A Multi-User Steganographic File System on Untrusted Shared Storage

Jin Han Meng Pan Debin Gao HweeHwa Pang
Singapore Management University
{jin.han.2007, mengpan, dbgao, hhpang}@smu.edu.sg

ABSTRACT

Existing steganographic file systems enable a user to hide the existence of his secret data by claiming that they are (static) dummy data created during disk initialization. Such a claim is plausible if the adversary only sees the disk content at the point of attack. In a multi-user computing environment that employs untrusted shared storage, however, the adversary could have taken multiple snapshots of the disk content over time. Since the dummy data are static, the differences across snapshots thus disclose the locations of user data, and could even reveal the user passwords.

In this paper, we introduce a Dummy-Relocatable Steganographic (DRSteg) file system to provide deniability in multi-user environments where the adversary may have multiple snapshots of the disk content. With its novel techniques for sharing and relocating dummy data during runtime, DRSteg allows a data owner to surrender only some data and attribute the unexplained changes across snapshots to the dummy operations. The level of deniability offered by DRSteg is configurable by the users, to balance against the resulting performance overhead. Additionally, DRSteg guarantees the integrity of the protected data, except where users voluntarily overwrite data under duress.

1. INTRODUCTION

Steganographic File Systems (stegfs) are intended to provide plausible deniability to data owners in the event that they are forced to disclose their secret data [4]. A stegfs hides encrypted user data among dummy data that contain only pseudo-random bits. Without the correct password, it is not possible to differentiate user data from dummy (based on the assumption that the output of the block cipher is indistinguishable from random bits [3, 4]), even for an adversary who understands the mechanisms of the file system and is able to gain access to the storage devices. This feature allows a data owner to selectively reveal some directories/files, but disclaim the existence of his sensitive data.

To be believable, the disclaimer of the data owner must be consistent with the information that the adversary is able to gather about the file system. This is much more challenging to achieve in modern computing environments when the user data are encrypted and stored in shared network storage. Compared to portable and local storage, network storage dramatically increases the availability and accessibility of user data. However, it also brings new challenges in securing user data. With shared network storage, the adversary is no longer limited to a single snapshot of the disk content at the point of attack. Instead, the adversary could now locate the physical server machines being used [17] and quietly amass multiple snapshots of the file system over a period of time before launching his attack. The additional knowledge that the adversary gleams from the multiple snapshots must be factored into the stegfs design.

In earlier stegfs designs [4, 15, 12, 16], dummy data are created when the disk is formatted and remain static thereafter. These schemes are effective against adversaries who only see the final state of the storage, but cannot defend against adversaries who possess multiple snapshots of the storage. Indeed, changes among different snapshots not only reveal the location of secret data, but could even be utilized to recover the access keys (for example, when the first scheme by Anderson et al. [4] is utilized). Recent stegfs schemes, which are proposed to defend against multiple-snapshots attacks, either cannot guarantee the integrity of user data even under legitimate data operations [8, 9], or require a trusted agent to manage all the user passwords and dummy data [20], which effectively presents a single point of disclosure for user passwords.

In this paper, we propose a multi-user stegfs for shared storage systems, which is named as DRSteg – Dummy Relocatable Steganographic file system. DRSteg is designed to meet the following requirements:

- Security: To provide plausible deniability of secret data in a multi-user environment in which the adversary could obtain multiple snapshots of the storage content. This protection should extend to any user even when the storage server and all the other users are completely compromised, i.e., they have surrendered all the information in their possession.
- Usability: To guarantee data integrity, and at the same time enable individual users to trade off between deniability and system performance.

To the best of our knowledge, DRSteg is the first stegfs that allows I/O operations observed on shared storage to be plausibly attributed to dummy data without requiring a trusted agent as used by Zhou et al. [20]. In addition, our work also manages to increase the deniability provided to individual users by sharing dummies among multiple users in the system. It is technically challenging to satisfy both the security and usability requirements, especially when dummies are shared. DRSteg incorporates a special dummy relocation mechanism that enables individual users to distinguish dummies from other users' data (in order to free dummies without destroying data), and to prevent adversaries from discerning the difference between dummy and user data even after obtaining multiple snapshots.

This is also the first work that formalizes the deniability achieved by a multi-user stegfs. The formalization enables us to develop a tunable mechanism for users to balance between deniability and system responsiveness. In DRSteg, the deniability enjoyed by individual users could be maintained beyond a specified threshold, whether or not all the other users are fully compromised. The amount of dummy operations is controlled individually; a user who specifies a more aggressive amount enjoys higher deniability at the expense of slower file operations.

To substantiate the usability of DRSteg, we present results of an empirical evaluation using file operation logs collected from 12 graduate students in our school. The results confirm that DRSteg is capable of achieving a wide range of user-specified deniability levels. We also implemented a prototype of DRSteg as a file system module in Linux kernel. Performance experiments on the prototype show that security and performance can be traded off against each other.

2. RELATED WORK

Cryptographic file systems (e.g., [5, 7, 11, 19]) and their implementations (e.g., [1, 2]) have been studied extensively in the last two decades . A cryptographic file system complements the access control mechanism of the operating system (OS). Even if the OS is compromised or the data storage is removed from the OS, data in the file system remain protected by the user's password. A weakness of cryptographic file systems is that they leave evidence of the existence of encrypted data, so a determined attacker may compel the users to reveal their decryption passwords.

In order to provide plausible deniability of the existence of secret data, Anderson et al. proposed two steganographic file system (stegfs) schemes [4]. In the first scheme, the disk is initialized with several cover files that have equal length and contain random data. A secret object is stored through an exclusive-or operation on a subset of the cover files, identified by the corresponding bits in the access key. To protect against brute force attacks, the number of cover files must be sufficiently large; this imposes heavy I/O overheads as each read/write request for an object translates into operations on multiple cover files. The scheme is effective against single-snapshot attacks but not multiple-snapshot attacks. In particular, the differences between just two snapshots of the storage can expose the access key used[1].

In Anderson's second scheme [4], the disk is first filled with random bits. Subsequently, secret data blocks are written to pseudorandom addresses. An implementation of this scheme on Linux is reported by McDonald et al. [15], a peer-to-peer version by Hand et al. [12] and a distributed version by Giefer et al. [10]. The disadvantage of the scheme is that the probability of collision in the locations where data are stored increases as more data are added to the disk. Although replicating each data block in different locations reduces the likelihood of data loss, the risk cannot be eliminated; hence data integrity is not guaranteed.

Pang et al. [16] utilized a bitmap to track block allocation to avoid overwriting data and to improve system performance. To defend against single-snapshot attacks, dummy data are added when the disk is initialized. The dummy data cannot be changed or relocated at runtime, so the scheme is susceptible to multiple-snapshot attacks. Zhou et al. [20] provided for the relocation of dummy blocks. Their solution requires a trusted agent to manage all the user passwords and dummy data, which effectively transfers the risk of password disclosure to the agent.

Diaz et al. [8] proposed to defend against traffic analysis [18] through a mix-based stegfs that employs a local mix to relocate files in the remote storage. They show that the security of the scheme depends on the file-size patterns in the system. Another work by Domingo-Ferrer et al. [9] addressed the problem of data loss in a stegfs with multiple users. It is not designed to defend against multiple-snapshot attacks though. Furthermore, neither of the two schemes guarantees data integrity under legitimate data operations.

