# A Web based Covert File System

Arati Baliga, Joe Kilian and Liviu Iftode
*Department of Computer Science*
*Rutgers University, Piscataway, NJ.*
{*aratib, jkilian, iftode*}*@cs.rutgers.edu*

## Abstract

We present the idea of a web based covert file system, CovertFS. This file system allows a user to store files covertly on media sharing websites while guaranteeing confidentiality and plausible deniability regarding the existence of the files. Further, it allows for selective and covert sharing of these files with other users. CovertFS can be built on top of any web based media sharing service. The files are hidden within the media using steganographic techniques. The user can plausibly deny the existence of the covert file system since the existence of it cannot be proven. The media sharing service provider is oblivious to the existence of the file system within the stored media, providing them plausible deniability as well. Since the user files are completely hidden, it gives only the user complete control over his confidential files.

## 1 Introduction

Web services such as email, photo sharing, video sharing, blogs, wikis and other collaborative and interactive services have become a part of our daily lives. The web provides an easy and portable means for storing and retrieving user content as well as sharing this content within a group of people for online collaboration.

All web services available today are for open storage and sharing, where the existence of the data is known to the service provider. The fundamental, implicit assumption here, is that the service provider can be completely trusted with the user data. Any content stored in the clear on these servers is vulnerable to unauthorized access by the service administrators. Further, the government could compel the service provider to turn over this data without the knowledge of the user. A more cautious user might encrypt all content that is stored on these servers. While this protects the data from unauthorized access, it cannot hide the fact that some data is stored by a particular user. The user might be subsequently coerced into revealing the encryption keys by legal instruments such as subpoenas. Thus, users may desire to hide the very presence of their data stored on public servers in such a way that its existence cannot be proven by the service providers themselves or another third party.

Storing and sharing data covertly over the internet serves several purposes. For example, this may be used as a means to share content in societies that tend to stifle free exchange of unpopular ideas. Even in more democratic countries, social taboos can force people to look for covert means for facilitating secret online collaborations. Finally, individual web users may use such covert means to backup, store and share their files online without the knowledge of the service providers.

In this paper, we propose the idea of a covert web based file system, CovertFS, which facilitates secure file storage and sharing amongst a group of people and yet provides plausible deniability. CovertFS can be built on top of any publicly available media hosting and sharing service. Flickr [2], a photo sharing service from Yahoo, is an excellent example of such a service as it provides large storage and excellent API. The file system is covertly hidden within the media hosted by the user using steganographic techniques. This file system provides plausible deniability for the user and the service provider. Plausible deniability is achieved because the presence of the hidden data cannot be determined by any external parties, including the service provider.

The salient features of the file system can be stated as follows:

- **Plausible Deniability:** The presence of the file system or files within the media sharing web account cannot be determined with certainty by analyzing the media or the traffic. Hence the user or the service provider cannot be compelled by court to disclose the contents of the file system. This form of information hiding is desired by users who can safely and securely store their documents on third party servers without the knowledge of the service providers. The service provider cannot determine with certainty whether the media is a plain media file or a media file with hidden content.

- **Online File Sharing and Collaboration:** The file system is built on top of a web based media sharing service. This makes the file system available online and to anyone anywhere to collaborate or share files with one another. The additional benefit of this is only the end-users are aware of the file system organization and contents. To others, the file sharing traffic looks like innocuous media sharing traffic.

- **Information Hiding:** The file system is aimed at hiding confidential documents, which can be stored and shared between a group of people. Data is hidden within the media using advanced steganographic techniques such as secure or zero divergence steganography [13, 12, 11], which cannot be steganalyzed to retrieve the hidden content.

## 2 Design Overview

The design of a covert file system on a media sharing website poses several research challenges. First, an efficient way to hide the file system data within photos is necessary. Advances in steganography help us here [13, 12, 11]. The file system data can be encrypted and hidden within the photos in such a way that an adversary cannot detect the difference between regular photos and photos with hidden data. Secondly, we need an efficient mapping scheme of the file system blocks to images in order to fully utilize the storage capacity offered by the public server. Finally, covert file access traffic should not be distinguishable from the innocuous photo sharing traffic on the same website, originating from the ordinary users. These users are likely to download new photos and ignore photos they have already seen, they seldom update photos they have already posted and do not delete old photos until there is a shortage of storage on their account. Such access patterns should not be violated when the media website is used to access the hidden file data. In what follows, we will discuss the key design issues that can address these challenges.

