A Framework for the Analysis of Mix-Based Steganographic File Systems

Claudia Diaz, Carmela Troncoso, and Bart Preneel

K.U.Leuven/IBBT – ESAT/COSIC, Belgium
{Claudia.Diaz,Carmela.Troncoso,Bart.Preneel}@esat.kuleuven.be

Abstract. The goal of Steganographic File Systems (SFSs) is to protect users from coercion attacks by providing plausible deniability on the existence of hidden files. We consider an adversary who can monitor changes in the file store and use this information to look for hidden files when coercing the user. We outline a high-level SFS architecture that uses a local mix to relocate files in the remote store, and thus prevent known attacks [TDDP07] that rely on low-entropy relocations. We define probabilistic metrics for unobservability and (plausible) deniability, present an analytical framework to extract evidence of hidden files from the adversary's observation (before and after coercion,) and show in a experimental setup how this evidence can be used to reduce deniability. This work is a first step towards understanding and addressing the security requirements of SFSs operating under the considered threat model, of relevance in scenarios such as remote stores managed by semi-trusted parties, or distributed peer-to-peer SFSs.

1 Introduction

Steganographic File Systems (SFSs) were first proposed by Anderson et al. in [ANS98]. The goal of these systems is to conceal not just the content of the files they store, but the very existence of some of those files. Steganography is required to protect users from coercion attacks, where they are forced (e.g., under the threat of violence) to disclose their cryptographic keys to the attacker if the existence of files is known. To protect against these attacks, SFSs typically provide the user with several security levels, each associated with a cryptographic key. In case of coercion, the user provides keys to some security levels (thus revealing some files) without leaking information on the existence of hidden security levels (containing hidden files that are undistinguishable from random data.)

Some of the previous SFS proposals [ANS98, MK99] were designed to protect against attackers who obtain a few snapshots of the file store sufficiently spaced in time (e.g., customs inspection performed when entering and leaving a country.) However, adversaries who permanently monitor the file store are of practical relevance. For example, the model in [ZPT04] considers a shared multi-user file store where a malicious user or system administrator monitors store accesses. And this threat model is particularly relevant for distributed

S. Jajodia, and J. Lopez (Eds.): ESORICS 2008, LNCS 5283, pp. 428–445, 2008.

peer-to-peer SFSs [GL04, HR02], given that any eavesdropper in the vicinity of the user can monitor her connections to other peers (each storing some file blocks,) and use the traffic information to obtain evidence of hidden files.

StegFS [ZPT04] is, to the best of our knowledge, the only previous proposal of SFS that implements measures to protect against this adversary model. StegFS avoids simple location access frequency analysis by continuously generating dummy accesses to random locations in the store, and by relocating file blocks every time they are accessed. In spite of these measures though, previous work [TDDP07] has shown that the low-entropy block relocation technique used in [ZPT04] enables very powerful traffic analysis attacks capable of uncovering virtually any "hidden" files. In order to counter these traffic analysis attacks, SFSs subject to continuous surveillance require some form of high-entropy block relocation strategy. Such relocation strategy can be achieved using mixes [Cha81], a well-known mechanism for implementing anonymous email services [DDM03, MCPS03]. Besides cryptographically changing the appearance of messages, mixes alter the message flow to prevent traffic analysis attacks based on input and output order, a useful property to introduce uncertainty in the block relocation process. This paper develops a framework for analyzing mix-based SFSs, and its purpose is to serve as basis for their design and evaluation. We define probabilistic metrics that characterize the security of an SFS by its unobservability and (plausible) deniability, present methods to analyze whether evidence of hidden files is leaked to the adversary, and validate our analysis through experiments. Our results highlight the power of traffic analysis techniques and the challenge of achieving acceptable levels of security against adversaries who can monitor SFS accesses.

The rest of the paper is organized as follows. Section 2 outlines MixSFS, a high-level SFS architecture that uses a local pool mix for relocating data blocks in a remote store. The adversary model is described in detail in Sect. 3. The probabilistic metrics used to characterize the security of MixSFSs are defined in Sect. 4; and Sect. 5 shows how they can be used to evaluate the security of MixSFS architectures. Finally, we present our conclusions in Sect. 6.

2 MixSFS Architecture

We assume the user has a set UK of secret keys, $UK = \{uk_s : s = 1 \dots S\}$, where each key uk_s corresponds to a ***security level*** s. Files in the system are classified according to their security level, such that a file f_s in level s is encrypted under key uk_s. For convenience, we assume that user keys are hierarchical [AT83], such that a key uk_s decrypts all files in security levels s and lower. The view of the user on the MixSFS file system contents depends on the level of security with which she logged in. For a security level s, we call ***visible files*** those files in level s or lower, and ***hidden files*** those (if any) in levels s' higher than s. The design goal of MixSFS is to make it impossible to distinguish whether or not s is the highest existing security level—and thus, whether or not there are any hidden files in addition to the visible ones. Transparently to the user, MixSFS

stores files in blocks of fixed size: large files occupy a few blocks and small files of the same security level are packed together in one block. We call ***file blocks*** the blocks that contain (encrypted) file data (file blocks can belong to visible or hidden files,) and ***dummy blocks*** empty blocks filled with random data.

Main Architectural Components. MixSFS has an architecture as depicted in Fig. 1. The system comprises two separate parts: the user local computer (accessible to the attacker only when coercing the user) and a remote store (always visible to the attacker,) which is divided in N blocks of equal size. The user local computer contains three main MixSFS components, namely:

- An ***agent*** that runs the MixSFS software and provides the user with an interface for file management. The agent translates the user's file requests into block-level operations, and generates automatic (dummy) accesses when the user is idle.
- A memory ***pool*** with capacity for P blocks. The pool is used by the agent to mix blocks and relocate them in the store.
- A ***table*** with $N + P$ entries (one per block storage location,) containing block meta-data. The table is used (and updated) by the agent to keep track of blocks and to manage files. Both the pool and the table are available to the attacker when she coerces the user.

Table. The table is indexed by location, and each entry contains the following fields (as shown in Fig.2):

- A ***location*** index i, with $1 \leq i \leq N$ for (remote) block store locations, and $N + 1 \leq i \leq N + P$ for (local) pool locations.
- A hash $\boldsymbol{H}(\mathrm{A})$ of the block A stored in that location, used to check that no error or active attack has corrupted the block since it was last stored.
- A randomly generated, one-time ***block key*** bk_A that is updated every time the block is accessed, and whose purpose is to provide forward and backward