TrueCrypt[2], an open-source disk-encryption software package, enables a user to create a deniable file system within a regular encrypted file or partition. The file system is deniable if the adversary only sees the final content of the disk. However, it cannot defend against an adversary who possesses multiple snapshots of the encrypted partition. The same weakness exists in similar products that provide deniability for secret files, e.g., Phonebook[3] and Rubberhose[4].

Note that deniability in stegfs is different from deniable encryption [6] which allows an encrypted message to be decrypted into different sensible plaintexts with different keys. Stegfs is also different from private information retrieval (PIR) [13] which allows a user to retrieve an item from a server without revealing which item is retrieved. A stegfs allows the untrusted server to be cognizant of which disk blocks are retrieved, yet provides deniability that they stemmed from operations on secret data. A stegfs is not designed to prove non-existence of secret data but to provide plausible deniability of the existence of secret data.

3. PROBLEM DEFINITION

3.1 Threat Model

Figure 1 depicts our model of a multi-user file system. In the model, user data are stored on a shared storage. The stegfs functionalities are implemented in the client module that runs on the user computers. This client module is secured so that sensitive data that are operated on as well as any passwords used for encrypting and decrypting the data are protected. The storage server manages the shared stor-

[1]This is because only cover files whose indexes correspond to bits with value "1" in the access key will be modified for any data modification. Those files which do not change will correspond to bits with value "0" in the access key. Thus, the access key can be reconstructed by observing the changes of the cover-file matrix in the storage.

[2]TrueCrypt, `http://www.truecrypt.org/`

[3]Phonebook, `http://www.freenet.org.nz/phonebook`

[4]Rubberhose, `http://iq.org/~proff/rubberhose.org`

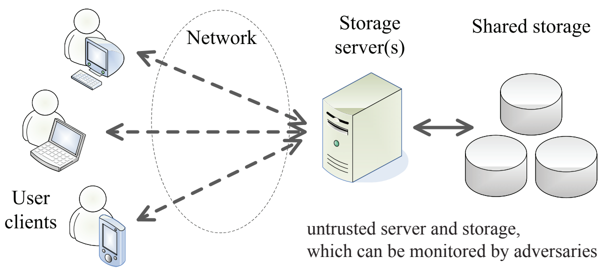

Network
Storage server(s)
Shared storage
User clients
untrusted server and storage, which can be monitored by adversaries

Figure 1: A multi-user stegfs with untrusted shared storage

age devices which provide block-level operations, including DAS (direct attached storage) and SAN (storage area network). Different from the model where the server manages all the user passwords [20], the storage server and shared storage in our model are not stegfs specific.

The server and the storage devices are not trusted. This means that an adversary may infiltrate the server or the storage devices directly (or the backup of these devices) to copy and analyze the stored content. Although our scheme provides better protection when the communication between users and the server is anonymized, it is not a necessary condition for DRSteg to provide deniability to users. We will analyze the deniability of DRSteg under different scenarios in Section 5.

In this paper, we focus on adversaries who are after the user data, and we explicitly rule out considerations of sabotage like overwriting/deleting data and denial of service. The threat posed by the adversary thus hinges on two factors: (a) his knowledge of the file system state, and (b) his access to the users of the system. These two factors together determine the adversary's ability to make deductions about the hidden data on the storage, and to verify any claims elicited from the users.

The first factor, knowledge of the storage state, is characterized by the number of observations of the storage content. An adversary who is able to access the storage only once (i.e., at the point of attack) only gains a *single snapshot* of the storage. An example is someone who is captured by criminals and forced to reveal all the contents in his portable drive. However, when the adversary has more than one chance to access the storage, he can record *multiple snapshots*. The information in those snapshots is then utilized to deduce the existence of secret data.

The second factor that defines the adversary's ability concerns his access to the users. Here, we make the following assumption:

Victim isolation assumption. In coercing information from the users, it would be effective for the adversary to interrogate them separately and cross-check the information elicited. Placed in isolation, a victim knows neither which other users have been compromised nor what information they have surrendered. Consequently, each victim has to assume the worst, i.e., that all the other users are compromised and all their secrets are revealed. He thus has to independently decide what data he can hide without being contradicted by other users' disclosure.

Multi-user encrypting file systems [2, 5, 7] are inadequate under the victim isolation assumption, as it is not safe for a user to claim his data to belong to someone else. A solution is to use dummy blocks, which should be operated on in similar ways as encrypted data blocks in order to defend against multiple snapshot attacks.

3.2 Definition of Deniability

To formalize the threat, an adversary has access to a sequence of snapshots $\mathsf{S} = \{\mathsf{s}_1, \mathsf{s}_2, \ldots, \mathsf{s}_\mathsf{T}\}$ of the stegfs partition on the disk, where s_T is the snapshot at the time of coercion. Following the victim isolation assumption, the adversary extracts all the passwords from other users (P') at the time of attack, and also coerces the victim to reveal his passwords $\mathsf{P}_\mathsf{t} = \{\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_\mathsf{t}\}$. The adversary then utilizes the passwords obtained to decode the information in each snapshot.

Let $\mathsf{H}_\mathsf{i}^{\mathsf{dummy}}$ and $\mathsf{H}_\mathsf{i}^{\mathsf{data}}$ denote the hypotheses that an allocated block blk_i is a dummy block and a data block, respectively. Let e_i denote the evidence on blk_i observed from S, and $\mathsf{E} = \{\mathsf{e}_\mathsf{i}\}$ the aggregate evidence across all the disk blocks. We define the *plausible deniability* of blk_i as follows.

Definition 1. *Given the evidence* $\mathsf{E} = \{\mathsf{e}_\mathsf{i}\} = \mathsf{S} \cup \mathsf{P}' \cup \mathsf{P}_\mathsf{t}$, *where* $\mathsf{S} = \{\mathsf{s}_1, \mathsf{s}_2, \ldots, \mathsf{s}_\mathsf{T}\}$ *is a sequence of snapshots taken by the adversary and* $\mathsf{P}' \cup \mathsf{P}_\mathsf{t}$ *is the set of passwords revealed to the adversary (along with the blocks decrypted with these passwords), the deniability of an allocated block* blk_i *is the posterior probability that* e_i *was generated by operations on dummy block* blk_i*:*

$$\mathsf{deny}_\mathsf{i} = Pr(\mathsf{H}_\mathsf{i}^{\mathsf{dummy}} | \mathsf{e}_\mathsf{i}) \tag{1}$$

A steganographic file system is said to be ***α-deniable*** *if*

$$\mathsf{deny}_\mathsf{i} \geq \alpha$$

for all blk_i *that cannot be decrypted with* $\mathsf{P}' \cup \mathsf{P}_\mathsf{t}$*, for any* $t \geq 1$ *of the user's choice.*

An α-deniable stegfs guarantees that any evidence gathered by an adversary (e.g., disk images across multiple snapshots) is caused by dummy data operations with at least a probability of α. This means that a user of the system can attribute the evidence to dummy operations without revealing his secret data.

4. DESIGN OF DRSteg

DRSteg is designed to enable a user to selectively disclose some of his data, while enjoying α-deniability for the rest of the data that he is withholding from the adversary. We begin this section with an overview of the DRSteg design, before presenting the detailed data structures and implementation considerations.