### 2.1 Mounting the File System

In CovertFS, files are stored remotely, hidden within the media hosted on a third party service provider. To access the hidden file system, a user mounts it at a desired mount point in the local file system. Before mounting the file system, the user should have a valid account on a media sharing site. During the mount, the user has to present proper authorization details such as the media sharing website url, account name and password for the account where his file system is hosted and the passphrase for encryption/decryption of the file system contents. After verifying the authorization information, the file system mounts the remote web based file system and begins downloading photos as dictated by the hidden file system accesses. To avoid repeated downloads of certain photos (unusual access pattern for media sharing), photos containing the hidden file system metadata are kept in a local image cache as long as possible.

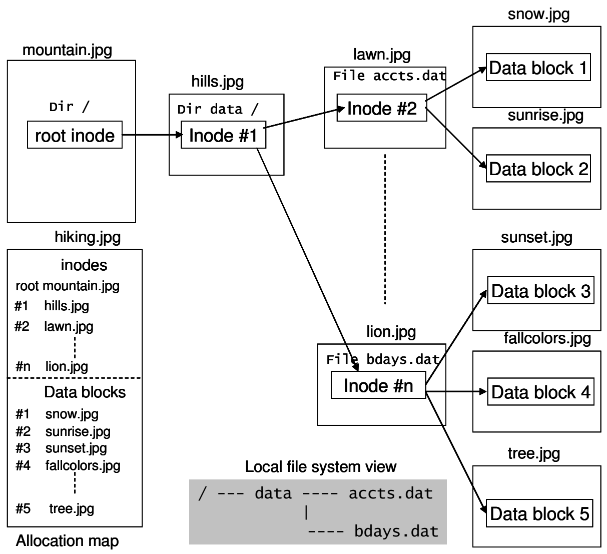
### 2.2 Mapping File System Data to Photos

The entire file system information along with file meta data and file contents are encrypted and stored within media content on the service provider. Current steganographic techniques are used to hide the contents of the file within the media. Our media content here are photos to be shared with friends and family. Typical photo sizes stored on Flickr range anywhere between 40KB to 300KB. Current steganographic techniques can safely allow embedding of about 10% of information within a JPEG image with no visual distortion or deviation in statistical properties of the image. Considering a minimum image size of 40 KB, a 4 KB disk block size can be stored. To keep the mapping simple, we can assume that disk blocks are mapped to images one-to-one, which means that we will only hide 4 KB of data even in larger images.

To store all the files within the file system remotely, we need as many images in the Flickr account as the number of blocks on the file system. The number of files that can be stored is constrained by the account storage size within Flickr. However, Flickr and most service providers have unlimited accounts for a minimal service fee per year, providing a virtually unlimited storage capacity. Alternative designs can store mutiple file block in larger images or can span over multiple user accounts and/or multiple service providers.

Metadata, such as inode blocks, and the direct and indirect disk blocks are also stored in photos. Inodes and file block addresses can be identified directly by the name of the image where they are stored or indirectly using inode and block allocation maps, themselves stored in one or multiple images. Retrieval of the photo containing the first block of the map is done through a name that, when hashed, maps to a special value, usually a function of the encryption passphrase entered by the user.

Fig. 1 shows the file system object hierarchy as embedded within different photos stored in the Flickr account. The photo *mountain.jpg* contains the root inode, which points to the only directory inode under the root directory embedded in the photo *hills.jpg*. The direc-



```
/ --- data ---- accts.dat
               |
               ---- bdays.dat
```

Figure 1: Covert file system layout embedded within Flickr photos and a local view of files when file system is mounted

tory contains two files, whose inodes are embedded in photos *lawn.jpg* and *lion.jpg* respectively. Data blocks are contained within photos *snow.jpg*, *sunrise.jpg*, *sunset.jpg*, *fallcolors.jpg* and *tree.jpg*. The allocation map for inodes and data blocks are stored within the photo *hiking.jpg*. The figure also shows the local view of the file system, within the gray box.

## 2.3 Handling File System Writes

In a read-write file system, metadata as well as data blocks change as a result of file accesses. In CovertFS, these changes may generate operations that may look suspicious for genuine photo sharing such as (i) frequent image changing and (ii) frequent access to certain old images. In the next two subsections, we will discuss mapping solutions to hide these two file system patterns.

To address frequent image changing due to inode and file system block updates, we propose to make photos immutable and apply an update scheme similar to one used in the log structured file system [10]. According to this scheme, modified file system objects will be hidden in new photos. To achieve this, the indirection through the allocation map is absolutely necessary.

