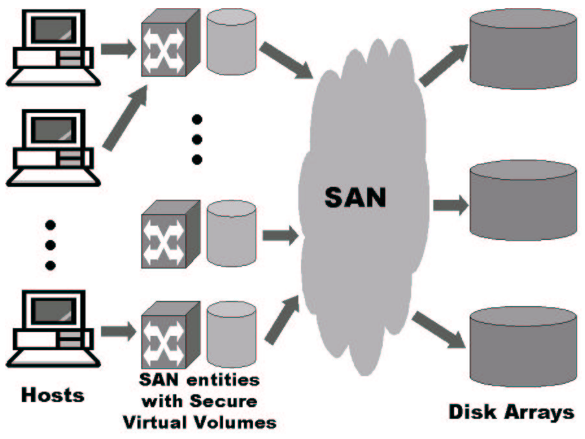


Figure 1. System Architecture

A *Storage Area Network*, as seen from the perspective of a host or a disk, is a network infrastructure that forwards, in an efficient and reliable way, both data blocks and commands required to retrieve and store these data blocks on disks. The SAN infrastructure is essentially a collection of network devices such as high-end switches for IP or Fibre Channel networks or storage routers. We use the term **SAN entities** to refer to these network devices. The SAN entities, typically, are equipped with powerful processors with the equivalent computational power of a high-end PC. Further, these devices are secured within the data center where access control is tightly enforced.

In most SAN frameworks the actual data owner can control, fully or in part, the SAN administration. This is the case in a typical enterprise scenario as well as in the Storage Service Provider (SSP) model (where SSP companies sell storage as a service to their customers).

In fact, in an enterprise scenario, user data is protected according to the security policy established by the enterprise and enforced within the enterprise SAN. As an example, data availability is ensured by applying a backup policy that is enforced in the SAN itself, without the user intervention (i.e., disks where user data are stored are mirrored, replicated or backed up under the control of the SAN administrators, and not of the users). In the same way, data integrity and privacy should also be ensured and enforced by the SAN administrators, in accordance with the security policy of the enterprise.

Even in the SSP model, the SAN is at least partially under control of the data owner. The SAN entities located on customer premises (that provide connectivity with the provider's part of the SAN) are, in fact, managed and controlled by the customer's administrators.

The fact that the data owner controls the SAN enables the powerful concept of virtualization for data security: the SAN entities can actively enforce data security policies by encrypting and decrypting on-the-fly blocks of data that are written to, or read from, the storage subsystem. In practice, the host sees the remote disk as a secure virtual volume with security attributes transparently provided by the SAN. The mapping between the secure virtual volume, the physical disk(s) where data is actually stored, and the security parameters/transforms applied to the data is performed by the SAN entities controlled by the data owner.

Since the SAN entities are responsible for the active enforcement of data security, there is the need to effectively protect them from unauthorized access on their management interface. This is a well known problem addressed by management architectures (such as SNMPv3 [9]) and is out of scope of this paper. We assume that management access to the SAN entities is governed by a strong access control mechanism ensuring that only authorized storage administrators can modify the configuration of parameters for a secure virtual volume.

2.1 System Events

The various events that take place in our model are summarized below:

1. Initialization: a storage administrator, through a management action in one of the SAN entities, creates a new secure virtual volume mapped over one or more physical disks. The initial encryption keys for that volume are chosen.

2. Disk Access: a disk read or disk write event is triggered when a host accesses a secure virtual disk through a SAN entity.

3. Key Update: occurs when the encryption key needs to be changed. This can be prompted by: a) compromised key(s), b) compromised SAN entity, or c) Periodic key refresh. (We use the term *Compromised SAN entity* to describe an entity that has been removed from the SAN for one of the following reasons: changes in network topology, reallocation of SAN resources for performance optimization or administrative reasons, or because the SAN entity was subverted. We assume that it is possible to detect subverted SAN entities and propagate this information to other SAN elements, e.g., by using an Intrusion Detection system.)

4. SAN entity Join: triggered when a secure virtual volume is instantiated for the first time by a new SAN entity (e.g., because a host connected to that SAN entity attempts to access that volume).

Assumptions and Scope of this Paper We assume that the storage subsystem (essentially, the set of disk arrays) is not intrinsically trusted by the data owner, whereas, some of the SAN entities are trusted. In other words, the data owner controls the hosts and part of the SAN but not necessarily the physical disk array. There are many solutions for the authentication and authorization between the host and the SAN (e.g., the iSCSI[10] architecture proposes a password-based approach for the authentication) and we do not address these issues here.

The use of block-level encryption is independent of the physical organization of the storage subsystem. Thus, data redundancy and high availability can be provided by the usual approach followed on disk arrays (it may be structured with RAID organization that better addresses the requirement in terms of resiliency to catastrophic failures of the disk drives). For example, a secure virtual volume may be physically mapped over a disk array organized as a RAID 5 storage subsystem, without compromising robustness. We note that even the backup strategy is completely unaffected by block-level encryption, since traditional backup strategies can be applied to the physical disks over which a secure virtual volume is mapped.

Other aspects out of scope of this paper have to do with the actual mechanisms used to provide integrity and encryption mechanisms used on data blocks of a secure virtual volume. Many well-known block encryption and data integrity protection algorithms can be applied[1]. We also do not consider the actual authentication mechanisms used between the SAN and the storage subsystem.

2.2 Design Goals

We now outline the key design goals for the secure SAN architecture.

Data Encryption: on-wire encryption is needed to protect the data against passive eavesdroppers. Disk encryption is necessary since disks are assumed not to be under direct control of data owners. (Also, keeping data encrypted on disk does not influence the ability to perform backups and other administrative functions.)

Data Integrity: providing data integrity is another key goal. In order to maintain the integrity of data, we use digital signatures and keyed MAC-s. These cryptographic tools make it possible to detect unauthorized data alterations. However, a malicious storage server or an attacker can still illegally modify the data (by means of a block substitution attack). As we noted before, safeguarding the integrity of stored data from such attacks requires certain special techniques (such as MAC trees).

Data Sharing: data privacy alone is not a challenging goal. Individual files can be encrypted by the owner with the keys of its choosing. However, this would be inefficient and impractical when files need to be shared among multiple users in a large distributed computing environment. Hence, effective key management is an important design goal to enable data sharing.