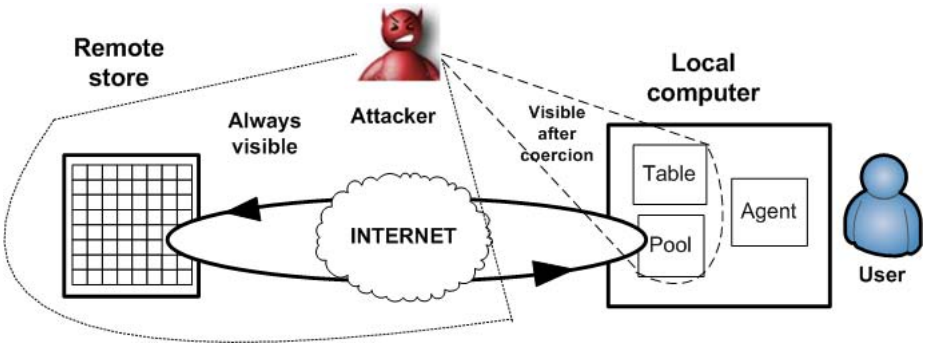

Fig. 1. MixSFS architecture

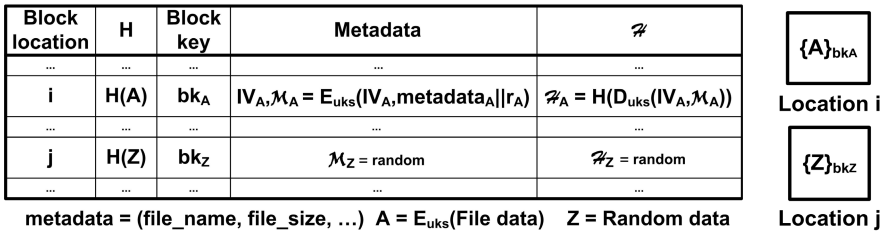

Fig. 2. Table entries (left) and storage blocks (right)

security. When coercing the user, the adversary obtains the current list of block keys from the table. Previous block keys however, are unavailable, meaning that old versions of the blocks cannot be decrypted. Future block keys will be randomly generated, so backward security is also provided.

- A ***metadata*** field that contains a random vector IV_A and $\mathcal{M}_A = \texttt{random}$ data if A is an empty block. If A contains file data of security level s, this field contains $\mathcal{M}_A = E_{uk_s}(IV_A, \texttt{metadata}_A||r_A)$, a randomized encryption of the metadata needed by the agent to manage A's content. The metadata is padded to a fixed length with a random string r_A, and encrypted under key uk_s using IV_A as initialization vector. Note that this is the only table field that is encrypted, and a secure mode of operation (e.g., CBC) must be used to ensure that it leaks no information on blocks that share similar metadata encrypted under the same key.
- The last field contains $\mathcal{H}_A = \texttt{random}$ if A is empty; if A belongs to a file f_s then $\mathcal{H}_A = H(D_{uk_s}(IV_A, \mathcal{M}_A))$ is a hash of the decryption of $\mathcal{M}_A$.

We assume that the table is locally stored securely, and only accessible to the adversary while coercing the user. When the user logs into MixSFS with uk_s, the agent loads the table and trial-decrypts every $\mathcal{M}_A$ field. If $\mathcal{H}_A = H(D_{uk_s}(IV_A, \mathcal{M}_A))$, then A is a file block and the decryption of $\mathcal{M}_A$ provides $\texttt{metadata}_A$. If $\mathcal{H}_A \neq H(D_{uk_s}(IV_A, \mathcal{M}_A))$, then A is considered empty (this happens either because A is a dummy block, or because it belongs to a hidden file $f_{s'}$ in level $s' > s$.)

Agent. Upon login, the user provides uk_s to the agent, with which the agent obtains the metadata of files in levels s and lower. The agent provides the user with an interface to operate (read, write, create and delete) on visible files, while making block-level operations transparent. To execute the ***file operations***, the agent assembles and disassembles files into blocks, taking care of the ***block redundancy***; decides which block to access next, and it cryptographically transforms and relocates blocks in each ***access cycle***. We assume the agent can securely generate random numbers, and securely delete any session data in RAM at log out.

Block Redundancy and File Operations. A user logged in with security level s that increases the size of a visible file f_s or creates a new one, risks

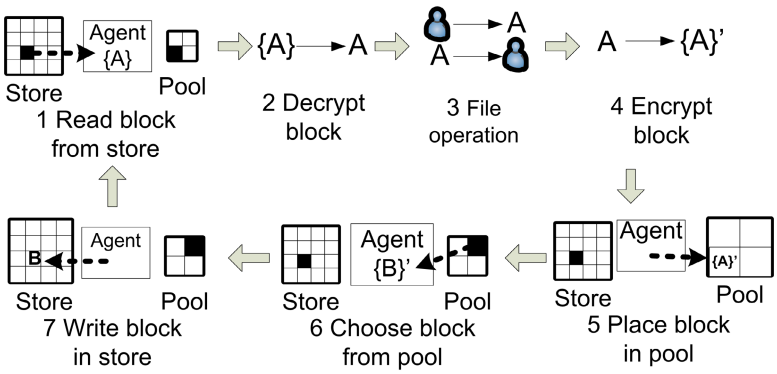

Fig. 3. Access cycle

overwriting the blocks of a hidden file $f_{s'}$ in security level $s' > s$, as these blocks appear empty. In order to improve file resilience towards block losses, MixSFS adds redundancy to file blocks with an erasure code [Rab89, Riz97] that is applied on the plaintext file. A (n, m) *optimal erasure code* converts an original m-block file into $n > m$ encoded blocks, such that any combination of m encoded blocks suffices to recover the file (i.e., up to $n - m$ file blocks may be lost) and the surviving blocks can be used to regenerate the lost ones. For multi-block files, the individual coded blocks do not follow any order of have any meaning by themselves. For single-block files, the coding relies on pure replication for block redundancy purposes. The agent translates files into encoded blocks and vice versa.

We classify operations on a file f_s of size (n, m) in two categories: ***file read*** and ***file update*** (which includes file creation, file deletion and file write operations.) For file reads, any subset $\mathcal{B}_f = \{b : b \in f_s\}$ of $|\mathcal{B}_f| = m$ blocks is sufficient to complete the operation; while $|\mathcal{B}_f| = n$ are needed to complete a file update (requires updating all redundant blocks.) Let us call $\mathcal{P}$ the set of blocks available locally in the pool. For any $b \in \mathcal{B}_f$ such that $b \in \mathcal{P}$, b is immediately (and unobservably) operated on, and $\mathcal{B} = \{b \in \mathcal{B}_f : b \notin \mathcal{P}\}$ is the set of blocks in $\mathcal{B}_f$ that need to be fetched from the store to complete the file operation.

File and Dummy Block Access Strategies. Once the agent has determined the set $\mathcal{B}$ of remote blocks it needs to complete a file operation, it could sequentially fetch every $b \in \mathcal{B}$, to finish the operation as fast as possible, but this would also facilitate the traffic analysis presented in Sect. 5. Instead, MixSFS uses a ***file access strategy*** that completes the operation in $|\mathcal{B}| \cdot e^{-1}$ access cycles on average, where e is the operation *efficiency*: for each of the blocks $b \in \mathcal{B}$ the agent flips a biased coin, and with probability e it fetches b from the store; with probability $1 - e$, it performs a dummy access on a store location selected at random, and flips the coin again, until b has been accessed.