With the proposed scheme, the allocation map becomes the file system object whose frequent changes must also be hidden. To keep the photos carrying the allocation map also immutable, we must devise a mechanism to locate the most recent copy of the map. For this, we propose two complementary schemes. The basic scheme takes advantage of the user-defined name space for photos to apriori decide the name of the photo to store the next version of the map and to embed it along with the version number in the photo of the current map (forward pointer). In this way, a file system user can easily determine when the allocation map has changed by looking at the photo name of the next map. If the new photo does not exist but the old one does, the client can assume that the map has not changed (photos in the same chain are garbage collected in the FIFO order) and use its cached copy. As a backup, in case this chain cannot be reconstructed due to garbage photo collection, the names of the map photos are chosen such that all map to the same special value when when hashed with the user passphrase. In this way, in the worst case, a complete inspection of all the images in the account, will allow a user to discover the most recent copy of the map.

Photo garbage collection is done when the user account reaches near full capacity. The photos containing the invalidated blocks will all be deleted in a batch during this process, freeing up space in the account, yet generating traffic patterns of photo sharing users.

## 2.4 Avoiding Photo Hotspots

The current design may expose suspicious hotspot patterns as metadata photos are likely to be more frequently accessed, which can be an indicative for a covert file system. Local caching can alleviate this behavior but only partially. To further diffuse this pattern, we plan to introduce forward pointers to all metadata objects and not just the maps. This means that subsequent copies of an inode, for instance, will be chained by embedding the name of the photo to store the next version of the inode in the one carrying the current one. A user who wants to retrieve the most recent version of an inode and has a cached photo of an potentially old version can follow this chain to retrieve it without referring to the allocation map every time. To guarantee that the file corresponding to that inode was not deleted, the most recent copy of the parent directory must also be checked. Finally, avoiding hotspots through this mechanism is an optimization. In case an inode version chain cannot be reconstructed, the user can go back to retrieve the most recent version of the inode starting from the allocation map.

## 2.5 File Sharing and Access Control

Flickr provides three types of sharing. Photos can either be made private, shared with a group, or made public. Private photos are only accessible to the user who created them. If photos are shared as a group, friends and family can access them and of course, photos made public can be accessed by anybody. However, group and public access sharing do not allow the user to modify the files.

We build our file sharing and access control model on top of the Flickr photo sharing model. Only the owner of the Flickr account is able to modify file system content, while members of the group or others can only read files or part of the file system that is enabled selectively for read sharing by the owner.

Selective sharing needs to be enabled by the owner who wants to share his files or directories with other users in the group. Each share is assigned a separate encryption passphrase as shown in Fig. 2. The directory Politics is shared with a group of friends with a separate encryption key. Every parent inode object that has a link pointing to a file or directory has a respective encryption key associated with it. Storing the encryption key in the inode allows the owner to access all the files at any time without retyping separate encryption passwords assigned to different shares. In case a separate encryption key is not assigned to any file or directory, the encryption key is replicated from the parent inode. All other directories in Fig. 2 are encrypted with the owners encryption key.

The photos corresponding to the directory to be shared (Politics directory in the fig) are moved to the appropriate category of photos in the Flickr account for sharing with the group. The encryption passphrase for files within the share is given to other users of the group. They can locate the root inode within the share by hashing with the given passphrase. Note that the passphrase is different for each share and can be changed by the owner at any given time, when he decides to revoke sharing.

## 2.6 Replication

Since the web based services can be unavailable at certain time periods, replicating the file system meta-data and data across different service providers is a desirable design choice. The replicas can be assigned priorities such that the downloads always take place from the primary replica. When the primary replica service provider is unavailable, the files can still be accessed from the secondary replicas. Updates may however be propagated first to the primary and then to the secondary replicas.

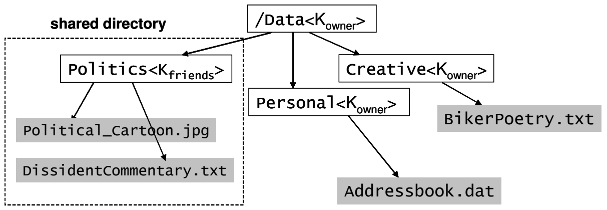


Figure 2: File sharing with CovertFS. The directory *Politics* is shared and has its own encryption key

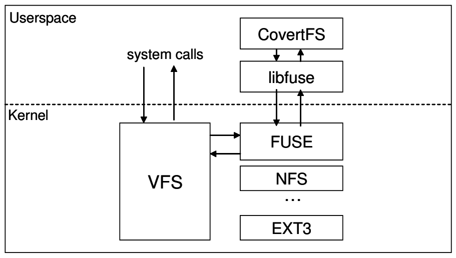


Figure 3: Overview of CovertFS design

## 2.7 CovertFS and Hidden Levels

It is possible to argue that the presence of CovertFS itself can be incriminating evidence that the user is hiding something. This is countered in the steganographic file systems by having different hidden levels. When compelled by court, the user can disclose only one or two levels with moderately incriminating evidence, while the presence of real data that he wants to hide can be plausibly denied. It is impossible with this design for the examiner to prove that the user is indeed hiding something extra than what he has already disclosed.