Granularity of Protection: the amount of data encrypted with a single key influences the overhead of storage and management of cryptographic keys. Of course, it is possible to naïvely use a single key to encrypt an entire virtual volume. However, if this key needs to be changed, the overhead of re-encrypting the whole volume would be staggering. In general, coarser granularity leads to fewer keys at the expense of costlier re-keying, whereas, finer granularity results in a potentially large number of keys.

Performance: incorporating security should not be prohibitively expensive. System performance is a crucial issue. For example, in case of key compromise, all keyholders must be notified and all affected data must be re-encrypted with the new key. This must be possible without significant overhead even if the volume of the data in the system is very large.

[1]We note that it might be important to choose one that provides protection against blocks substitution attacks, whereby the attacker substitutes an entire block of encrypted data with a previously encrypted value observed on the same block. Approaches such as MAC trees or the incremental MAC technique suggested in [11] might address this problem.

3 Cryptographic Building Blocks

In order to support secure sharing of files among a group of SAN entities – without relying on any centralized entity – public key cryptography is a natural choice due to its simple key management. At the same time, the use of public key cryptography must be minimized because of its relatively high cost. Therefore, a two-tiered approach is often used: bulk data is encrypted using a fast symmetric cipher such as AES [12] and the symmetric encryption keys are themselves encrypted under the public keys of all authorized SAN entities. (We refer to this below as the Encrypted Master Key approach.) One very viable alternative is to encrypt symmetric (bulk data encryption) keys under a single group key known only to all authorized SAN entities. This can be achieved through the use of a secure group key agreement mechanism [13]. Both approaches are discussed in detail in the remainder of this paper.

3.1 Notation

Before we introduce both approaches, we summarize the notation used throughout the rest of the paper.

S_j	j^{th} SAN entity; $j \in \{1, \ldots, k\}$
V_i	i^{th} virtual volume; $i \in \{1, \ldots, l\}$
PK_j	Public Key of j^{th} SAN entity
SK_j	Private Key of j^{th} SAN entity
N_j	Secret random number chosen by j^{th} SAN entity
EK	Data encryption key
$A \hookrightarrow B$	Save A in B
$A \vert\vert B$	Concatenation of A and B

3.2 Public Key Cryptography

In a typical public key cryptosystem (PKCS), a party A has a public key and a corresponding private key. A public key PK is used to encrypt messages (and/or verify signatures) while the corresponding private key SK is used to decrypt ciphertext (and/or sign messages). A public key is, as the name suggests, widely available and only its authenticity is required to guarantee that A is the only party who knows the corresponding secret key [14]. Examples of PKCS-s include RSA [15] and ElGamal [16].

3.3 Group Key Agreement

Group key agreement [17] is a process whereby a shared secret key GK is jointly computed by a group of users.[2] Fundamental properties of group key agreement include:

- a group key GK generated as $f(N_1, N_2, \ldots, N_k)$ where $f()$ is a one-way function and N_j (for $0 < j \leq k-1$) is a key share randomly and uniformly chosen by each group member.
- no information about GK can be extracted from a protocol run without knowledge of at least one of N_j
- all key shares are kept secret, i.e., if a member j is honest then, even if all other parties collude, they cannot extract any information about N_j from their combined view of the protocol.

Many group key agreement protocols – such as [19], [20] and [21] – support only secure group creation, i.e., they enable a static group of users to share a key. Recently, some of these protocols have been extended to handle group membership changes (GDH [13], STR [22] and TGDH [17]). We are particularly interested in the following features:

Condition 1: provide efficient mechanisms for *member join* and *member evict* events (in order to add a new member or expell a member) and

Condition 2: not require all current group members' contribution for join and evict operations (since all members may not be available/active at a given time).

We do not consider the Burmester-Desmedt protocol [21], since it does not satisfy condition 2 above, i.e., a join or evict operation requires all current members to update their key shares [13]. This puts an undue burden on the members (SAN entities). SSDW [19] and STR are not considered since they are quite inefficient in handling group eviction events, requiring, at worst, $O(k)$ exponentiations (where k is the total number of members). GDH is equally expensive for both evict and join events. (Recent results on the performance of practical group key agreement protocols can be found in Amir, et al. [23].) Therefore, we focus on the TGDH protocol which requires, on the average, only $O(\log k)$ modular exponentiations to handle any group event.

3.4 TGDH Protocol

TGDH is a group key agreement technique combining Diffie-Hellman key exchange [24] with key-trees [25]. It implements fully distributed contributory group key agreement and handles key adjustments due to group membership changes and periodic re-keying needs. A group key is derived from the individual contributions of all group members using a binary key-tree. TGDH assumes reliable communication among group members for protocol correctness and fault-tolerance (but not for security). In the SAN setting, however, instead of counting on the presence of a reliable group communication system, we use shared storage to maintain the group key-tree. Below, we describe the protocol in detail. (A detailed description of TGDH appears in Kim, et al. [17].)

[2] Group *key distribution* [18] mechanism also enables a group of users to share a secret key. However, it requires a centralized server which becomes a single point of failure or corruption.

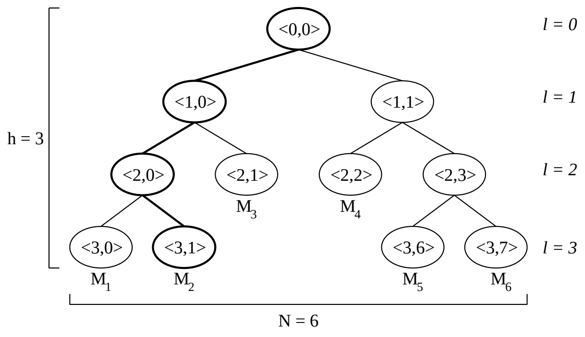


Figure 2. TGDH Key-Tree Notation

A group key-tree is organized in the following manner: each node $\langle l, v\rangle$ is associated with a key $K_{\langle l,v\rangle}$ and a corresponding blinded key (bkey) $BK_{\langle l,v\rangle} = g^{K_{\langle l,v\rangle}} \bmod p$ where g is a generator of a subgroup of $\mathbb{Z}_p^*$ and p is a large prime. This shared key-tree includes **only blinded keys**. All keys – including the root key and the members' invidual contributions – are private to each member.