4.1 Overview of DRSteg

In DRSteg, each user must be able to protect his data with different passwords, so that he can surrender some data but not others. To achieve α-deniability for the data blocks that he is withholding, our approach is to (a) enforce a joint ownership for allocated disk blocks to prevent the adversary from associating with certainty a withheld block with any particular user, and (b) introduce dummy blocks that are operated on at runtime, so that changes to the withheld blocks can be plausibly explained by dummy operations.

We realize the joint ownership through a voting protocol. For every allocated block, m ownership shares are created and distributed to m users, including the user who requested for the block (also known as the creator). A block can subsequently be altered or freed only after all the m shares have been garnered from consenting owners. By following this policy, we ensure that the block is never deallocated without the creator's share, yet the creator of the block is obfuscated among the share owners. The creator may use an allocated block either for his data or as a dummy.

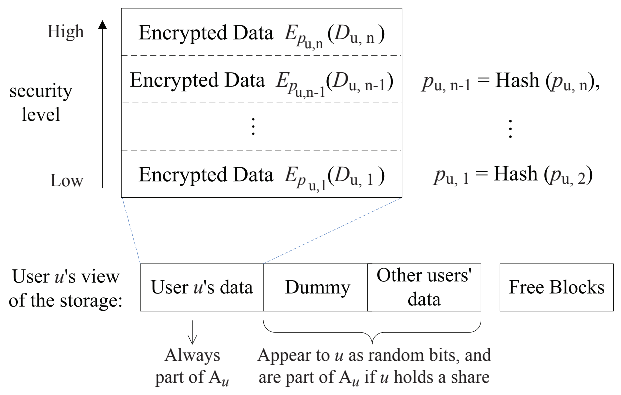
For each user, the disk blocks that hold his data are protected by one of his passwords $p_1, p_2, \dots, p_n$. The number of passwords n is expected to vary from user to user, though we use the same symbol n across users for brevity. Moreover, the passwords are generated as a hash chain [14], i.e., $p_l = h(p_{l+1})$ for a hash function h and $1 \le l < n$ (as illustrated in the upper part of Figure 2). By supplying any password p_l, $1 \le l \le n$, the user can access all the secret data at and below level l.

As for those disk blocks that are allocated as dummies, no bookkeeping information is maintained to track them directly; otherwise, the adversary can simply demand the bookkeeping information from the users, and with it discover the dummy blocks in the file system. Instead, a dummy block can only be identified through the cooperation of its owners: Each shareholder of the block checks whether it is protected with one of his passwords; if not, the block is a potential dummy – it may indeed be a dummy, or it may hold the data of some other user. It is freed in the same way as data blocks, i.e., after gathering m shares.

In the event of an attack, our DRSteg design allows a coerced user to supply some password p_t, $1 \le t < n$, to the adversary and deny the existence of the passwords p_j for $t < j \le n$. The data blocks that are protected by p_j then appear to be potential dummies, thus enabling the user to hide the existence of the data.

4.2 Detailed Design of DRSteg

Drawing on the approaches introduced above, we now put together the concrete DRSteg design. Each user u keeps track of a set of blocks A_u on which he currently holds a share. Moreover, each password p_l protects a set of data blocks $D_{u,l}$. The set difference $A_u - \cup_l D_{u,l}$ gives the blocks that exclude u's data, and dummy blocks are the allocated blocks that contain nobody's data, i.e., $\cap_u (A_u - \cup_l D_{u,l})$. Figure 2 depicts our detailed design for DRSteg (the encryption is done at the granularity of individual blocks).



High
security level
Low
Encrypted Data $E_{p_{u,n}}(D_{u,n})$
Encrypted Data $E_{p_{u,n-1}}(D_{u,n-1})$
Encrypted Data $E_{p_{u,1}}(D_{u,1})$
$p_{u,n-1}$ = Hash ($p_{u,n}$),
$p_{u,1}$ = Hash ($p_{u,2}$)
User u's view of the storage:
User u's data
Dummy
Other users' data
Free Blocks
Always part of A_u
Appear to u as random bits, and are part of A_u if u holds a share

Figure 2: Key management and user view of the storage

Whenever a user u requires a disk block blk from the file system to write data or dummy patterns, a free disk block is allocated and shares of the block are also created. One share is given to u, while the remaining shares of blk are distributed to other users u', i.e., $A_u \leftarrow A_u \cup \{\mathsf{blk}\}$ and $A_{u'} \leftarrow A_{u'} \cup \{\mathsf{blk}\}$. If user u encrypts data with his password p_l and stores it in blk, then $D_{u,l} \leftarrow D_{u,l} \cup \{\mathsf{blk}\}$.

Any user u may propose the deletion of a block in his A_u. The deletion is effected only after all the users who hold shares of the block have acquiesced. Obviously, if the block holds the data of user u', he would relocate the data before supporting the deletion. This is to avoid leaving clues for differentiating between dummy and data blocks.

With DRSteg, user u can surrender any password p_t, $1 \le t < n$ and claim that data blocks in $A_u - \cup_{1 \le j \le t} D_{u,j}$ are not his data. Claiming that data blocks in $D_{u,j}$ for $t < j \le n$ are dummy blocks is plausible since they also appear in $A_{u'} - \cup_l D_{u',l}$ of other users u' who hold shares of the blocks.

4.2.1 *Joint ownership of blocks*

We implement the joint ownership of disk blocks through a voting protocol and two data structures – a set of encrypted user share boxes (USB) and a global voting table (GVT) in clear text. A USB is used to track the A_u of each user, and a GVT records the votes surrendered by users. Two other structures are additionally maintained in clear text in the storage: a list of the users' public keys, and a bitmap to track the allocation status of the disk blocks.

When a user allocates a disk block $\mathsf{blk_i}$, he 1) sets the bit of this block to "1" in the bitmap; 2) creates m shares and writes them to the corresponding USBs; 3) writes the encrypted/random data content to the block. The format of each encrypted share is given as $E(K_{pub,u}, i)$, an encryption of i with a user's public key. The encrypted shares denote the ownership of this block. A block $\mathsf{blk} \in A_u$ if the share $E(K_{pub,u}, i)$ exists in the USB of user u. The m owners of a block include the creator and $m-1$ other users randomly selected from the public-key list.

Any of the m owners can subsequently initiate the deletion of the block blk by writing i to the global voting table (GVT) and removing his share from his USB. To support the deletion, other owners also contribute their shares into GVT. When the number of accumulated shares of a block reaches m, this block can be removed from GVT and its bit in the bitmap is set to "0" (indicating that this block is free). The share constitution ensures that the block can be deallocated only when block creator signals his agreement by surrendering his share to the GVT.

4.2.2 *Management of data blocks*

In order to provide plausible deniability against multiple-snapshot attacks, disk blocks that contain data must be managed carefully so that they leave the same evidence as operations on dummy blocks.

First, consider the modification of secret data. By comparing snapshots, the adversary may discover that the content of a block changes before all the m shares are added into GVT. This would never happen to a dummy block according to our voting protocol. Therefore, instead of overwriting data blocks, each user always migrates his updated content to new blocks, and initiates the deletion of the outdated blocks in GVT so that they will be freed in due course. However, the initiation of the deletion operation is delayed, in order to break the temporal correlation between the allocation of new blocks and the deallocation of outdated blocks.

Next, consider the case where some user's data block is registered for deallocation in GVT by other users. If the user never concurs, the adversary will suspect that the block contains data, since deallocation of dummy blocks are supported readily. To avoid suspicion, the user has to migrate the content to a fresh disk block, before relinquishing his share to the old data block.