When the agent does not have any pending file operation, it generates automatic dummy accesses to the store. The agent's strategy for deciding which

store locations to access while the user is idle is very important for concealing actual file operations. In this paper, we have tested a ***uniform dummy strategy*** (i.e., the agent selects the next remote block location $i : 1 \leq i \leq N$ uniformly at random) in order to better understand how file accesses generate evidence that is distinguishable from white noise, and how this can be used by the adversary to coerce the user. More sophisticated dummy access strategies that emulate file operations can be designed and tested extending the analysis presented in Sect. 5, but for reasons of space limitation they are not included in this paper and are left as a subject of future work.

Access Cycle. Whenever MixSFS is running, the agent accesses store locations at a rate independently of user activity, so that user file requests cannot be inferred from fluctuations in the access rate. All types of access cycles (read, update, or dummy) consist of the same basic steps, the only exception being the third step, that is performed if the block is part of a file (read or update) operation, but skipped in dummy accesses. Figure 3 illustrates the steps of an access cycle:

1. **Read Block from Store Location.** The agent chooses, according to its access strategy, a storage location i ($1 \leq i \leq N$,) and reads its content $\{A\} = E_{bk_A}(A)$. If $A \in f_s$, then $A = E_{uk_s}(\texttt{data})$. And $A = \texttt{random}$ if it is a dummy block (or a hidden file block,) as shown in Fig. 2 (right.)
2. **Decrypt Block.** The agent decrypts $\{A\}$ with the one-time block key bk_A, and verifies its integrity by computing the hash $H(A)$ and comparing it to the value stored in table entry i.
3. **[Optional] File Operation.** If the block access is part of a file operation, then A is either (1) part of a file read: A is further decrypted with uk_s, and the file data is passed in plaintext to the user; or (2) part of a file update: A is overwritten with an (encrypted) updated version of the data. Additionally, $H(A)$ is updated and the same applies to $\mathcal{M}_A$ and $\mathcal{H}_A$ if A's metadata has changed with the update. These operations are internal to the user's computer and unobservable to the adversary.
4. **Encrypt Block.** The agent generates a new random bk'_A, and uses it to encrypt A (which is unchanged unless it is part of a file being updated,) so that the new $\{A\}' = E_{bk'_A}(A)$ cannot be linked through its appearance to its old version $\{A\}$.
5. **Place Block in the Pool.** The pool contains $P - 1$ blocks and an empty location $p : 1 \leq p \leq P$, where $\{A\}'$ is placed. The table entry for pool location p (table index $N + p$) is updated with $\{A\}'$ metadata, including the new bk'_A, and updates in $H(A)$, $\mathcal{M}_A$, and $\mathcal{H}_A$ if appropriate.
6. **Select New Block from Pool.** The agent chooses uniformly at random a pool location $p' : 1 \leq p' \leq P$, and reads the block $\{B\}'$ it contains.
7. **Write New Block to Store Location.** $\{B\}'$ is written to location i of the store, and the table entries i and $N + p'$ are updated; i.e., B's information is moved to table entry i, and pool position p' becomes the empty space in the pool for the next cycle. Note that the pool contains $P - 1$ blocks at all times except for step 5 of the access cycle, where it contains P blocks.

3 Adversary Model

The goal of an SFS is to protect the user against coercion attacks by providing plausible deniability on the existence of files. In a ***coercion attack***, the adversary forces the user to disclose the keys to access her data. The user willingly provides key uk_s and reveals s security levels, and the adversary inspects the system and uses any available information (obtained both before and after coercion) to determine if there are any remaining (hidden) files. We say an SFS provides ***plausible deniability*** if with all available information the adversary is not able to determine the existence of hidden files. We propose in the next section a metric for plausible deniability, and develop in Sect. 5 a method to test the protection offered by MixSFS against this attacker.

Previous SFSs [ANS98, MK99] are designed to protect against coercive attacks where the only information available to the adversary are a few snapshots of the store contents taken under coercion, and sufficiently spaced in time. These systems however, fail to protect the user if the adversary is allowed take as many snapshots as desired, separated by arbitrarily small amounts of time, prior to coercion: the analysis of which locations had their content modified in between snapshots provides the attacker with valuable evidence on the existence and location of hidden files—depriving the user of plausible deniability. This paper considers a rather strong adversary model, similar to [TDDP07, ZPT04], that passively monitors the store to accumulate evidence prior to coercion (note that our system is secure towards the weaker adversary considered in previous work.) We assume the adversary records the (encrypted) contents of the store at all moments, as well as the (temporal) sequence of accessed locations, on which she performs traffic analysis in real-time.

Active attacks that compromise the integrity of blocks are detectable by MixSFS: if a block A has been modified, its hash no longer matches the $H(A)$ stored in the table. We assume that the adversary has incentives to stay undetected before coercion, and therefore, only passive attacks are taken into account.

4 Security Metrics

We characterize the security of MixSFS by two properties that we define probabilistically, ***unobservability*** (see [PH01] for a more general definition) and ***plausible deniability***. To formalize these notions, let us first introduce the notation:

- Let Ψ be the set of all possible sequences of store location accesses; and $\psi \in \Psi$ be a particular sequence seen by the adversary (evidence obtained prior to coercing the user.)
- Let Φ denote the set of all possible internal states of MixSFS' pool and table; and $\phi \in \Phi$ be a particular state seen by the adversary after coercing the user and obtaining key uk_s.
- Let H_0 and H_1 be, respectively, the hypotheses that the observed MixSFS activity was generated by dummy cycles (H_0); or by file operations (H_1.)
- Let $\mathcal{U}$ denote unobservability, and let $\mathcal{D}$ denote deniability.

Definition 1. *We define* $\Psi_0 \subset \Psi$ *as* $\Psi_0 = \{\psi : \Pr(\psi|H_0) > \Pr(\psi|H_1)\}$*; and* $\Psi_1 \subset \Psi$ *as* $\Psi_1 = \{\psi : \Pr(\psi|H_1) > \Pr(\psi|H_0)\}$*. Note that* $\Psi_0 \cup \Psi_1 = \Psi$*, and* $\Psi_0 \cap \Psi_1 = \emptyset$

Definition 2. *We define* $\Phi_0 \subset \Phi$ *as* $\Phi_0 = \{\phi : \Pr(\phi|H_0) > \Pr(\phi|H_1)\}$*; and* $\Phi_1 \subset \Phi$ *as* $\Phi_1 = \{\phi : \Pr(\phi|H_1) > \Pr(\phi|H_0)\}$*. Note that* $\Phi_0 \cup \Phi_1 = \Phi$*, and* $\Phi_0 \cap \Phi_1 = \emptyset$

We say that file operations are unobservable if they generate evidence ψ that the adversary believes is most likely the result of dummy activity (i.e., $\psi \in \Psi_0$.)

Definition 3. *We define* ***unobservability*** *as the probability of a file operation being undetected by the adversary:*

$$\mathcal{U} = \Pr(\psi \in \Psi_0|H_1) = 1 - \Pr(\psi \in \Psi_1|H_1) \tag{1}$$

At any point in time, the adversary may coerce the user and obtain evidence ϕ from inspecting MixSFS' pool and table. The goal of the attacker is to use ψ and ϕ to check if there is any hidden file $f_{s'}$ for which the user has not provided the keys. We say the user has plausible deniability if under coercion she can prove that ψ and ϕ are plausibly consistent with her claim that no $f_{s'}$ exists. To evaluate plausible deniability in MixSFS we examine the worst-case scenario, in which coercion happens just as the user has made an operation on $f_{s'}$ that provided the adversary with ψ and ϕ. We then analyze whether the user can plausibly claim that ψ and ϕ are the result of dummy activity.