Figure 2 shows a key-tree example. The key at the root node is the secret group key shared by all members, and a key at the leaf node is the member's contribution. (Each leaf node is associated with a distinct member M_i.) Every member knows all keys on the path from its leaf node to the root as well as all bkeys on the key-tree. Each key $K_{\langle l,v\rangle}$ is computed recursively as follows:

$$\begin{aligned} K_{\langle l,v\rangle} &= (BK_{\langle l+1,2v+1\rangle})^{K_{\langle l+1,2v\rangle}} \bmod p \\ &= (BK_{\langle l+1,2v\rangle})^{K_{\langle l+1,2v+1\rangle}} \bmod p \\ &= g^{K_{\langle l+1,2v\rangle} K_{\langle l+1,2v+1\rangle}} \bmod p \end{aligned}$$

Clearly, computing a key at $\langle l, v\rangle$ requires knowledge of the key for one of the two children and the bkey of the other. For example, in Figure 2, M_2 can compute $K_{\langle 2,0\rangle}, K_{\langle 1,0\rangle}$ and $K_{\langle 0,0\rangle}$ using $BK_{\langle 3,0\rangle}, BK_{\langle 2,1\rangle}$,

$BK_{\langle 1,1\rangle}$, and $K_{\langle 3,1\rangle}$. Following each group membership change, a particular member (called a sponsor) recomputes all affected keys and bkeys and updates the shared key-tree file.[3] Note that the role of a sponsor is unique to each membership event.

In general, the insertion point for a join event is the shallowest rightmost node, where the join would not increase the height of the key-tree. Otherwise, if the key-tree is fully balanced, the new member node is joined to the root. The sponsor is the rightmost leaf node in the subtree rooted at the insertion node. Figure 3 depicts the join protocol.

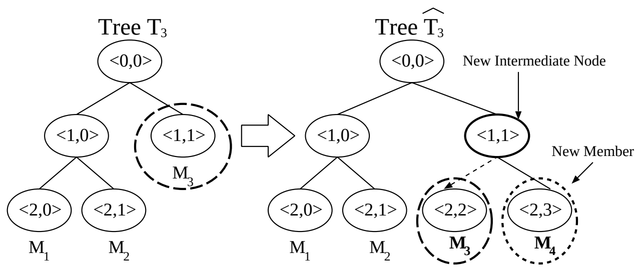


Figure 4. Tree update: join

Figure 4 shows an example where M_4 joins a group and the sponsor (M_3) performs the following actions:

1. renames node $\langle 1,1\rangle$ to $\langle 2,2\rangle$
2. creates new intermediate node $\langle 1,1\rangle$ and new leaf node $\langle 2,3\rangle$
3. promotes $\langle 1,1\rangle$ as parent node of $\langle 2,2\rangle$ and $\langle 2,3\rangle$

Since all members know $BK_{\langle 2,3\rangle}$ and $BK_{\langle 1,0\rangle}$, M_3 can compute the new group key $K_{\langle 0,0\rangle}$.

Member eviction is similar to a join. The key-tree is updated by deleting the leaf corresponding to the evicted member (M_d). The former sibling of M_d is promoted to replace M_d's parent node. The member performing the update (sponsor) generates a new key share, computes all $[key, bkey]$ pairs on the path up to the root, and reveals the updated key-tree (containing a new set of bkeys) to the rest of the group. Thereafter, only current members can compute the new group key; equivalently, outsiders (including evicted former members) cannot compute subsequent group keys. The sponsor is always determined as the rightmost leaf of the subtree rooted at the evicted member's sibling.

TGDH protocol is proven secure under the well-known Decision Diffie-Hellman (DDH) assumption [26]. For more details of the proof and the actual protocol, please refer to Kim, et al. [27].

[3]We stress, once again, that the shared file contains only bkeys.

We need to adapt the TGDH protocol for the SAN environment. In its original form, TGDH relies on the presence of a reliable group communication system to notify the group of all membership changes and to provide reliable and sequenced protocol message delivery. However, in the present environment, two (related) issues arise: 1) reliable group communication requires constant on-line presence of all current members, and 2) network partitions and congestion may cause membership changes, i.e., group membership is dependent on the state of the network. In a SAN, membership in a group of SAN entities is a long-term concept and should not be influenced by *short-term* network perturbations. Therefore, group membership changes should not be triggered by the instantaneous reachability or availability of members, but, instead, by explicit and infrequent events such as: a new member being introduced into the group (join) or a current member being expelled (eviction) from the group.

As a consequence of the above, some of the heuristics of TGDH need to be amended. In particular, sponsor selection must change since on-line presence of the sponsors (as defined above) cannot be assumed.

We observe that the original sponsor selection criteria aims to keep the key tree as balanced as possible. However, in case of a join event, any node can be the sponsor. Many possible courses of action are possible; the simplest one is for the first available member to "lock" the shared key tree (thus claiming the sponsor role) and perform the update by either inserting the new leaf as described above or at the root.

Member eviction is a little more involved. As stated earlier, the normal sponsor for an evict event is the rightmost leaf in the subtree rooted at the evicted leaf's sibling. In fact, we note that any leaf in that subtree is actually capable of performing an eviction-triggered key-tree update. Furthermore, any leaf in a subtree rooted at the evicted leaf's grand-parent is likewise capable.[4] To avoid contention, we can employ the same approach as in member join, i.e., any leaf in either subtree can "lock" the key-tree in shared storage and perform the update.

Clearly, unless the group is a singleton, there exists at least one remaining leaf in the subtree rooted at the grand-parent (or parent) of the evicted leaf. However, it is conceivable that all leaves in that subtree are currently unavailable, e.g., off-line. In that unlikely case, the update becomes more complicated. Specifically, any other leaf can perform the key-tree update as follows:

1. Let M_d be the evicted leaf and let $Node_g$ be M_d's grand-parent. Let M_i be a leaf NOT in the subtree rooted in $Node_g$.

[4]We state these properties without proof to conserve space.

Step 1: New member broadcasts request for join along with its bkey.

$$M_{n+1} \xrightarrow{BK_{\langle 0,0\rangle}=g^{N_{n+1}}} \{M_1,\ldots,M_n\}$$

Step 2: sponsor member

- update key-tree by adding a new member node and a new intermediate node,
- generate new share and compute all $[bkey, key]$ pairs on the path from node associated with it to the root,
- save the updated tree $\widehat{T_s}$ only with bkeys on the shared file.