In real implementations, the block creating operations are carried out immediately, but the voting (including removing shares from USB and writing block numbers into GVT) are delayed. We pass the voting operations to a background user process that survives beyond user log-off. The background process repeatedly initiates the deletion of a block in its pool after sleeping for a random duration. This makes the operations for data blocks plausible since the creation and voting could be caused by either creating and freeing dummy blocks or creating, modifying and freeing data blocks.

4.3 Discussions

4.3.1 *Comparing to naive designs*

There also exist alternatives in designing a multi-user steganographic file system. A naive one could simply let each user manage his own blocks (including data and dummy). Since dummy blocks are no longer shared, one has to create many more dummy blocks in order to achieve the same deniability compared to our design, when anonymous channels are used between the users and the storage server. When this channel is not anonymized, our design still provides similar security and disk utilization compared to the naive design. The deniability provided by DRSteg under both scenarios is analyzed in the next section.

4.3.2 *Encryption of the block shares*

Another security issue relates to the encryption of the shares in USB. If the shares are stored in clear text, it will be straightforward for an adversary to identify who the owners of any particular block are. By encrypting the shares, the owners of any block are obfuscated so long as multiple blocks have been allocated between snapshots. In this way, our approach safeguards shareholders from being earmarked to be the next target of coercion.

4.3.3 *Organization of the user passwords*

The last design issue concerns the organization of the user passwords. One option is to have only one password in each account and to give every user multiple accounts. Under coercion, a user reveals some of his accounts and tries to hide the remaining ones. However, this simple option fails when the adversary captures all the users of the system. When that happens, the adversary can check whether there are m shares among the surrendered accounts for every allocated block; if not, there must exist more user accounts. This is why we choose to allow multiple passwords (for different security levels) in each user account.

Organizing multiple passwords in a hash chain has been proposed in other stegfs [4, 10, 16], and its one-way property meets our requirements well. Under coercion attack, the disclosure from surrendering t independent passwords is the same as giving up the t lowest-level passwords in a hash chain. Thus, in our system design, the hash chain mechanism is chosen due to the performance and usability benefits gained compared to independent passwords.

5. PLAUSIBLE DENIABILITY OF DRSteg

Having introduced the design of DRSteg, we now quantify the deniability it provides under a spectrum of progressively challenging attack scenarios. Based on the last and most demanding scenario, we then show how to operationalize the DRSteg design so as to sustain the system security above user-specified deniability thresholds. Table 1 summarizes the terms and notations which are used in the analysis.

5.1 Analysis of Deniability

We first expand Equation 1.

$$\mathsf{deny_i} = \Pr(\mathsf{H_i^{dummy}}|\mathsf{e_i}) = \frac{\Pr(\mathsf{e_i}|\mathsf{H_i^{dummy}}) \times \Pr(\mathsf{H_i^{dummy}})}{\Pr(\mathsf{e_i})} \qquad (2)$$

According to our problem formulation in Section 3, the adversary is capable of taking multiple snapshots of the storage content. He may also augment the snapshots with secrets that he coerced from one or more users. The following attack scenarios differ on the amount of secrets thus extracted, and deserve particular attention in deploying DRSteg. These scenarios will be further evaluated in Section 6. In the following analysis, we consider the case where the evidence contains two snapshots. The analysis extends easily to multiple snapshots. Note that Equation 2 implicitly takes the frequency of these snapshots into consideration by evaluating $\mathsf{e_i}$, i.e., the more frequently snapshots are taken, the more information $\mathsf{e_i}$ would include.

5.1.1 *Passive-adversary scenario*

In this scenario, the adversary may be curious and has not resorted to force, or he may not be ready to expose himself just yet. Thus he only relies on the snapshots collected, i.e., the evidence $\mathsf{E} = \mathsf{S}$. By comparing any two recorded snapshots ($\mathsf{s_1}$, $\mathsf{s_2}$), the adversary could observe a lot of user activities, e.g., new blocks being created, deleted, and etc.

Let us first consider the creation of new blocks. A block $\mathsf{blk_i}$ is created between $\mathsf{s_1}$ and $\mathsf{s_2}$ if $\mathsf{flag_i}$ changes from 0 in $\mathsf{s_1}$ to 1 in $\mathsf{s_2}$. Let $\mathsf{crt^{data}}$ represent the net number of data blocks created between $\mathsf{s_1}$ and $\mathsf{s_2}$, and $\mathsf{crt^{dummy}}$ the net number of dummy blocks created in the same period. $\mathsf{ttl_{s_2}}$, $\mathsf{ttl_{s_2}^{dummy}}$, and $\mathsf{ttl_{s_2}^{data}}$ denote, respectively, the total number of allocated blocks, the total number of dummy blocks, and the total number of data blocks in $\mathsf{s_2}$. Given an evidence that $\mathsf{blk_i}$ is newly allocated, the probability that $\mathsf{blk_i}$ is a dummy block in $\mathsf{s_2}$ is calculated with Equation (2) as

$$\mathsf{deny_i} = \frac{\mathsf{crt^{dummy}}}{\mathsf{ttl_{s_2}^{dummy}}} \times \frac{\mathsf{ttl_{s_2}^{dummy}}}{\mathsf{ttl_{s_2}}} \Big/ \frac{\mathsf{crt^{data}} + \mathsf{crt^{dummy}}}{\mathsf{ttl_{s_2}}} = \frac{\mathsf{crt^{dummy}}}{\mathsf{crt^{data}} + \mathsf{crt^{dummy}}}$$

This derivation extends to block deletion and other evidence listed in Table 2. Denoting the number of data/dummy block operations between $\mathsf{s_1}$ and $\mathsf{s_2}$ by $\mathsf{op^{data}}$ and $\mathsf{op^{dummy}}$, the deniability can be calculated as $\mathsf{op^{dummy}}/(\mathsf{op^{data}} + \mathsf{op^{dummy}})$.

For an individual user u in DRSteg, let $\mathsf{op}_u^{\mathsf{data}}$ denote the number of data blocks operated on in $\cup_l D_{u,l}$ between $\mathsf{s_1}$ and $\mathsf{s_2}$, and $\mathsf{op^{dummy}}$ denote the number of dummy blocks operated on in the system. The deniability that DRSteg provides for u under this scenario is expressed as

$$\mathsf{deny_{u,i}} = \frac{\mathsf{op^{dummy}}}{\mathsf{op}_u^{\mathsf{data}} + \mathsf{op^{dummy}}} \qquad (3)$$

Notation	**Explanation**
$\mathsf{S} = \{\mathsf{s}_1, \mathsf{s}_2, \ldots, \mathsf{s}_\mathsf{T}\}$	Snapshots (of the stegfs partitions) taken by the adversary.
$\mathsf{P}_\mathsf{t} = \{\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_\mathsf{t}\}$	Passwords revealed to the adversary under coercion.
$\mathsf{E} = \{\mathsf{e}_\mathsf{i}\} = \mathsf{S} \cup \mathsf{P}' \cup \mathsf{P}_\mathsf{t}$	Evidence possessed by the adversary.
$\mathsf{s}_\mathsf{k} = \{\mathsf{BLK}, \mathsf{USB}, \mathsf{GVT}\}_\mathsf{k}$	$\mathsf{BLK} = \{\mathsf{blk}_\mathsf{i}\}$: Blocks in the stegfs partition (blk_i is the i-th block). $\mathsf{USB} = \{\mathsf{USB}_u\}$: User share boxes ($\mathsf{USB}_u$ is the USB of user u). GVT: Global voting table.
$\mathsf{blk}_\mathsf{i} = \langle \mathsf{text}_\mathsf{i}, \mathsf{flag}_\mathsf{i} \rangle$	text_i: If blk_i is dummy, text_i contains random bits; If blk_i holds user data, $\mathsf{text}_\mathsf{i} = E(\mathsf{p}, \mathsf{plaintext}_\mathsf{i})$ flag_i: A flag indicating whether blk_i has been allocated.
$\mathsf{H}_\mathsf{i}^{\mathsf{dummy}}$, $\mathsf{H}_\mathsf{i}^{\mathsf{data}}$	Hypothesis that blk_i is a dummy/data block in s_T.