Definition 4. *Given evidence* ψ *and* ϕ*, we define* ***deniability*** *as the scaled posterior probability (obtained with Bayes' theorem) that* ψ *and* ϕ *have been generated by dummy activity:*

$$\mathcal{D} = \min\{1, 2 \cdot \Pr(H_0|\psi, \phi)\} \tag{2}$$

We define a ***plausibility threshold*** δ*,* $0 < \delta \leq 1$*, such that deniability is plausible if* $\mathcal{D} \geq \delta$*. We define* ***plausible deniability*** *as* $\mathcal{PD} = \Pr(\mathcal{D} \geq \delta)$*.*

We scale $\Pr(H_0|\psi, \phi)$ by multiplying by two so that $\mathcal{D} = 1$ when there is perfect undistinguishability; i.e., $\Pr(H_0|\psi, \phi) = \Pr(H_1|\psi, \phi) = 0.5$. In cases where ψ and ϕ seem most likely the result of dummy activity (i.e., $\Pr(H_0|\psi, \phi) > \Pr(H_1|\psi, \phi)$,) we also consider that $\mathcal{D} = 1$.

The value of δ depends on the security needs of the user. In some cases (e.g., evidence in court,) it may be enough for the user to prove that there is a small chance (e.g., $\delta = 0.01$) that no $f_{s'}$ exists for the attacker to fail; while in others she may need higher values of δ to be safe (e.g., if the adversary is willing to use torture if $f_{s'}$ is more likely to exist than not, then $\delta = 1$.) The security goal of MixSFS is to ensure that $\mathcal{PD} = 1$ for the δ required by the user. The analysis presented in the next section can be used to determine the configuration and conditions under which MixSFS provides $\mathcal{PD} = 1$.

Note that the values of $\mathcal{U}$ and $\mathcal{D}$ are independent, even if only evidence ψ is taken into account. Consider two cases where, for simplicity, we assume that ϕ does no provide any information (i.e., $\Pr(\phi|H_0) = \Pr(\phi|H_1)$.) In case (a) the attacker obtains $\psi_a \in \Psi_1$ with $\Pr(\psi_a|H_0) = 0.1$ and $\Pr(\psi_a|H_1) = 0.2$. While in case (b) the attacker obtains $\psi_b \in \Psi_1$ with $\Pr(\psi_b|H_0) = 0.5$ and $\Pr(\psi_b|H_1) = 1$. Applying the definitions (1) and (2,) we obtain that $\mathcal{U}_a = 0.8$ and $\mathcal{U}_b = 0$, while in both cases the deniability is $\mathcal{D}_a = \mathcal{D}_b = 2/3$. The intuition behind this is the following: in the first case, the adversary only detects 20% of the file operations, and once suspicious evidence $\psi_a \in \Psi_1$ is detected, there are 33% chances that ψ_a was generated by dummy traffic. In the second case, every time the user operates on a file the adversary gets $\psi_b \in \Psi_1$ (i.e., no unobservability,) but half the dummy-generated sequences are also classified as $\psi_b \in \Psi_1$, so the level of deniability is the same as in the first case.

5 Evaluation of Traffic Analysis Resistance

5.1 How to Extract the Information from ψ

The starting point for performing traffic analysis is the evidence ψ obtained by the adversary prior to time t_c, the moment of coercion. We consider time discrete, with each time unit corresponding to an access cycle, and refer to Sect. 2 for the steps of the access cycle. ψ is the sequence of accesses to block locations in the remote store, and we denote by $\psi(t) : 1 \leq \psi(t) \leq N$, the store location accessed in cycle t, with $0 \leq t \leq t_c$. In previous work [TDDP07] it was shown that the analysis of $\psi(t)$ can reveal the location of hidden files in StegFS [ZPT04] (in spite of constant rate dummy accesses to the store) due to low-entropy block relocations.

MixSFS introduces high-entropy block relocation by using its local pool to mix blocks, and therefore methods such as [TDDP07] are not powerful enough to analyze MixSFS' relocation mechanism. Pool mixes have been analyzed in the context of anonymous communication, and we draw on the literature [DP04] as base for our probabilistic analysis of mixes. In anonymous communication however, a potentially infinite number of messages pass once through the mix; while in MixSFS a finite number N of locations in the store are repeatedly accessed. In this paper we extend existing mix analysis techniques to capture the *feedback* induced by repeatedly accessing locations, and show that our function $q_{\psi(t)}(t)$ is a useful tool to detect correlations induced by file operations. We first define the following notation:

- Let $\mathcal{B}$ be a set of blocks of interest of size $|\mathcal{B}|$.
- Let $q_{loc}(t)$ be the probability that at the end of cycle t, any block $b \in \mathcal{B}$ is in the remote store location loc, and $q_{\psi(t)}(t)$ denote this probability for the location $loc = \psi(t)$ accessed in cycle t.
- Let $E_{pool}(t)$ be the expected number of blocks $b \in \mathcal{B}$ in the pool (of size P) at the end of cycle t, $0 \leq E_{pool}(t) \leq \min(|\mathcal{B}|, P - 1)$.

Let us assume that before cycle t, location $loc = \psi(t)$ contains any block of interest $b \in \mathcal{B}$ with probability $q_{\psi(t)}(t-1)$. In the first step of cycle t the block in $\psi(t)$ is read, and placed in the pool (in step 5 of access cycle t.) If $\psi(t)$ contained a block $b \in \mathcal{B}$ with probability $q_{\psi(t)}(t-1)$ before t, the expected number of blocks $b \in \mathcal{B}$ in the pool increases by $q_{\psi(t)}(t-1)$. At the end of cycle t, a block b' is chosen uniformly at random from the pool and stored in $\psi(t)$. The updated probability of $\psi(t)$ containing $b' \in \mathcal{B}$ is given by:

$$q_{\psi(t)}(t) = \frac{1}{P}[E_{pool}(t-1) + q_{\psi(t)}(t-1)] \tag{3}$$

And the variation of $E_{pool}(t)$ from cycle $t-1$ to cycle t is:

$$E_{pool}(t) = E_{pool}(t-1) + q_{\psi(t)}(t-1) - q_{\psi(t)}(t) \tag{4}$$

Analysis for One Block. Let us consider $\mathcal{B}$ with $|\mathcal{B}| = 1$, such that the only block $b \in \mathcal{B}$ is known to be in location $loc = \psi(t_0)$ until it is accessed in cycle t_0. At $t_0 - 1$, the initial probability distribution describing the location of b is $q_{\psi(t_0)}(t_0 - 1) = 1$ for position $\psi(t_0)$, and $q_{\psi(t)}(t_0 - 1) = 0$ for $\psi(t) \neq \psi(t_0)$. Note that $q_{loc}(t) = q_{loc}(t-1)$ if the location is not accessed in t (i.e., if $loc \neq \psi(t)$,) thus when location $\psi(t)$ is accessed for the first time in cycle $t > t_0$, $q_{\psi(t)}(t-1) = q_{\psi(t)}(t_0 - 1)$. Let us assume that the sequence ψ of accesses between t_0 and $t_1 - 1$ is such that no location is accessed more than once; i.e., $\psi(t) \neq \psi(t')$, $\forall t \neq t' : t_0 \leq t, t' < t_1$. Applying equations (3) and (4,) and taking into account that $q_{\psi(t_0)}(t_0 - 1) = 1$, and $q_{\psi(t)}(t-1) = 0$ for $t_0 < t < t_1$ we obtain:

$$q_{\psi(t)}(t) = \frac{1}{P}(\frac{P-1}{P})^{t-t_0}, \;\; t_0 \leq t < t_1$$

Intuitively, the meaning of $q_{\psi(t)}(t)$ is the following: b enters the pool in t_0, and with probability $\frac{1}{P}$ it is written to $\psi(t_0)$, while it stays in the pool until $t_0 + 1$ with probability $\frac{P-1}{P}$. If $\psi(t_0+1) \neq \psi(t_0)$, then $q_{\psi(t_0+1)}(t_0) = 0$ (i.e., nothing is added to $E_{pool}(t_0 + 1)$,) and the block written to $\psi(t_0 + 1)$ contains b if b stayed in the pool in cycle t_0 (with probability $\frac{P-1}{P}$,) *and* was selected in step 6 of cycle $t_0 + 1$ (with probability $\frac{1}{P}$.) The block b is in the pool at the end of $t_0 + 1$ with probability $(\frac{P-1}{P})^2$.

Let $\psi(t_1)$ be the first location that is accessed twice since t_0; i.e., $\exists t'$, $t_0 \leq t' < t_1 : \psi(t_1) = \psi(t')$. When $\psi(t_1) = \psi(t')$ is accessed in cycle t_1, it contains b with probability $q_{\psi(t_1)}(t_1 - 1) = q_{\psi(t')}(t')$, and therefore after reading $\psi(t_1)$'s content the probability that b is in the pool increases by $q_{\psi(t')}(t')$. Consequently, at the end of cycle t_1, location $\psi(t_1)$ contains b with probability:

$$q_{\psi(t_1)}(t_1) = \frac{1}{P}[(\frac{P-1}{P})^{t_1-t_0} + \frac{1}{P}(\frac{P-1}{P})^{t'-t_0}]$$

The next cycle $t_2 > t_1$ when location $\psi(t_2) = \psi(t_1) = \psi(t')$ is accessed a third time, the probability that is added to the pool is $q_{\psi(t_2)}(t_2 - 1) = q_{\psi(t_1)}(t_1)$. The effect of feeding $q_{\psi(t)}(t-1)$ back to the pool is that $q_{\psi(t)}(t)$ grows when accessing

locations that contain b with probability $q_{\psi(t)}(t-1) > q_{\psi(t-1)}(t-1)$. Figure 4 (left) shows an example for MixSFS with $N = 951$ remote blocks and a pool of size $P = 50$; i.e., total capacity of $N+P-1 = 1000$ blocks (these are the default N and P used in all our experiments.) We can see that until $t_1 = 70$, $q_{\psi(t)}(t)$ follows a geometric distribution. At t_1, the same location $\psi(t')$ that was accessed at $t' = 33$ is chosen for the second time since $t_0 = 0$, causing a bump in $q_{\psi(t)}(t)$. As $t \to \infty$, b disperses across locations becoming more uniformly distributed, and it stabilizes when:

$$q_{\psi(t)}(t) = q_{\psi(t)}(t-1) = q_{\psi(t-1)}(t-1) = \frac{1}{N+P-1} \ , \quad t \to \infty$$

$$E_{pool}(t) = E_{pool}(t-1) = \frac{P-1}{N+P-1} \ , \quad t \to \infty$$

Analysis for File Operations. Let us now consider a file operation that starts at t_0 and requires fetching $|\mathcal{B}| > 1$ blocks of interest from the store (i.e., initially $E_{pool}(t_0 - 1) = 0$,) and let e be the efficiency of the operation (see *file access strategy* in Sect. 2.) We assume that $|\mathcal{B}|$, e and t_0 are known, but not the locations of the blocks $b \in \mathcal{B}$. If the efficiency $e = 1$, then the agent selects the blocks $b \in \mathcal{B}$ sequentially, thus the adversary can infer that the blocks of interest are those in locations $\psi(t_0) \ldots \psi(t_0 + |\mathcal{B}| - 1)$. In this case, $q_{\psi(t)}(t_0 - 1) = 1$ for $\psi(t) : t_0 \leq t < t_0 + |\mathcal{B}|$, and $q_{\psi(t)}(t_0 - 1) = 0$ otherwise. We now examine the case where the file operation efficiency is $0 < e < 1$.

Let us define $n(t)$, a function that counts the number of fresh, distinct locations accessed before cycle t, with $t \geq t_0$:

$$n(t) = \begin{cases} 0 & \text{if} \quad t = t_0 \\ n(t-1) + 1 & \text{if} \quad \forall t' : t_0 \leq t' < t, \ \ \psi(t) \neq \psi(t') \\ n(t-1) & \text{if} \quad \exists t' : t_0 \leq t' < t, \ \ \psi(t) = \psi(t') \\ N & \text{if} \quad t \to \infty \end{cases}$$

For cycles t such that $\psi(t)$ is fresh, the probability $q_{\psi(t)}(t-1)$ depends on the number $n(t)$ of fresh locations that have already been accessed since t_0. If $n(t) < |\mathcal{B}|$, not all the blocks in $\mathcal{B}$ have yet been read, and the agent selects locations containing $b \in \mathcal{B}$ with probability equal to the efficiency e of the file operation. At t_1 such that $\psi(t_1)$ is fresh and $n(t_1) = |\mathcal{B}|$, the agent has already succeeded in getting all blocks of interest from the store with probability $e^{|\mathcal{B}|}$, and the probability that $\psi(t_1)$ was selected because of containing a block of interest is $q_{\psi(t_1)}(t_1 - 1) = e \cdot (1 - e^{|\mathcal{B}|})$. In general, a location $\psi(t)$ accessed for the first time in cycle t, has probability $q_{\psi(t)}(t-1)$ of the form:

$$q_{\psi(t)}(t-1) = \begin{cases} e & \text{if } n(t) < |\mathcal{B}| \\ e \cdot \sum_{k=0}^{|\mathcal{B}|-1} \binom{n(t)}{k} e^k (1-e)^{n(t)-k} & \text{if } n(t) \geq |\mathcal{B}| \end{cases}$$

Figure 4 (right) shows the evolution in time of $q_{\psi(t)}(t)$ when two operations with $e = 0.5$ are made on a file of $|\mathcal{B}| = 40$ blocks. The first operation on $\mathcal{B}$ starts at time $t_0 = 0$ (finishes at $t_1 = 94$,) and the second starts at $t_2 = 294$ (finishes