Figure 3. Join Protocol

2. M_i replaces the leaf formerly labeled M_d with its own label, M_i.
3. M_i moves the entire subtree rooted at $Node_g$ into the place previously occupied by M_i itself.
4. Since the previous step results in $Node_g$'s former parent having only one child, M_i compacts the key-tree by promoting its own ancestor to replace $Node_g$'s former parent.
5. Finally, M_i changes its key share, re-computes all necessary keys and bkeys, stores the new key-tree and releases the lock.

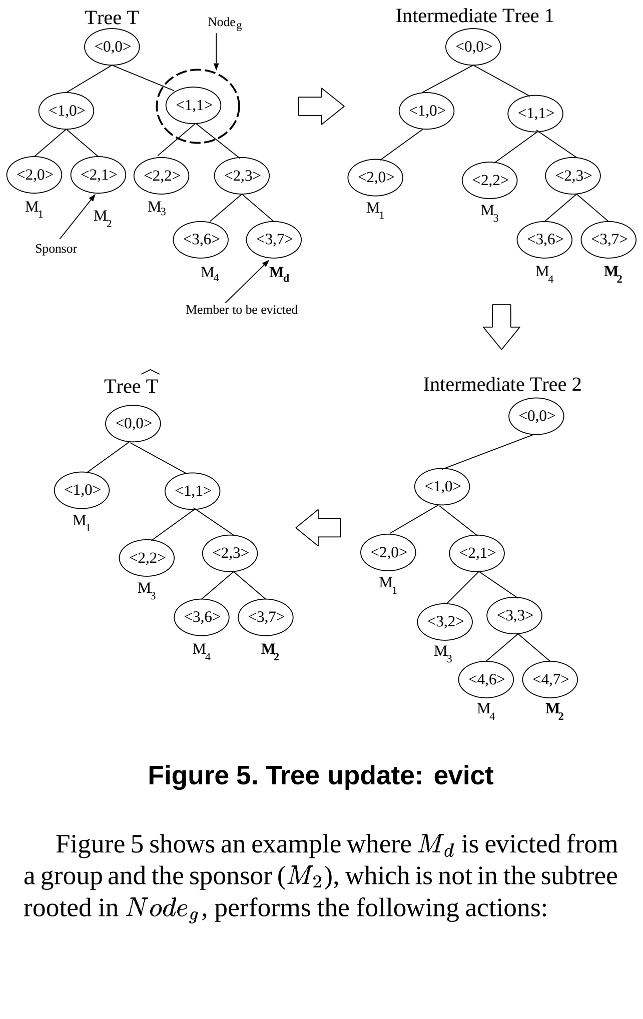


Figure 5. Tree update: evict

Figure 5 shows an example where M_d is evicted from a group and the sponsor (M_2), which is not in the subtree rooted in $Node_g$, performs the following actions:

1. renames node $\langle 2,1\rangle$ to $\langle 3,7\rangle$ (Intermediate Tree 1)
2. renames node $\langle 1,1\rangle$ to $\langle 2,1\rangle$ (Intermediate Tree 2)
3. Since the previous step results in $\langle 0,0\rangle$ having only one child, M_2 compacts the key-tree by promoting its own ancestor $\langle 1,0\rangle$ to replace $\langle 0,0\rangle$ (Final key-tree).
4. Finally, M_2 changes its key share, re-computes all necessary keys and bkeys, stores the new key-tree and releases the lock.

4 Architecture

The simplest solution to safeguard data in the SAN setting is to encrypt all data in a virtual disk with a single key and make sure that only authorized SAN entities know this key. The SAN entity that encrypted the data can also store a digital signature along with the encrypted data. This would help maintain data integrity. This solution is simple to implement and the associated storage overhead is minimal. However, it is clearly impractical, since changing the key would require re-encryption of the entire virtual disk data, which can be very expensive.

One straight-forward enhancement (to improve performance) is to divide each secure virtual disk into multiple logical segments or *Encrypted Data Units* (EDU-s) [5]. Consequently, data in each EDU can be encrypted with a separate key and a key change operation would cause only the data in the relevant EDU(s) to be re-encrypted. The size of individual EDU-s and, therefore, the total number of EDU-s on a disk can be either fixed or variable. To simplify things, we assume that the EDU-size is a global parameter and is fixed and enforced in the SAN by the SAN administrators. Clearly, the choice of EDU-size affects the encryption granularity. In other words, the number of data encryption keys is determined by the number of EDU-s. Choosing the optimum size is an important issue which affects the overall performance of the system. However, we will not

[5]This will require storing additional metadata corresponding to each EDU on the disk.

discuss the factors that may affect this decision in this paper.

In essence, encrypting data in each EDU with a separate key results in several data encryption keys for a given virtual disk. These keys need to be shared among the group of SAN entities authorized to virtualize that disk. Additionally, any authorized SAN entity should be allowed to unilaterally change a EDU-specific key. To avoid explicit communication of key updates, all EDU keys for a particular virtual disk should be stored on that disk. Of course, these keys are themselves encrypted to enable their seamless retrieval by the authorized SAN entities. One way to do so is by encrypting each EDU-specific key with the public key of each authorized SAN entity (via public key encryption)[6]. Assuming n EDUs and k authorized SAN entities, a total $(n * k)$ EDU keys would be stored on a given volume.

A more elegant approach is to use a single *Master Key* to encrypt individual EDU keys. All SAN entities authorized to virtualize a volume can be viewed as a peer group and the Master Key that is used to encrypt individual EDU keys can be shared by the members of this peer group. The EDU keys are stored in a key *lockbox* (similar to the group lock box concept suggested in Cepheus [3]) secured by the *Master Key* and the master key itself is securely shared among all group members. (However, unlike in Cepheus, in our approach, each of the authorized SAN entities take part in the generation of the Master Key. This will be explained in the next section) This hierarchical key structure is depicted in Figure 6.

The key sharing problem is now essentially reduced to sharing the *Master Key* among the group members. Before going into the details of our proposal, we describe the fundamental components (building blocks) of a virtual volume taking into account the security-related information (EDU keys, lockboxes and Master Keys) that needs to be stored along with the encrypted data.

4.1 System components

A virtual disk contains three basic objects: *Encrypted Data Units (EDU-s)*, *Key Lockbox* and *Master Key Component*.