A similar analogy applies to CovertFS as well. Hidden levels can be created in CovertFS. Each level has a different encryption passphrase and can only be opened when the user provides the correct passphrase. Additional levels are also mapped by using other photos within the same account. Alternatively, hidden levels may also be created involving additional user accounts on the same or other service providers.

## 2.8 Implementation Plan

CovertFS can be built as a user-level file system. We plan to implement this on top of the FUSE [1] file system interface as shown in Fig. 3. FUSE facilitates easy development of user-level file systems. It has a kernel mode driver and a user-level library. The user-level library *libfuse* interacts with the kernel mode driver through a device called */dev/fuse*. The system calls that operate on files in the FUSE file system are redirected from the virtual file system (VFS) layer in the kernel to the FUSE driver. The driver in turn forwards this call to the user-space library. The new filesystem, CovertFS in the figure, that links into this library can handle this call and implement new functionality. We plan to develop our proof-of-concept prototype over Flickr [2], since they have a public API available for this purpose.

## 3 Discussion

In this section, we provide security analysis and discuss other design related issues.

## 3.1 Security Analysis

We define two types of adversaries. A passive adversary simply observes the traffic and checks for anomalies. An active adversary, on the other hand, actively performs steganalysis on random images from time to time to detect hidden data within the images. We examine why CovertFS is indeed covert from the point of view of both the active and the passive adversary. The active adversary is primarily concerned with steganalysis, while the passive adversary mainly performs traffic analysis on the Flickr account traffic.

### 3.1.1 Active Adversary and Steganalysis

Steganalysis is a technique where the adversary can determine that the image is used as a cover for hidden information. Steganalysis techniques watch for signature distortions created by known steganography tools. With respect to JPEG images, some steganography tools use simple techniques such as LSB encoding. In this method each bit of the hidden text is encoded in the least significant bit of every byte in the JPEG image. This technique does not cause visual distortion perceptive to the human eye, but creates huge deviations in the statistical properties of the JPEG image. Such tools that perform bit manipulations are called image domain tools and can be detected easily by steganalysis.

Other set of techniques used for hiding information is called the transform domain tools. These group of tools use techniques that involve manipulation of algorithms and image transforms. One of the popular techniques used for JPEG images is called the discrete cosine transform (DCT). These methods hide information in more significant areas of the image and may manipulate image properties such as luminance. These techniques are far more robust and much harder to detect using steganalysis. The tradeoff however is that such methods can encode much smaller amount of information within the cover. Recent research has come up with zero divergence steganography or secure steganography that use statistical restoration techniques [13, 12, 11]. The basic idea is to thwart steganalysis methods by hiding information in few bits within the image and adjusting other bits to offset the deviations caused by the hidden information. Hence, advances in steganography have made it possible to build tools that can thwart steganalysis. We plan to use one such advanced technique in our prototype to hide the file system data.

### 3.1.2 Passive Adversary and Traffic Analysis

The passive adversary simply sniffs traffic to look for anomalies and tries to deduce if any hidden text exists within the image. All the traffic during upload and download of the files within the file system must appear like innocuous photo sharing traffic. However, the pattern in which the files are accessed may leak some information to the adversary. The adversary however, must not be able to determine with certainty that a specific pattern at the beginning of accesses, implies hidden text.

Traffic patterns can be obfuscated by introducing pseudo random dummy image fetches. The client can cache already visited photos to ensure that it does not download those photos too frequently. CovertFS is designed such that only new photos are uploaded and old ones are deleted when the account reaches near full capacity, which resembles the behavior of normal photo sharing users. Also, the additions are done in a batch as the file system operates in a disconnected mode, making additions in a batch to the photo store, similar to how regular users add photos. Since Flickr has an open API, several other applications have been built on top of it that perform specific tasks, customized to the user. Each of these tasks gives rise to different upload/download patterns.

## 3.2 Feature or Misuse

CovertFS can be built on top of media sharing service such as Flickr. While this provides an innovative use of a commonly used web service, this can be viewed as abuse of a service designed for a different purpose. We argue that since Flickr is a photo sharing service, what else is embedded in the photos does not really affect Flickr's business model. Users still host photos on Flickr for CovertFS to work.