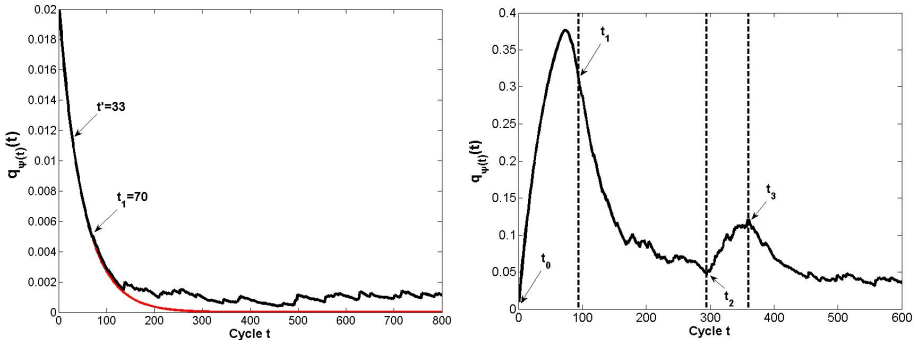

Fig. 4. Function $q_{\psi(t)}(t)$ for a single block accessed at $t_0 = 0$ (left); and for two operations with $e = 0.5$ on a file of size $|\mathcal{B}| = 40$, at $t_0 = 0$ and $t_2 = 294$ (right.)

at $t_3 = 360$,) and the agent performs dummy accesses in cycles $t_1 < t < t_2$ and $t > t_3$. We can clearly see how $q_{\psi(t)}(t)$ grows as the blocks in $\mathcal{B}$ are accessed for the second time in cycles $t_2 \leq t \leq t_3$. The intuition is that there is a correlation between the destinations of $b \in \mathcal{B}$ in the first file operation, and the locations $\psi(t_2) \ldots \psi(t_3)$. And the function $q_{\psi(t)}(t)$ detects this correlation even if the exact locations of the blocks $b \in \mathcal{B}$ are not known at any point.

When the attacker guesses correctly that at t_0 the user operates with efficiency e on a file of $|\mathcal{B}|$ blocks, she assigns high probabilities $q_{\psi(t)}(t)$ to the locations $\psi(t)$ which are likely to contain any of those blocks $b \in \mathcal{B}$. When the file is accessed for the second time starting at t_2, their locations $\psi(t)$ feed back to the pool probabilities $q_{\psi(t)}(t-1) > q_{\psi(t-1)}(t-1)$, such that the function $q_{\psi(t)}(t)$ grows (see (3)) in the time interval corresponding to the second file operation. The correlation becomes stronger when the efficiency e or the file size $|\mathcal{B}|$ increase, and it is most visible when the two file operations are separated by two or three hundred cycles. When the two operations are closer in time, many blocks from

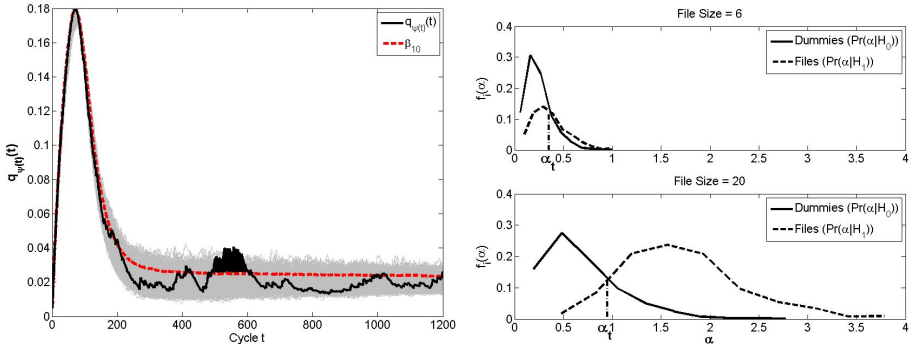

Fig. 5. Area α defined by $q_{\psi(t)}(t)$ going over $\beta_{10}(t)$ (left); and distributions of $f_0(\alpha)$ (dummy traffic) and $f_1(\alpha)$ (file operations) for files of sizes $|\mathcal{B}| = 6$ (upper right,) and $|\mathcal{B}| = 20$ (bottom right.)

the first operation are still in the pool, and can be obtained without accessing the store; and when the two operations are too far apart, the correlation becomes weaker (due to multiple relocations per block) and eventually disappears.

As illustrated by this example, $q_{\psi(t)}(t)$ can be used by the attacker to detect correlations when files are accessed several times, and therefore distinguish between sequences ψ generated by dummy traffic and file operations, as shown in the next section. The computation of $q_{\psi(t)}(t)$ requires knowing (or guessing) t_0, e, and $|\mathcal{B}|$. The efficiency e is a known system parameter, but the adversary would need to try all possible file sizes up to a certain maximum size, and start computing a few new $q_{\psi(t)}(t)$ functions (one per file size) for every cycle t. Our experiments indicate that the required memory and computing power to do so are moderate, and that it would be feasible for the adversary to perform this type of analysis in real-time on a standard PC.

5.2 Example of Test Prior to Coercion

Prior to coercion, the only information available to the attacker is the sequence ψ of accesses to the remote store, and its function $q_{\psi(t)}(t)$. In order to use ψ as evidence of hidden files, the attacker first needs a way to distinguish whether ψ is (most likely) a sequence of dummy accesses (i.e., $\psi \in \Psi_0$,) or it contains file operations (i.e., $\psi \in \Psi_1$.) We assume that the adversary runs her own MixSFS system, and learns the typical shapes of $q_{\psi(t)}(t)$ corresponding to dummy traffic (H_0) and to operations on files of different sizes (H_1.) She uses this information to compute $\Pr(\psi|H_0)$ and $\Pr(\psi|H_1)$ as illustrated by the rest of this section.

The adversary first tests a large number of dummy sequences ψ_0. Given that dummy traffic selects remote locations uniformly at random, its sequence ψ_0 generates functions $q_{\psi_0(t)}(t)$ that fluctuate as white noise around a "baseline." Let $\beta_x(t)$ be a *baseline function* such that in cycle t, $q_{\psi_0(t)}(t) > \beta_x(t)$ only in a percentage x of cases; i.e., $\beta_{100}(t)$ is the lower bound, $\beta_0(t)$ the upper bound, and $\beta_{50}(t)$ represents the median of all the experiments (computed independently for each point t.) As shown in the previous section, file operations generate increases in $q_{\psi(t)}(t)$ that are unlikely to happen at random. In order to exploit this feature to distinguish file and dummy sequences (ψ_1 and ψ_0, respectively,) the adversary compares how much $q_{\psi_0(t)}(t)$ and $q_{\psi_1(t)}(t)$ go over the baseline functions.