4.1.1 EDU-s

An EDU is a container for encrypted data segment. A virtual disk includes one or more EDUs. One possible EDU realization is shown in Figure 7.

[6]S/MIME (Secure Multipurpose Internet Mail Extensions) [28] and Privacy-Enhanced Mail (PEM) [29] use this technique to send encrypted e-mail to multiple recipients: A session key is generated, the message is encrypted using this session key using a symmetric encryption algorithm, and the session key is then encrypted with each recipient's public key (via public key encryption).

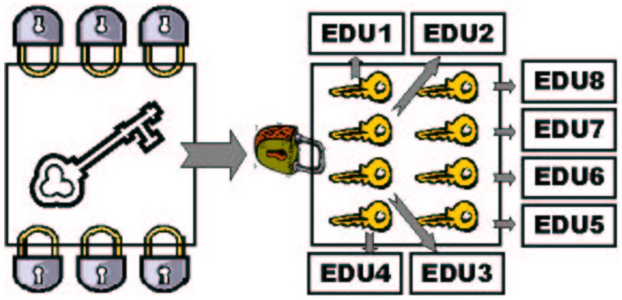
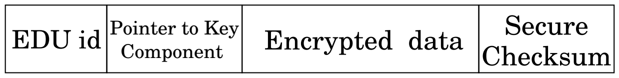


Figure 6. Hierarchical Structure



EDU id	Pointer to Key Component	Encrypted data	Secure Checksum

Figure 7. Encrypted Data Unit

The EDU-id uniquely identifies an EDU on a virtual disk. Each EDU contains multiple 512-byte blocks of data encrypted under a symmetric encryption algorithm with a single key. The EDU encryption key is stored in a *Key Lockbox* the location of which is stored in the key pointer field. Finally, every EDU contains a secure checksum (i.e., a keyed hash such as HMAC [30]) of the cleartext data[7].

4.1.2 Key Lockbox

A *Key Lockbox* (KLB) stores EDU keys (see Figure 8 for one potential representation) encrypted under a *Master Key*. The pointer field of the KLB object points to the *Master Key* component, which, in turn, stores the master key itself. Each triple: SKD_i: {*EDU id*, *Encrypted key*, *Validity*} determines the encrypted EDU key. Also, *Validity* contains a keyed hash computed as:

$$HMAC(\text{clear\_text}||\text{dirty\_bit})$$

where *dirty_bit* denotes the "compromise" status of the data in this EDU. When this bit is set, the data has been read or written by a potentially compromised SAN entity. The Validity field is useful when we employ so-called *Lazy Re-encryption* discussed in section 5 below. Finally, the *SAN entity id* field identifies the SAN entity that last modified this KLB and the corresponding signature is basically the SAN entity's digital signature over the entire KLB. The signature protects the integrity of

[7] Note that an encryption key and an HMAC key are different. In general, we can derive two keys from the same secret key K by, say, $k_1 = h(K)$ and $k_2 = f(K)$ where $h()$ and $f()$ are distinct one-way functions.

the KLB itself and provides origin authentication of the SAN entity last to modify the KLB.

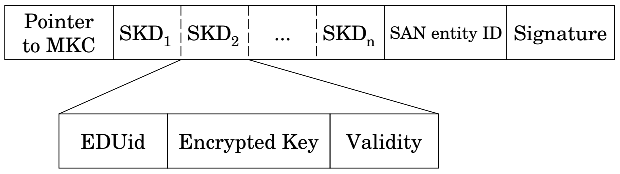


Figure 8. Key Lockbox

4.1.3 Master Key Component

The *Master Key Component* (MKC) contains the information necessary to retrieve the Master Key. Two different types of MKC-s are used depending upon whether *Encrypted Master Key* or *Shared Group Key* approach is used. The structural representation of the two types are as follows:

Encrypted Master Key (Public Key) approach is shown in Figure 9. We refer to the corresponding MKC as *MKC_PK*. In this case, each tuple: $MKD_i = \{\text{SAN Entity id}, \text{Encrypted Key}\}$ stores the Master Key encrypted for each authorized SAN entity. Also included is the identity and the signature of the SAN entity that last modified the MKC_PK object.

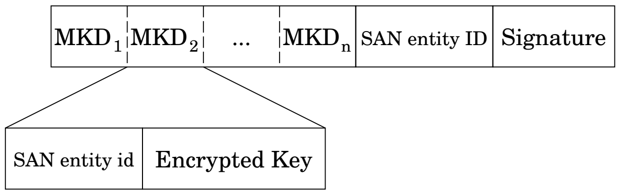


Figure 9. Master Key Component (Encrypted Master Key approach)

Shared Group Key approach is shown in Figure 10. The MKC is referred to as *MKC_GK*. This object stores public key-related information (blinded key-tree in TGDH) derived from the contributions of all members. Any *current* member can compute the group key (which is also the Master key) by combining this public information with its own secret share (as described in section 3.3). Once again, the signature of the SAN entity that last modified this object is included.

5 Architectural Details

In this section, we discuss the *secure* SAN architecture. More concretely, we discuss the details of two Master-key management schemes: one using Encrypted Master Key approach and the other using group-key agreement method.

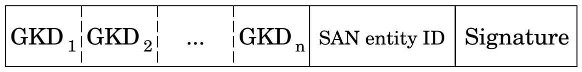


Figure 10. Master Key Component (GK approach)

5.1 Encrypted Master Key Approach

We saw in section 4 that introducing a higher-level *Master Key* helps key management. In the Encrypted Master Key approach, this Master Key for a virtual disk is simply chosen by one of the SAN entities authorized to virtualize that volume. All EDU keys are then encrypted with the Master Key and stored in a KLB. The Master Key is, in turn, encrypted individually for all other authorized SAN entities and stored in a MKC_PK. Thus the total number of encrypted keys to be stored on the virtual disk is $(n + k)$, i.e., n EDU keys encrypted with the Master Key and the Master Key encrypted k times (there are k authorized SAN entities). A change of an EDU key, requires one encryption of the new key (under the Master Key) and an update of the Key Lockbox (including a signature). Only when the Master Key is changed, a SAN entity must modify the MKC_PK component by encrypting the new key k times. Figure 11 summarizes the basics of this approach and the scheme is briefly explained below:

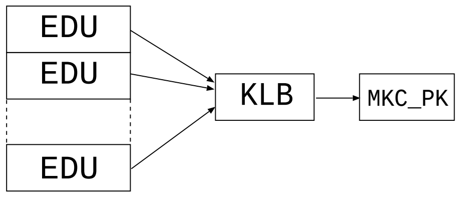


Figure 11. Encrypted Master Key Approach

Virtual Disk Initialization Event Assume virtual disk V_i is created by the storage administrator with the help of SAN entity S_j. The master key MK for V_i is chosen (randomly) by S_j. MK is encrypted with PK_j (S_j's public key) and stored in MKC_PK of V_i, along with S_j's signature.