## 4 Related Work

The steganographic file system that gives the user plausible deniability was first proposed by Anderson and Shamir [6]. They did not have a working prototype of the file system. McDonald et al [8] were the first to build a working prototype of a steganographic file system called StegFS. StegFS is a local file system that provides plausible deniability by hiding files in unused disk blocks. The prototype did not require a separate partition but worked along with the Linux ext2 partition. Pang et al [9] demonstrated improvements to the hiding schemes and design of StegFS, which demonstrated significant improvements in performance. All the file systems mentioned above work with the local hard drive and provide plausible deniability to the user. None of these provide the ability to globally access or share files. Since all of these hide in unused disk blocks, they run the risk of being overwritten when the driver is not operating in the steganographic mode. Therefore these require a high degree of replication, severely limiting the disk space usage. CovertFS, on the other hand, provides file sharing between geographically distant users as well as plausible deniability. CovertFS hides files from the service providers themselves and is built over a media sharing service. The design considerations are significantly different in both cases.

The gmail file system [3] allows the user to store his data as email messages in his mail account. The service provider is aware of the existence of the user files in this mail account. This file system does not allow plausible deniability or enable file sharing with others. Httpfs [7] is a network file system that provides access to files on a remote machine using the http protocol. It requires a component to run on the remote server, from where documents can be fetched on the client. This is similar to the network file system implementation but using http. For CovertFS, no such component is required on the server side. DavFS [4] allows to mount files from a WebDAV server on a local driver. WebDAV is an extension of http that allows remote collaborative authoring of web resources. DavFS allows a remote web server to be edited simultaneously by a group using standard applications. DavFS, fundamentally differs from our implementation as it requires a server component. None of the above file systems provide plausible deniability either. CovertFS can run on top of any media hosting service. The control lies with the user on how he accesses/modifies his hidden files.

The Web File system [5] provides a file system interface to the world wide web. The goal here is completely different from our goal. This file system allows the user to browse the web as different files that are downloaded on the local hard drive.

## 5 Conclusion

In this paper, we motivate the need for a web based covert file system, CovertFS. This file system allows users to store their files, hidden inside the media hosted on a public server and access them from anywhere in the world with complete confidentiality from any third party including the service provider. Additionally, the very existence of the file system is known only to the user and cannot be determined or proven by anyone else. Further, it allows files to be selectively and covertly shared with others as and when needed. As part of ongoing work, we are developing a working prototype of such a file system and will evaluate it in terms of latencies, scalability, security and privacy.

## Acknowledgments

We would like to thank the anonymous reviewers and Pandurang Kamat for their insightful comments and feedback. This work has been supported in part by the NSF CAREER grant CCR-0133366.

## References

[1] Filesystem in userspace. http://fuse.sourceforge.net/.

[2] Flickr photo sharing. http://www.flickr.com/.

[3] The gmail file system. http://richard.jones.name/google-hacks/gmail-filesystem/gmail-filesystem.html.

[4] Webdav linux file system (davfs2). http://dav.sourceforge.net/.

[5] ADYA, A. Web file system: File-like access to the web. In *5th Annual MIT Student Workshop on Scalable Computing. Wellesley, MA. August 1995*.

[6] ANDERSON, NEEDHAM, AND SHAMIR. The steganographic file system. In *IWIH: International Workshop on Information Hiding* (1998).

[7] KISELYOV, O. A network file system over HTTP: Remote access and modification of files and files. pp. 75–80.

[8] MCDONALD, A. D., AND KUHN, M. G. Stegfs: A steganographic file system for linux. In *Information Hiding* (1999), pp. 462–477.

[9] PANG, H., TAN, K.-L., AND ZHOU, X. Stegfs: A steganographic file system. *icde 00* (2003), 657.

[10] ROSENBLUM, M., AND OUSTERHOUT, J. K. The design and implementation of a log-structured file system. In *SOSP '91: Proceedings of the thirteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1991), ACM Press, pp. 1–15.

[11] SOLANKI, K., SULLIVAN, K., MADHOW, U., MANJUNATH, B. S., AND CHANDRASEKARAN, S. Statistical restoration for robust and secure steganography. In *IEEE International Conference on Image Processing* (Sep 2005).

[12] SOLANKI, K., SULLIVAN, K., MADHOW, U., MANJUNATH, B. S., AND CHANDRASEKARAN, S. Provably secure steganography: Achieving zero k-l divergence using statistical restoration. In *IEEE International Conference on Image Processing 2006 (ICIP06)* (Oct 2006).

[13] SULLIVAN, K., SOLANKI, K., MANJUNATH, B., MADHOW, U., AND CHANDRASEKARAN, S. Determining achievable rates for secure zero divergence steganography. In *IEEE International Conference on Image Processing 2006 (ICIP06)* (Oct 2006).