Let us denote as α the largest area defined by function $q_{\psi(t)}(t)$ going above a given baseline $\beta_x(t)$, as shown filled in black in Fig 5 (left) (the light background shows many dummy functions $q_{\psi_0(t)}(t)$ superimposed.) Fig. 5 (right) shows the probability density functions $f_1(\alpha)$ and $f_0(\alpha)$, computed for small and large files with a large amount of file (H_1) and dummy (H_0) sequences. As we can see, dummy sequences ψ_0 produce smaller α than sequences ψ_1 containing operations on big files. On the other hand, sequences ψ_1 that contain operations on small files are hard to distinguish from dummy. The adversary constructs Ψ_0 and Ψ_1 using α as a distinguisher as follows:

- Let α_t be the threshold area such that $f_0(\alpha_t) = f_1(\alpha_t)$.
- We say $\psi \in \Psi_0$ if $q_{\psi(t)}(t)$ produces α such that $\alpha < \alpha_t$, and $\psi \in \Psi_1$ if $\alpha > \alpha_t$.

- Dummy traffic (H_0) generates $\psi \in \Psi_0$ with probability $\Pr(\psi \in \Psi_0|H_0) = \int_0^{\alpha_t} f_0(\alpha)$, and $\psi \in \Psi_1$ with probability $\Pr(\psi \in \Psi_1|H_0) = \int_{\alpha_t}^{\infty} f_0(\alpha)$.
- Similarly, file operations (H_1) generate sequences ψ that are unobservable with probability $\mathcal{U} = \Pr(\psi \in \Psi_0|H_1) = \int_0^{\alpha_t} f_1(\alpha)$, and observable with probability $\Pr(\psi \in \Psi_1|H_1) = \int_{\alpha_t}^{\infty} f_1(\alpha)$.

The quality of the distinguisher α depends strongly on the baseline $\beta_x(t)$ chosen by the attacker. To select the optimal $\beta_x(t)$, the adversary computes $f_0(\alpha)$ and $f_1(\alpha)$ for several baseline functions $\beta_x(t)$, $x \in (0, 100)$, and chooses as optimal the one that minimizes the probabilities $\Pr(\psi \in \Psi_0|H_1) = \int_0^{\alpha_t} f_1(\alpha)$, and $\Pr(\psi \in \Psi_1|H_0) = \int_{\alpha_t}^{\infty} f_0(\alpha)$. For our experiments, we have implemented an adaptive algorithm that finds the optimal $\beta_x(t)$ for each file size.

5.3 Example of Test After Coercion

Consider a setting where a fraction $0 < \sigma < 1$ of all locations are occupied by visible blocks (i.e., blocks that belong to files of security level s or lower, as explained in Sect. 2,) and let $\phi(t) = \phi$ (with $0 \leq \phi \leq P - 1$) be the number of visible blocks in the pool at time t. If MixSFS has been performing dummy traffic in the cycles preceding t, then $\Pr(\phi|H_0)$ is given by the probability mass function $f_0(\phi)$ of a binomial distribution with parameters ϕ, $P - 1$, and σ:

$$\Pr(\phi|H_0) = f_0(\phi) = \binom{P-1}{\phi} \sigma^{\phi}(1-\sigma)^{P-1-\phi}$$

Let us assume that the adversary coerces the user at time t_c, and finds $\phi(t_c) = \phi$ visible blocks in the pool. Intuitively, the fraction $\frac{\phi}{P-1}$ of visible blocks in the pool provides the adversary with the following information:

- If $\frac{\phi}{P-1}$ is similar to σ, then it is likely that MixSFS was performing dummy traffic for a period of time before coercion, and no evidence of hidden files is available.
- If it is significantly higher than σ, then evidence ϕ suggests that operations on visible files may have recently taken place. Note that this evidence reinforces the claim of the user that all files are visible and no hidden files exist, meaning that the user has perfect deniability.
- And finally, an abnormally high number of empty blocks in the pool (i.e., proportion of visible blocks much lower than σ,) could be the result of recent operations on hidden files.

For any file size $|\mathcal{B}|$, the adversary can run experiments to determine the distribution of $\Pr(\phi|H_1) = f_1(\phi)$. Figure 6 shows the probability mass functions $f_0(\phi)$ and $f_1(\phi)$ corresponding to dummy and file operations (on files of two sizes,) respectively. We have experimentally computed $f_1(\phi)$ assuming a worst-case scenario in which a hidden file operation finalized at $t_c - 1$, the cycle prior to coercion. As we can see, operations on large hidden files may result in values of ϕ rarely generated by dummy traffic. On the other hand, the ϕ resulting from

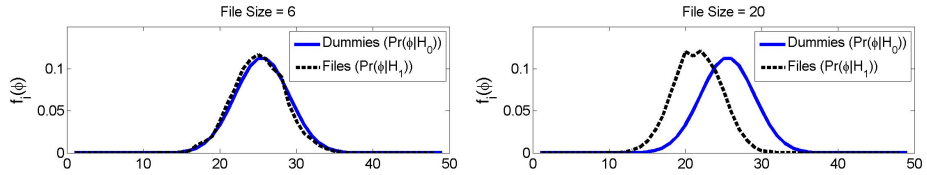

Fig. 6. Probability mass functions $f_0(\phi)$ and $f_1(\phi)$, for $\sigma = 0.5$, and hidden file sizes $|\mathcal{B}| = 6$ (left) and $|\mathcal{B}| = 20$ (right.)

operations on small hidden files follows the same distribution as dummy traffic (to the extent that at coercion, ϕ provides nearly no information on whether small files exist.)

Analogously to the previous section, we can define a threshold ϕ_t such that $f_0(\phi_t) = f_1(\phi_t)$. We say that $\phi \in \Phi_0$ if $\phi > \phi_t$, and $\phi \in \Phi_1$ if $\phi < \phi_t$, and compute the probabilities $\Pr(\phi \in \Phi_0|H_i) = \sum_{\phi=\lceil\phi_t\rceil}^{P-1} f_i(\phi)$ and $\Pr(\phi \in \Phi_1|H_i) = \sum_{\phi=0}^{\lfloor\phi_t\rfloor} f_i(\phi)$, associated to dummy ($i = 0$) and hidden file operations ($i = 1$.)

5.4 Results for Unobservability and Deniability

We have implemented a MixSFS simulator to validate our analysis. This section describes the experimental setup of our implementation, and presents the results we have obtained for unobservability and deniability in the studied configuration. The results are meant to be illustrative and optimizations of MixSFS parameters are out of the scope of this paper.

Experimental Setup. We considered a MixSFS with $N = 951$ remote storage locations and a pool of capacity $P = 50$ (i.e., $N + P - 1 = 1000$ blocks in total.) Files occupy between one and ten blocks, and block redundancy ensures that, for $\sigma = 0.5$, the probability of losing a hidden file is smaller than 10^{-6} even if visible files grow by 10%. This redundancy is proportionally larger for smaller files (one block files are converted to $(n, m) = (6, 1)$, and ten block files to $(n, m) = (20, 10)$.) The efficiency of read and update operations are, respectively, $e_r = 0.75$ and $e_w = 0.25$. In each experiment files are accessed twice, and we have tested the four combinations of read and update operations ('rr', 'rw','wr', and 'ww',) where the two operations are separated by a minimum of 50 and a maximum of 800 cycles.