SAN Entity Join Event Assume that the group of SAN entities authorized to access virtual disk V_i, currently has j members: $\{S_1, \ldots, S_j\}$. Assume S_{j+1} wants to instantiate V_i. We make an assumption that when a SAN entity is attempting to join the group, at least one of the current members is available to provide assistance. The current member encrypts the Master Key of V_i for S_{j+1} (using PK_{j+1}) and includes it in the MKC_PK component. The current member also computes a new signature on MKC_PK to reflect this change.

Key Update Event As mentioned in Section 2.1, a key update event may be triggered because of a compromised key, a compromised SAN entity or periodically when the key needs to be refreshed. These situations are dealt with differently as explained below:

- EDU Key refresh event: Assume SAN entity S_j wants to refresh the data encryption key of EDU_a. S_j chooses a new key EK_{new} and re-encrypts the data in EDU_a with EK_{new}. It also encrypts EK_{new} with the master-key MK and stores the encrypted key in KLB. Additionally, it sets the *validity* field corresponding to that EDU in KLB to indicate that the EDU key and data are *fresh* (i.e., not compromised). Finally, S_j adds its signature to KLB and writes re-encrypted EDU (along with its new secure checksum) back to disk.
- Master Key refresh event: Assume SAN entity S_j wants to refresh the master key of disk V_i[8]. S_j chooses a new master key MK_{new}, re-encrypts all the EDU keys in KLB with MK_{new}

  $$\forall EK \in KLB, E_{MK_{new}}(EK) \hookrightarrow KLB$$

  S_j appends its signature to KLB. Next, S_j encrypts MK_{new} for all authorized SAN entities and updates MKC_PK of V_i. S_j also generates a signature on MKC_PK.

  $$\forall S_i \in \{\text{Authorized SAN Entities for } V_i\},\\ \{E_{PK_i}(MK_{new})\} \hookrightarrow MKC\_PK$$

- Key update event due to key compromise: In the event of a EDU key compromise, the key needs to be changed and the affected data needs to be re-encrypted. This event handling is similar to a EDU key-refresh event as described above. Note that we do not consider the possibility of an attacker breaking only the Master Key without first compromising a SAN entity. Since the master key is used only to encrypt the EDU encryption keys, we claim that the amount of information encrypted using the Master key is insufficient to enable successful cryptanalysis. Hence, we do not consider Master key compromise event separately.
- Key update event due to SAN entity compromise: Compromised SAN entity implies compromised Master Key. (In other words, this implies compromise of all data stored on that virtual disk.) Handling this event requires the Master key and all individual EDU keys to be changed and the entire disk data to be re-encrypted. Since re-encrypting the entire disk data can be very expensive, we employ *lazy re-encryption* mechanism proposed in Cepheus[3]: the SAN entity handling this event simply changes the Master Key (the master key is changed by triggering a master-key refresh event) and additionally sets the *validity* field value for every EDU in KLB as compromised. The EDU key is changed and the data is re-encrypted subsequently during a disk-access or a key-refresh event on that EDU.

[8]Master key, like any other key, is subject to aging and prudent security practices require that it should be periodically refreshed

Disk Access Event Assume S_j wants to access EDU_a of virtual disk V_i. S_j obtains the encrypted MK from MKC_PK of V_i and decrypts it using its private key SK_j. Using MK, key lockbox KLB is unlocked to obtain the encryption key EK for EDU_a. EK is used for accessing the data. If that EK does not exist (i.e., EDU_a is empty and S_j wants to write to it), then S_j chooses a new EK and updates the KLB to include this new entry. Additionally, S_j should change this EK, if the *validity* field corresponding to that EDU is set as compromised. S_j can change the EDU key by triggering a EDU key refresh event.

5.2 Shared Group Key approach

The other approach that we propose for sharing a single key among all group members is based on group key agreement (Section 3.3). In this approach, instead of relying on any one SAN entity to choose the Master key for a virtual volume, the SAN entities share a *Group Key* that has contributions from all the members. All *blinded* (public) keys are stored in the MKC_GK component of that disk (Figure 12). Any authorized member can compute the group key by using the blinded keys on the key-tree and its own secret share. The scheme is briefly explained below:

Virtual Disk Initialization Event Assume virtual disk V_i is created by the storage administrator with the help of SAN entity S_j. S_j randomly chooses its private

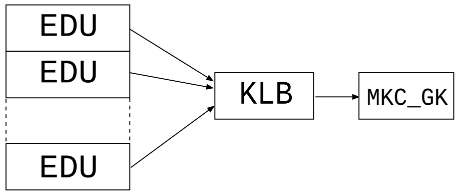


Figure 12. Shared Group Key Approach

key share N_j which is also the initial *Group Key* (GK) since S_j is the sole member of the group. S_j computes the corresponding public key (*blinded value*) and stores this public information in MKC_GK of V_i. S_j's signature is also stored in MKC_GK.

SAN Entity Join Event Assume V_i is currently instantiated by j members: $\{S_1, \ldots, S_j\}$. Assume S_{j+1} also wants to instantiate V_i. The group member that receives the `join_request` from S_{j+1}, updates the key-tree by including S_{j+1}'s blinded share in the group key-tree, thereby modifying the MKC_GK component (Please refer to Section 3.4 for details of the protocol). This results in a new group key for V_i and KLB is updated by encrypting all EDU keys with the new group key. The group member helping S_{j+1} recomputes the signatures on KLB and MKC_GK.