Table 1: Summary of notations used

Evidence	**DRSteg operation**
flag_i changes from 0 to 1 and new shares appear in some USBs	Create blk_i as a new dummy or data block
A share of blk_i is moved from USB_u to GVT	User u votes to delete blk_i
flag_i changes from 1 to 0, and blk_i's entry is removed from GVT	Delete blk_i as enough votes are present in GVT
Some combination of the above	Some combination of the above

Table 2: Evidences and the corresponding DRSteg operations

5.1.2 Anonymous-channel scenario

Once the adversary starts to coerce users, by the victim isolation assumption in Section 3, one has to assume that all of the users have been captured and be wary about offering conflicting information to the adversary. In this scenario, we consider a victim u who discloses the passwords for up to level t of his files and attempts to hide his remaining data, when all the other users are compromised ($\mathsf{E} = \mathsf{S} \cup \mathsf{P}' \cup \mathsf{P}_\mathsf{t}$). We assume that all the user requests where sent through an anonymous channel to the storage server, so that the adversary is not able to trace each request to a specific user.

With all the passwords of every user except u, the adversary not only sees all the data of the other users, he also uncovers the dummy blocks for which the ownership is limited to those users. The only outstanding blocks are those on which u holds a share (A_u). Figure 3 illustrates the distinction between various groups of blocks in the system, and also the ones used in the calculation of $\mathsf{deny}_{\mathsf{u,i}}$.

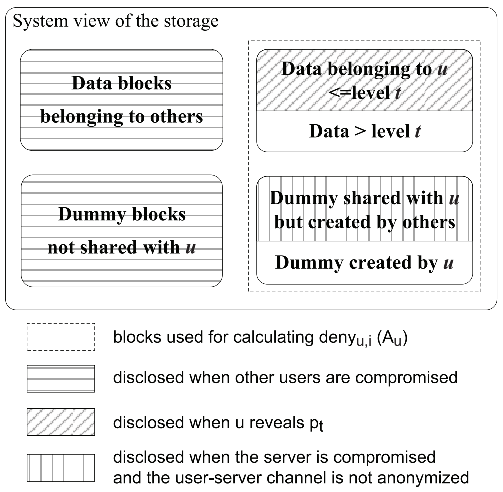

System view of the storage
Data blocks belonging to others
Data belonging to u <=level t
Data > level t
Dummy blocks not shared with u
Dummy shared with u but created by others
Dummy created by u
blocks used for calculating deny_u,i (A_u)
disclosed when other users are compromised
disclosed when u reveals p_t
disclosed when the server is compromised and the user-server channel is not anonymized

Figure 3: System view of allocated blocks

Taking into account the organization of the user data into different password levels n and P_t, operations on data blocks in level t and below are disclosed to the adversary. Let $\mathsf{op}_{u,l}^{\mathsf{data}}$ denote the number of data blocks in $D_{u,l}$, and $\mathsf{op}_u^{\mathsf{dummy}}$ denote the number of dummy blocks recorded in USB_u. The deniability of a user u (who has revealed p_t) is a function of the undisclosed blocks held by him:

$$\mathsf{deny}_{\mathsf{u,i}} = \frac{\mathsf{op}_u^{\mathsf{dummy}}}{\sum_{l>t} \mathsf{op}_{u,l}^{\mathsf{data}} + \mathsf{op}_u^{\mathsf{dummy}}} \qquad (4)$$

The disclosed passwords do not affect $\mathsf{op}_u^{\mathsf{dummy}}$ in the above equation. Therefore, a bigger t improves the deniability for the data of user u being withheld from the adversary. This is intuitive, since a bigger t means that there is less user data to be hidden among the fixed pool of dummy blocks.

5.1.3 Worst-case scenario

When the user-server channel is not anonymized and the storage server is compromised by the adversary, the adversary is able to distinguish the creator from other share holders by monitoring the requests sent to the server. Under such a scenario, a user cannot utilize the dummy blocks that are not created by himself to provide deniability for his secret data (even if he is one of the owners of these dummy blocks). This leads to the worst-case deniability $\mathsf{deny}_{\mathsf{u,i}}$ for DRSteg since $\mathsf{op}_u^{\mathsf{dummy}}$ in Equation 4 only contains dummy blocks created by user u himself.

5.2 α-deniable DRSteg

We now show how to operationalize the dummy manipulation mechanism to secure DRSteg under the worst-case scenario described above. Specifically, we demonstrate how to manipulate dummy data to maintain the deniability above a given threshold α_T, thus making DRSteg α_T-deniable.

5.2.1 Number of Dummy Blocks to Manipulate

Let $\sigma_{u,l} = \mathsf{op}_{u,l}^{\mathsf{dummy}} / \mathsf{op}_{u,l}^{\mathsf{data}}$. The number of dummy blocks operated on by u, $\mathsf{op}_u^{\mathsf{dummy}} = \sum_l \mathsf{op}_{u,l}^{\mathsf{dummy}} = \sum_l \mathsf{op}_{u,l}^{\mathsf{data}} \times \sigma_{u,l}$.

Substituting into Equation (4), we have

$$\mathsf{deny_{u,i}} = \frac{\sum_l(\mathsf{op}^{\mathsf{data}}_{u,l} \times \sigma_{u,l})}{\sum_{l>t}\mathsf{op}^{\mathsf{data}}_{u,l} + \sum_l(\mathsf{op}^{\mathsf{data}}_{u,l} \times \sigma_{u,l})} \tag{5}$$

In order to ensure that every $\mathsf{blk_i} \in A_u$ meets the deniability threshold of α_T no matter which password level user u chooses to surrender, we need

$$\mathsf{deny_{u,i}} = \frac{\sum_l(\mathsf{op}^{\mathsf{data}}_{u,l} \times \sigma_{u,l})}{\sum_l\mathsf{op}^{\mathsf{data}}_{u,l} + \sum_l(\mathsf{op}^{\mathsf{data}}_{u,l} \times \sigma_{u,l})} > \alpha_T$$

Simplifying the above equation, we get

$$\sigma_{u,l} > \frac{\alpha_T}{1-\alpha_T} \tag{6}$$

Since $\sigma_{u,l} = \mathsf{op}^{\mathsf{dummy}}_{u,l}/\mathsf{op}^{\mathsf{data}}_{u,l}$, Equation (6) implies that to achieve the target deniability threshold α_T, the number of dummy blocks manipulated must be at least $\frac{\alpha_T}{1-\alpha_T}$ times $\mathsf{op}^{\mathsf{data}}_{u,l}$, the number of data operations.