Unobservability. Figure 7(left) shows the results for unobservability ($\mathcal{U}$) in this setup, depending on file size and type of file operations. We can see that consecutive file read operations are always unobservable for files of size up to $(n, m) = (14, 6)$; and that even for sizes $(n, m) = (20, 10)$, file reads are unobservable 90% of the times. Consecutive file updates however, are observable much more often: over 10% of the times for the smallest files ($(n, m) = (6, 1)$ blocks,) and over 70% of the times for the largest files ($(n, m) = (20, 10)$ blocks.) File reads have higher unobservability because a random subset of m blocks is chosen

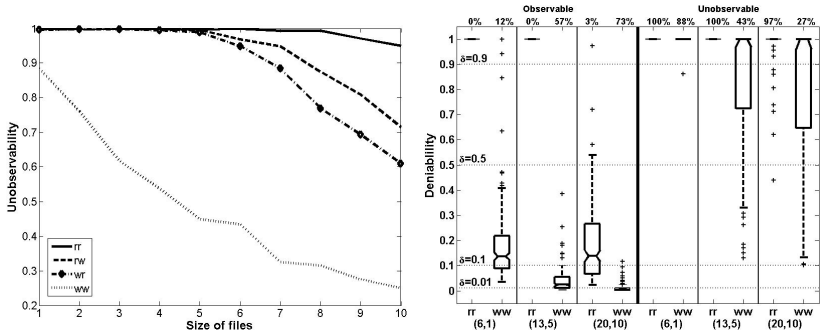

Fig. 7. Results for unobservability (left) and deniability (right)

for each read operation (i.e., erasure codes have the side effect of substantially reducing correlations between file read operations.) For file updates however, the redundancy introduced by erasure codes causes more blocks to be updated, and effectively increases the file size for update operations.

Deniability. We have analyzed deniability ($\mathcal{D}$) in worst-case scenarios, where the adversary coerces the user right after an operation on a hidden file has taken place (whether or not the operation was observable.) We show in Fig. 7(right) our results (boxplots of the distribution of $\mathcal{D}$) for three file sizes and pairs of read and update operations. We have classified the results depending on whether the operations were or not unobservable, and we can see that generally, observable operations result in lower $\mathcal{D}$. This implies that an adversary who can choose to attack the user after observing a file operation has an advantage for finding evidence of hidden files at coercion. We can also see that $\mathcal{D}$ is much higher if hidden files are just read and rarely or never updated: in this case MixSFS offers plausible deniability $\mathcal{PD}$ with threshold $\delta = 1$ for small and medium files, and with $\delta = 0.01$ for big files. If files are meant to be regularly updated however, this configuration of MixSFS can only guarantee plausible deniability for small files and $\delta = 0.01$, even if only a small percentage (12%) of small file update operations have a risk of providing low $\mathcal{D}$.

6 Conclusions

This work studies the security of Steganographic File Systems (SFSs) intended to protect the user against adversaries who monitor accesses to the store. We have presented an architecture of SFS that uses pool mixes to achieve high-entropy block relocation, and prevent known vulnerabilities to traffic analysis attacks [TDDP07] that exploit low-entropy relocation algorithms [ZPT04].

We have defined probabilistic metrics to quantify the unobservability and (plausible) deniability provided by SFSs against coercion attacks. Building on existing mix analysis techniques [DP04], we have presented novel traffic analysis

methods to evaluate the security of SFSs subject to continuous observation. In order to validate our approach we have implemented a MixSFS simulator, examined each step of the attack process, and computed results for unobservability and deniability in a experimental setup. Although we use as example in our analysis a particular type of pool mix, it is trivial to adapt our analysis to other probabilistic relocation mechanisms. The methods introduced here serve as basis for further work on the design and evaluation of traffic analysis resistant SFSs. We note that previous designs have given little or no attention to preventing these types of attacks, in spite of sometimes relying on architectures that use distributed peer-to-peer storage [GL04, HR02], or remote stores observable by third parties, and are thus vulnerable to the adversary and attacks described here.

To better illustrate our contribution, we have considered a very simple dummy access strategy, that chooses blocks uniformly at random amongst all blocks in the store. Our results show that this naive strategy can only conceal accesses to small files. The design of more sophisticated dummy access strategies that offer better unobservability and deniability remains as an open line for further research. Similarly, a fully functional MixSFS implementation would require the specification of additional operations, such as regenerating files after some blocks have been lost or changing user keys after coercion.

Acknowledgements. This work was partially supported by the IWT SBO ADAPID project, the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and by the IAP Programme P6/26 BCRYPT (Belgian Science Policy.) C. Troncoso is funded by a research grant of the Fundación Barrié de la Maza (Spain.) Peter Fairbrother should be credited for the idea of using mixes in the context of SFSs.

References

[ANS98] Anderson, R.J., Needham, R.M., Shamir, A.: The steganographic file system. In: Aucsmith, D. (ed.) IH 1998. LNCS, vol. 1525, pp. 73–82. Springer, Heidelberg (1998)

[AT83] Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. ACM Trans. Comput. Syst. 1(3), 239–248 (1983)

[Cha81] Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 4(2), 84–88 (1981)

[DDM03] Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: Design of a type iii anonymous remailer protocol. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy, pp. 2–15 (2003)

[DP04] Diaz, C., Preneel, B.: Reasoning about the anonymity provided by pool mixes that generate dummy traffic. In: Fridrich, J. (ed.) IH 2004. LNCS, vol. 3200, pp. 309–325. Springer, Heidelberg (2004)

[GL04] Giefer, C., Letchner, J.: Mojitos: A distributed steganographic file system. Technical Report, University of Washington (2004)

[HR02] Hand, S., Roscoe, T.: Mnemosyne: Peer-to-peer steganographic storage. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 130–140. Springer, Heidelberg (2002)

[MCPS03] Ulf Moller, Lance Cottrel, Peter Palfrader, and Len Sassaman. Mixmaster protocol - version 2 (2003), http://www.abditum.com/mixmaster-spec.txt

[MK99] McDonald, A.D., Kuhn, M.G.: Stegfs: A steganographic file system for linux. In: Pfitzmann, A. (ed.) IH 1999. LNCS, vol. 1768, pp. 462–477. Springer, Heidelberg (2000)

[PH01] Pfitzmann, A., Hansen, M.: Anonymity, Unobservability and Pseudonymity – A Proposal for Terminology. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 1–9. Springer, Heidelberg (2001)

[Rab89] Rabin, M.: Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of ACM 36(2), 335–348 (1989)

[Riz97] Rizzo, L.: Effective erasure codes for reliable computer communication protocols. Computer Communication Review 27(2), 24–36 (1997)

[TDDP07] Troncoso, C., Diaz, C., Dunkelman, O., Preneel, B.: Traffic analysis attacks on a continuously-observable steganographic file system. In: Furon, T., et al. (eds.) IH 2007. LNCS, vol. 4567, pp. 220–236. Springer, Heidelberg (2008)

[ZPT04] Zhou, X., Pang, H., Tan, K.-L.: Hiding data accesses in steganographic file system. In: Proceedings of the 20th International Conference on Data Engineering, pp. 572–583. IEEE Computer Society, Los Alamitos (2004)