Key Update Event

- EDU Key refresh event: This event is similar to EDU key refresh event in Encrypted Master Key approach. Assume S_j wants to refresh the data encryption key of EDU_a. S_j chooses a new key EK_{new} and re-encrypts the data in EDU_a with EK_{new}. It also encrypts EK_{new} with the group key GK and updates KLB accordingly. Additionally, it sets the *validity* field corresponding to that EDU in KLB to indicate that the EDU key is valid. Finally, S_j adds its signature to KLB and writes re-encrypted EDU back to disk.

- Group Key refresh event: Assume S_j wants to refresh the group key of disk V_i. S_j picks a new share for itself - $N_{j_{new}}$ and updates the blinded key-tree to reflect its new share, thereby changing the group key (Please refer to Section 3.4 for details of the protocol for updating the key-tree). S_j also re-encrypts all the EDU keys in KLB with GK_{new}. Finally, S_j generates new signatures for KLB and MKC_GK.

$$\forall EK \in KLB, E_{GK_{new}}(EK) \hookrightarrow KLB$$

- Key update due to key compromise: In the event of a EDU key compromise, the key needs to be changed and the affected data needs to be re-encrypted by the SAN entity handling that event. This event handling is similar to a EDU key-refresh event as described above. Once again, we do not consider the possibility of an attacker compromising only the group key without compromising a SAN entity.

- Key update due to SAN entity compromise: When a member SAN entity is evicted, effectively, the group key is compromised. This implies that the data on the entire virtual disk is compromised. Handling this event requires the group key and all individual EDU keys to be changed and the entire disk data to be re-encrypted. Since, this can be a very expensive operation, we employ *lazy re-encryption* mechanism: the SAN entity handling the event changes its private key share, deletes the leaf node corresponding to the evicted member from the group tree and recomputes the new group key (Please refer to Section 3.4 for details of the *evict* protocol). The rest is the same as handling lazy re-encryption in Encrypted Master Key approach: SAN entity handling the event updates the KLB by encrypting all EDU keys with the new group key and sets the *validity* field value for every EDU in KLB as compromised. The EDU key is changed and the data is re-encrypted subsequently during a disk-access or a key-refresh event on that EDU.

Disk Access Event Assume S_j wants to access EDU_a of V_i. S_j obtains the blinded key-tree information corresponding to V_i's GK from MKC_GK and computes the value of GK using this public information and its own private secret share N_j. This GK is used to open the key lockbox KLB to obtain the encryption key EK for EDU_a. Again, if that EK does not exist (i.e., EDU_a is empty and S_j wants to write to it), then S_j should choose a new EK and update the KLB to include this new entry. Additionally, S_j should change EK during a disk access event if the *validity* field corresponding to that EDU is set as compromised. S_j can change the EDU key by triggering a EDU key refresh event.

6 Discussion

In this section, we compare the security and efficiency of these two key-management approaches.

6.1 Security

One of the major differences in the two approaches concerns the Master Key: In the Encrypted Master Key approach, the master key is chosen by a single SAN entity, whereas, in the Shared Group Key approach, the group key is determined by contributions from all SAN entities that have currently instantiated that virtual volume. Hence, in the Shared Group Key approach, the randomness of the master key does not depend on the ability of a "single" SAN entity to choose cryptographically strong random keys. Additionally, in the Encrypted Master Key approach, the master key is encrypted using each SAN entity's public key whereas, in the Shared Group Key approach, no such long term keys are used (Since the group key and the individual shares are periodically refreshed).

Another important distinction between the two approaches pertains to the SAN entity join event. The master key in Encrypted Master Key approach remains the same as new entities instantiate the volume unless a master key update event is triggered specifically to change it. In contrast, the master key in Shared Group Key approach is changed automatically while handling the join event. (See Step 2 in Figure 3). As a result, the Shared Group Key approach is able to provide *forward and backward secrecy* (and therefore, *key independence*) in the current setting[9]. In [27], TGDH is proven to provide key independence. In order to get the same level of security in the Encrypted Master Key approach, an explicit master key update event needs to be triggered every time the group membership changes as a result of a SAN entity join event.

6.2 Efficiency

Table 1 compares the two approaches discussed in the previous section with respect to the number of modular exponentiations required to handle the basic system events. The cost of each modular exponentiation differs depending on the public key encryption algorithm. Therefore, in case of Encrypted Master Key approach (called PK in the table), we counted the number of public key operations (encryption or decryption). The cost of Shared Group Key approach (called GKA) is more straight-forward: only pure modular exponentiations are considered. These operations are performed by the SAN entity handling an event. The count does not include signature generation (since same number of digital signatures are used in both the approaches). In table 1, k denotes the total number of SAN entities that have instantiated a given virtual volume. We assume that the average height of the TGDH key-tree is $\log k$.

Details of the cost are as follows:

- Initialization event: For PK, the master key needs to be encrypted with the public key of the SAN entity that was involved in the initialization event. For GK, on the other hand, the group key needs to be blinded. Therefore, both approaches require 1 public key operation each.

- Key update event: To refresh the master key for PK, the new master key needs to be encrypted k times (for each member SAN entity), while GKA requires $2 \log k$ public key operations. SAN Entity compromise (or eviction) cost is almost the same as master key refresh operation, since it essentially involves refreshing the master key for $k-1$ remaining members. Note that we do not need any public key operations for EDU key refresh.

- Disk access event: When accessing the virtual disk, PK requires just one public key operation to compute the master key, while GKA needs $\log k$ operations. To improve efficiency, we can allow the SAN entities to cache the master key locally. Note that when cached master key is valid, no public key operations are required for disk access events.

- SAN Entity join event: In case of GKA, adding a new member takes $2 \log k$ public key operations. As we mentioned in section 6.1, this automatically provides key independence. In order to get key independence in PK, we need to first trigger a Master key update event (which costs k encryptions) and then encrypt the new master key for the new member. Hence, the total cost is $k + 1$ public key operations.

- Memory requirement: total memory size required to store the master key (group key) on the virtual disk, i.e., it measures the size of MKC_PK and MKC_GK respectively. For PK, it is $k * (keysize)$. $keysize$ depends on the public key encryption algorithm. If we use RSA, $keysize$ is at least 1024 bits. For GKA, total memory required is $(2k-1) * (keysize)$ (The group key-tree is a binary tree with k leaves). The $keysize$ for GKA is 1024 bits. Note that we do not consider the memory requirements for KLB here, since the size of KLB is identical in both the approaches.