5.2.2 Controlling dummy operations

Having determined the number of dummy blocks to manipulate, we give the procedures for controlling the dummy manipulation in DRSteg in order to achieve the deniability configured by users.

There are three types of operations on the dummy blocks – creating, deleting and voting – among which dummy creation is the easiest to control. When a user logs in at security level l, he configures σ_l (which is bigger than $\frac{\alpha_T}{1-\alpha_T}$). If x free blocks are allocated for creating or modifying a secret file, then after a random delay, the DRSteg client creates $x \cdot \sigma_l$ dummy blocks to maintain the deniability.

Deletion is more complex because a user does not know which blocks are really dummy blocks (he can only identify blocks that are not his data, as illustrated in Figure 2). To conceal the deletion of x data blocks, the DRSteg client has to delete $x \cdot \sigma_l$ dummy blocks. This is done by moving the shares of $x \cdot \sigma_l$ randomly selected blocks in $A_u - \cup_l D_{u,l}$ from USB_u to GVT after a random delay. Although some of these $x \cdot \sigma_l$ blocks may be data blocks of other users, the respective data owners will turn these (data) blocks into dummy anyway as explained next.

Now suppose that user u' logs in, and discovers that a block $\mathsf{blk} \in A_{u'}$ has been put up in GVT for deletion. If blk does not contain his data, i.e., if $\mathsf{blk} \in (A_{u'} - \cup_l D_{u',l})$, u' will support the deletion by adding his votes on blk in GVT. If blk is a data block of u' (i.e., $\mathsf{blk} \in \cup_l D_{u,l}$), then u' has to migrate the content to a new block before voting for the deletion. As discussed in Section 4.2, this is to avoid leaving clues that blk contains user data.

5.2.3 Security Discussions

There are several security concerns relating to dummy manipulation. First, in our current design, every block operation is either a direct data operation or the effect of a data operation. Besides introducing random delays, their association could be masked by breaking each of the dummy creations and block deletions into smaller steps and interleaving them with data block operations. In addition, DRSteg could initiate dummy operations independently of data operations. These enhancements will be incorporated in future work.

Second, the parameter $\sigma_{u,l}$ is of special interest to the adversary, who might force the victims to reveal their choices of $\sigma_{u,l}$. With the $\sigma_{u,l}$ values, the adversary may estimate the actual number of data block operations, thus limiting the victims' flexibility to attribute as dummy those data blocks that they are trying to hide. To substantiate his denial in the event of an attack, DRSteg furnishes each user u with a fake $\sigma^{\mathsf{fake}}_{u,t}$ at log-out, where t is the password level that the user is willing to disclose. $\sigma^{\mathsf{fake}}_{u,t}$ is calculated as the ratio between the number of blocks claimed to be dummy (including dummy blocks and hidden data blocks), and the number of revealed data blocks: $\sigma^{\mathsf{fake}}_{u,t} = (\Sigma_{l>t}\mathsf{op}^{\mathsf{data}}_{u,l} + \mathsf{op}^{\mathsf{dummy}}_{u})/\Sigma_{l\le t}\mathsf{op}^{\mathsf{data}}_{u,l}$.

Another potential security threat is, if the adversary is able to take snapshots of the storage content with infinitesimal delay, he may be able to distinguish dummy blocks from data blocks. Troncoso et al. [18] showed that this distinction is possible because data blocks belonging to the same file are often accessed one after another, whereas dummy blocks are accessed individually and are not likely to exhibit the same access pattern. To mitigate against such a threat, one possible solution is to introduce dummy files into DRSteg. A dummy file would span several dummy blocks, which are then accessed sequentially like data blocks. In order to present similar access pattern as data files, dummy files should also be accessed frequently. Such an improvement in dummy file operations is left for future work.

6. EVALUATION

6.1 Empirical Evaluation on Deniability

To investigate DRSteg's ability to maintain user-specified deniability thresholds under multiple-snapshot attacks, we perform an empirical evaluation by re-playing file operations logged in a typical office environment. We deployed a logger to record the file operations (operation type and time) on the computers of 12 graduate students in our lab. Over 9 days, we recorded more than 50,000 *user* file operations[5].

We begin by mirroring the user files of all 12 computers in DRSteg, which add up to about 1 Tbyte of data. We also initialize the same number of dummy blocks, making the original utilization of data blocks 0.5. The shares for data and dummy blocks are distributed randomly among the 12 users. We assume that users are automatically logged out from the stegfs system after some period of inactivity (10 minutes in our experiments), and they login again right before their next observed data operations. For each session, the user enters the password to one of his security levels l (randomly chosen by our simulator) and picks a $\sigma_{u,l}$ value (chosen to follow a power-law distribution $p(\sigma) \propto L(\sigma)\sigma^{-\xi}$ assuming that more users will tend to choose lower σ values to minimize overhead). We set $\alpha_T = 0.4$, $\sigma_{min} = 0.7$ and $\xi = 3.0$ for all users. The parameters and statistics are summarized in Table 3.

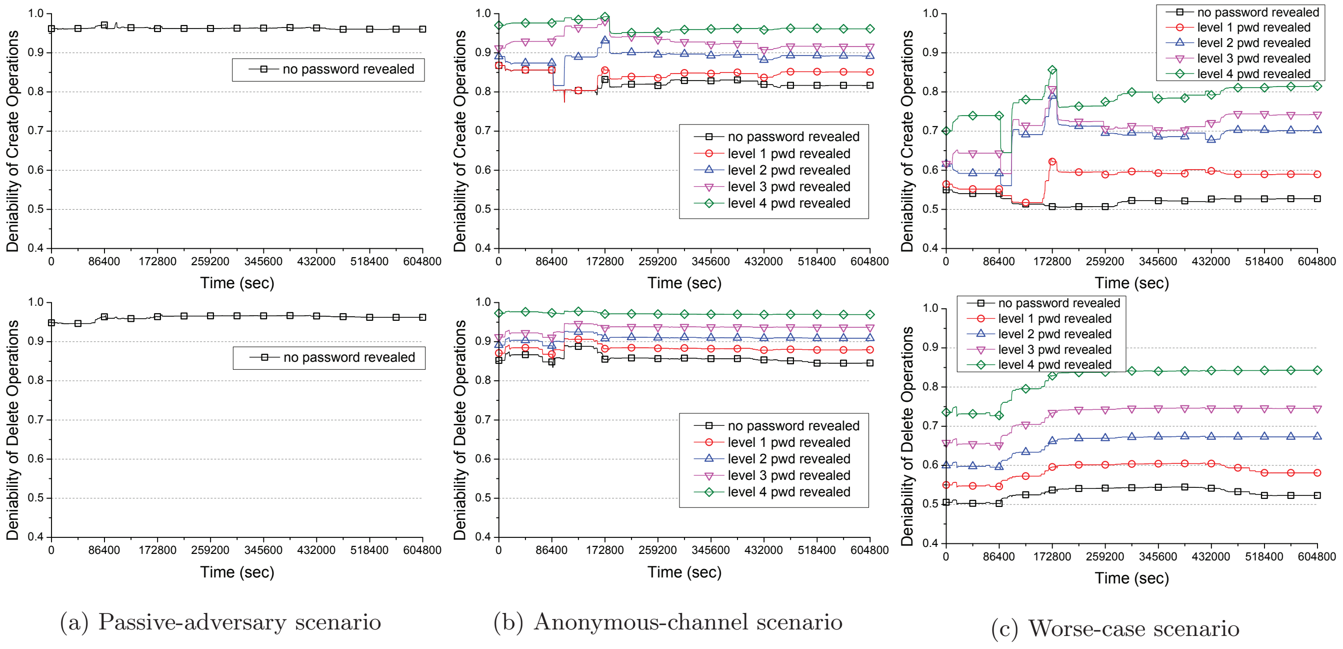
We use the first two days of logs to warm up DRSteg. As the remaining seven days of traces are executed, we take a snapshot of the disk image every 10 minutes. Figure 4 shows the deniability for one of the (randomly chosen) users by comparing each successive snapshot with the first one.