[9]Forward secrecy (not to be confused with Perfect Forward Secrecy or PFS) guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys. In contrast, backward secrecy guarantees that a passive adversary who knows a contiguous subset of group keys cannot discover preceding group keys. Key independence guarantees that a passive adversary who knows a proper subset of group keys $\hat{K} \subset \mathcal{K}$ cannot discover any other group key $\bar{K} \in (\mathcal{K} - \hat{K})$. [17]

Table 1. Cost comparisons

			PK	GKA
Modular exponentiations	Initialization Event		1	1
	Key update event	Periodic refresh	k	$2 \log k$
		EDU Key Compromise	0	0
		SAN Entity Compromise	$k - 1$	$2 \log k$
	Disk access event	Data read (normal)	1	$\log k$
		Data read (cached key)	0	0
		Data write (normal)	1	$\log k$
		Data write (cached key)	0	0
	SAN Entity join Event		$1 + k$	$2 \log k$
Memory requirement			$k * (keysize)$	$(2k - 1) * (keysize)$

7 Conclusions

In this paper, we proposed a security architecture for preserving privacy and integrity of SAN data with an additional emphasis on secure and efficient key management. In our approach, SAN entities are responsible for active enforcement of security. Our schemes utilize the nascent computing power of the SAN entities to carry out computationally intensive cryptographic functions. We specifically addressed the key management problem. Exploiting the peer group nature of the SAN entities virtualizing a secure disk, we presented two mechanisms to enable key sharing that do not require any centralized servers.

8 Acknowledgements

The authors wish to thank the anonymous reviewers for their insightful comments.

References

[1] M. Blaze, “A cryptographic file system for unix,” in *1st ACM Conference on Computer and Communications Security*, pp. 9–15, ACM Press, November 1993.

[2] K. Fu, M. F. Kaashoek, and D. Mazieres, “Fast and secure distributed read-only file system,” *Computer Systems*, vol. 20, no. 1, pp. 1–24, 2002.

[3] K. Fu, “Group sharing and random access in cryptographic storage file systems,” Master’s thesis, Massachusetts Institute of Technology, 1999.

[4] H. Gobioff, *Security for a High Performance Commodity Storage Subsystem*. PhD thesis, Carnegie Mellon University, 1999.

[5] E. Reidel, M. Kallahalla, and R. Swaminathan, “A framework for evaluating storage system security,” in *Conference on File and Storage Technologies (FAST)*, Jan. 2002.

[6] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel, “Separating key management from file system security,” in *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP ‘99)*, pp. 124–139, Dec. 1999.

[7] E. Miller, D. Long, W. Freeman, and B. Reed, “Strong Security for Network-Attached Storage,” in *Conference on File and Storage Technologies (FAST)*, pp. 1–13, Jan. 2002.

[8] T. Wu, “The secure remote password protocol,” in *Symposium on Network and Distributed Systems Security (NDSS ’98)*, (San Diego, California), pp. 97–111, Internet Society, Mar. 1998.

[9] J. Case, R. Mundy, D. Partain, and B. Stewart, “Introduction to Version 3 of the Internet-standard Network Management Framework,” Internet Request for Comment RFC 2570, Internet Engineering Task Force, Apr. 1999.

[10] M. Krueger, R. Haagens, C. Sapuntzakis, and M. Bakke, “Small Computer Systems Interface protocol over the Internet(iSCSI) Requirements and Design Considerations,” Internet Request for Comment RFC 3347, Internet Engineering Task Force, July 2002.

[11] M. Fischlin, “Incremental cryptography and memory checkers,” in *Advances in Cryptology – EUROCRYPT ’97* (W. Fumy, ed.), no. 1233 in Lecture Notes in Computer Science, pp. 393–408, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1997.

[12] National Institute of Standards and Technology (NIST), Computer Systems Laboratory, "Advanced Encryption Standard." Federal Information Processing Standards Publication (FIPS PUB) 197, Nov. 2001.

[13] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, pp. 769–780, Aug. 2000.

[14] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications, CRC Press, 1997. ISBN 0-8493-8523-7.

[15] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

[16] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31, pp. 469–472, July 1985.

[17] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *7th ACM Conference on Computer and Communications Security* (S. Jajodia, ed.), (Athens, Greece), pp. 235–244, ACM Press, Nov. 2000.

[18] C. K. Wong, M. G. Gouda, and S. S. Lam, "Secure group communications using key graphs," in *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 68–79, 1998. Appeared in ACM SIGCOMM Computer Communication Review, Vol. 28, No. 4 (Oct. 1998).

[19] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system," in *Advances in Cryptology – CRYPTO '88* (S. Goldwasser, ed.), no. 403 in Lecture Notes in Computer Science, (Santa Barbara, CA, USA), pp. 520–528, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1990.

[20] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to groups," in *3rd ACM Conference on Computer and Communications Security*, pp. 31–37, ACM Press, March 1996.

[21] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Advances in Cryptology – EUROCRYPT '94* (A. D. Santis, ed.), no. 950 in Lecture Notes in Computer Science, pp. 275–286, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1995. Final version of proceedings.

[22] Y. Kim, A. Perrig, and G. Tsudik, "Communication-efficient group key agreement," in *Information Systems Security, Proceedings of the 17th International Information Security Conference IFIP SEC'01*, 2001.

[23] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, "On the performance of group key agreement protocols," in *IEEE International Conference on Distributed Computing Systems*, July 2002.

[24] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, Nov. 1976.

[25] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: Issues and architectures," Internet Request for Comment RFC 2627, Internet Engineering Task Force, June 1999.

[26] D. Boneh, "The Decision Diffie-Hellman problem," in *Third Algorithmic Number Theory Symposium*, no. 1423 in Lecture Notes in Computer Science, pp. 48–63, Springer-Verlag, Berlin Germany, 1998.

[27] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," available from `http://eprint.iacr.org`, Record 2002/009, Cryptology ePrint Archive, Feb. 2002.

[28] B. Ramsdell, "S/MIME Version 3 Message Specification," Internet Request for Comment RFC 2633, Internet Engineering Task Force, June 1999.

[29] J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures," Internet Request for Comment RFC 1421, Internet Engineering Task Force, Feb. 1993.

[30] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology – CRYPTO '96* (N. Koblitz, ed.), no. 1109 in Lecture Notes in Computer Science, pp. 1–15, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1996.