Figure 4(a) shows the deniability under the *passive-adversary* scenario, calculated with Equation 3. The upper graph gives

[5]We assume that the operating system and software programs are not installed in the stegfs partition.

Parameter	Value	User-log Statistics	Value	Simulated DRSteg Statistics	Value
# of users	12	Total logging time	9 days	Initial amt. of data blocks	1011.34 GB
α_T	0.4	# of file operations	50,113	Initial amt. of allocated blocks	2022.68 GB
# of security levels	5	Data blocks created	26.613 GB	# of user sessions	294
Interval before auto logout	10 mins	Data blocks deleted	80.069 GB	Final amt. of data blocks	970.60 GB
Avg. # of shares per block	3	Data blocks modified	160.317 GB	Final amt. of allocated blocks	1995.89 GB

Table 3: Simulation parameters and statistics



(a) Passive-adversary scenario (b) Anonymous-channel scenario (c) Worse-case scenario

Figure 4: Deniability of DRSteg under different scenarios

the deniability with respect to block creation evidence, while the lower is for delete operations. As seen from the graphs, sharing dummy blocks among users enables individual users to enjoy high deniability.

Next, we examine Figure 4(b) for the *anonymous-channel* scenario, which is calculated with Equation 4. Here, the selected user has revealed up to level t of his passwords (the lines in the graphs represent different settings of t), whereas the other users have revealed all their passwords. Since the selected user can only rely on the operations on dummy blocks which are recorded in his UMB, the deniability is lower than that in the previous scenario. Nevertheless, DRSteg still manages to achieve high deniability.

Turning to the *worst-case* scenario where the adversary is aware of the creator of every block, Figure 4(c) shows the deniability levels achieved. In this scenario, deniability is derived solely from operations on the dummy data created by the user himself, which explains the much reduced deniability. Even so, DRSteg manages to keep the deniability above the configured threshold of $\alpha_T = 0.4$.

The deniability for the other 11 users are similar to the results in Figure 4 quantitatively and qualitatively. In particular, the lowest deniability observed for the worst-case attack scenario is 0.46. These results affirm the security property of our proposed DRSteg.

6.2 Implementation and Performance Evaluation

We have implemented DRSteg as a file system module in parallel with ext3 in Linux kernel 2.6, on the client machines which communicate with the shared storage through a server (see Figure 1). The client module manages the blocks in the shared storage automatically according to the password entered by the user. This includes creating new data and dummy blocks (and allocating shares to other owners), voting blocks for deallocation, etc. We explain below how the storage is organized by the system and benchmark the performance of DRSteg.

6.2.1 File system construction

In our DRSteg file system, the (remote) disk storage is partitioned into blocks of 1 Kbyte in size by default. A bitmap tracks the allocation status of the blocks: 1 corresponds to an allocated block and 0 a free block. An allocated block is either a dummy or a data block, both of which appear to contain random patterns.

To accelerate access to directories and files, DRSteg uses a designated storage area, called the *super block* (see Figure 5), to store inode structures so that they can be located efficiently. The super block is essentially a mini-DRSteg system for the addresses of inode roots, and is calved into fixed-size slots that are capable of holding one address each. A slot may be a free slot, a dummy slot, or may contain

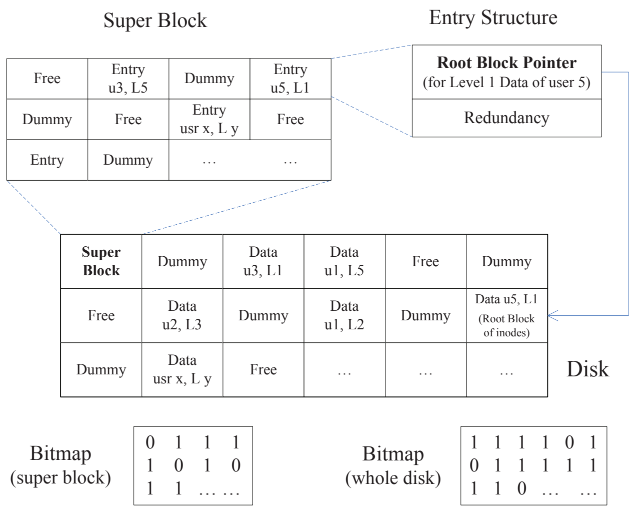


Figure 5: Block organization in DRSteg

the encrypted address of an inode root (with redundancy so that it is distinguishable from random bits upon decryption). Each password level of a user is allocated one slot. Since the super block is expected to be only a few Kbytes in size, it can be scanned quickly to find the inode roots for each user. The super block has its own bitmap to track slot allocation, while it shares the same set of user share boxes and the global voting table with the main file system.

6.2.2 Performance Evaluation

The key parameters of the computing hardware for our experiments are listed in Table 4, while Table 5 summarizes the workload parameters and their default settings.

The first experiment is designed to study how well DRSteg performs. For comparison, we include StegCover, StegRand [4] and NSteg [16] as baselines. StegCover is configured with 20 cover files (the authors recommended 16 to 100 [4]). For StegRand, we use a replication factor of 4 to reduce the probability of data loss [15]. NSteg is set to populate 30% of the disk with dummy blocks during initialization. We also include two settings of the native Linux file system (ext3) in our tests. In the CleanDisk setting, data files are loaded into a freshly formatted native Linux partition, so that the files occupy contiguous disk blocks; with file operations translating to sequential I/Os, CleanDisk gives the best-case timings. In contrast, results of FragDisk are obtained with a well-used ext3 partition in which the free space is fragmented.

In the first experiment, we configure DRSteg with $\sigma_{u,l} = 0.25$, which produces a worst-case deniability of 0.2. For a given concurrency level, we generate file creation requests one after another for each user and measure the elapse time. Figure 6(a) shows the average write time for various file systems, with the number of concurrent users ranging from 1 to 32. Every performance result is averaged over 1000 observations.

The results show that StegCover is the worst performer; this is because each file operation translates into disk I/Os on several cover files. StegRand is also slow because it has to modify all the replicas. DRSteg and NSteg use a bitmap to track the status of disk blocks, so they can ensure data

Parameter	Value
CPU	Intel Duo Core 2.53GHz
RAM	2GB (1GB DDR2-667 x 2)
Hard Disk	SATA 7200rpm, 250 GB with 8MB cache

Table 4: Hardware Parameters

Parameter	Default Value
Capacity of the test partition	40 Gbytes
Size of each disk block	1 Kbytes
Number of blocks for each file	1024
File access pattern	Interleaved

Table 5: Workload Parameters

integrity with just one copy of each data file. Consequently, they are substantially faster than StegCover and StegRand. They are slower than FragDisk though, because they encrypt the protected files block by block and spread them across the disk, resulting in higher fragmentation.

Recall that DRSteg needs to write additional messages into the UMB's during block creation and generate dummy operations dynamically. As the file creation requests in our experiment are issued one after another with no delay, the file system is fully loaded, leaving no idle period for DRSteg to schedule its dummy operations. Thus, the dummy operations add directly to the write times, and the observed timings represent the worst-case performance of DRSteg. For example, with $\sigma_{u,l} = 0.25$ it is roughly 30% slower than NSteg. This is the cost paid by DRSteg to achieve better security protection, compared to NSteg which is not able to relocate its dummy blocks.

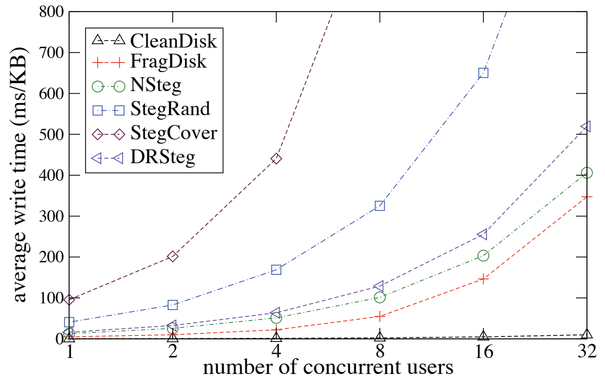
In the second experiment, we investigate the performance of DRSteg under different load conditions. The load condition is determined by various factors, including the σ parameter that controls the amount of dummy operations, the concurrency level, and the activity level of each user. We model the activity level after a Poisson process with mean arrival rate of λ block operations per minute. The results are summarized in Figure 6(b), which plots the average write time against λ for several σ-concurrency combinations.

We first consider the impact of λ. For every σ-concurrency combination, DRSteg's write time is short initially because there are ample lull periods during which dummy operations can be scheduled so as to reduce contention with data operations. Such opportunities diminish with increasing λ, leading to longer write times observed in the figure. Next, we compare the three σ-concurrency combinations with $\sigma = 0.25$. With the same σ and λ settings, raising the concurrency level introduces more contention between the data and dummy operations and lengthens the write time. Similarly, a bigger σ generates more dummy operations to cover the data operations, again resulting in longer write times.

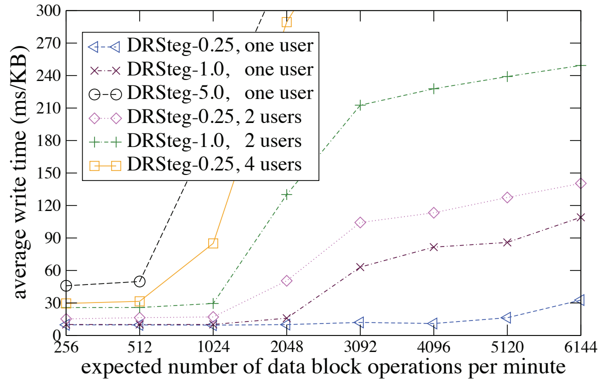
In summary, our experiment demonstrates that DRSteg is capable of striking a wide range of trade-offs between deniability and system performance. If high deniability is required, the file system should be configured with enough resources to prevent it from becoming overloaded. On the other hand, to support a heavy workload, we could configure DRSteg for a lower deniability assurance.

7. CONCLUSION

In this paper, we address the threat to steganographic



(a) Comparing with previous stegfs designs



(b) Trade-off between deniability and performance

Figure 6: Performance evaluation results

file systems (stegfs) that arises when the underlying storage is untrusted and shared by multiple users. In such systems, an adversary could obtain and analyze multiple snapshots of the storage content to deduce the existence of secret user data. To counter the threat, we introduce a Dummy-Relocatable Steganographic (DRSteg) file system that employs novel techniques to share and relocate dummy data at runtime. This enables users to surrender only some of their data, and attribute any unexplained changes across snapshots to dummy operations. The deniability enjoyed by users is configurable individually. DRSteg guarantees the integrity of the protected data, except where users voluntarily overwrite data under duress. A trace-driven simulation confirms the security of our scheme. Further experiments on a Linux prototype demonstrate that DRSteg is able to effectively trade off deniability with system performance.

8. REFERENCES

[1] eCryptfs, a POSIX-compliant enterprise-class stacked cryptographic filesystem for Linux. `https://launchpad.net/ecryptfs`.

[2] Encrypting File System in Windows XP and Windows Server 2003. `http://www.microsoft.com/technet/prodtechnol/winxppro/deploy/cryptfs.mspx`.

[3] R. J. Anderson and E. Biham. Two practical and provably secure block ciphers: Bears and lion. In *Proceedings of the Third International Workshop on Fast Software Encryption*, pages 113–120, 1996.

[4] R. J. Anderson, R. M. Needham, and A. Shamir. The steganographic file system. In *Proceedings of the 2nd International Workshop on Information Hiding*, pages 73–82, 1998.

[5] M. Blaze. A cryptographic file system for unix. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 9–16, 1993.

[6] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 90–104, 1997.

[7] G. Cattaneo, L. Catuogno, A. D. Sorbo, and P. Persiano. The design and implementation of a transparent cryptographic file system for unix. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 199–212, 2001.

[8] C. Diaz, C. Troncoso, and B. Preneel. A framework for the analysis of mix-based steganographic file systems. In *Proceedings of the 13th European Symposium on Research in Computer Security*, pages 428–445, 2008.

[9] J. Domingo-Ferrer and M. Bras-Amorós. A shared steganographic file system with error correction. In *Proceedings of the 5th International Conference on Modeling Decisions for Artificial Intelligence*, pages 227–238, 2008.

[10] C. Giefer and J. Letchner. Mojitos: A distributed steganographic file system. Technical report, Univerisity of Washington, 2004.

[11] F. Graf and S. D. Wolthusen. A capability-based transparent cryptographic file system. In *Proceedings of the 2005 International Conference on Cyberworlds*, pages 101–108, 2005.

[12] S. Hand and T. Roscoe. Mnemosyne: Peer-to-peer steganographic storage. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, pages 130–140, 2002.

[13] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[14] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11), 1981.

[15] A. D. McDonald and M. G. Kuhn. StegFS: A steganographic file system for Linux. In *Proceedings of the 3rd International Workshop on Information Hiding*, pages 462–477, 2000.

[16] H. Pang, K.-L. Tan, and X. Zhou. StegFS: A steganographic file system. In *Proceedings of the 19th International Conference on Data Engineering*, pages 657–668, 2003.

[17] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 199–212, 2009.

[18] C. Troncoso, C. Diaz, O. Dunkelman, and B. Preneel. Traffic analysis attacks on a continuously-observable steganographic file system. In *Proceedings of the 9th Information Hiding*, pages 220–236, 2008.

[19] C. P. Wright, M. C. Martino, and E. Zadok. NCryptfs: A secure and convenient cryptographic file system. In *Proceedings of the 2003 USENIX Annual Technical Conference*, pages 197–210, 2003.

[20] X. Zhou, H. Pang, and K.-L. Tan. Hiding data accesses in steganographic file system. In *Proceedings of the 20th International Conference on Data Engineering*, pages 572–583, 2004.