CryptoCache: A Secure Sharable File Cache for Roaming Users

Christian D. Jensen
Department of Computer Science
Trinity College Dublin
Christian.Jensen@cs.tcd.ie

Abstract

Small mobile computers are now sufficiently powerful to run many applications, but storage capacity remains limited so working files cannot be cached or stored locally. Even if files can be stored locally, the mobile device is not powerful enough to act as server in collaborations with other users. Conventional distributed file systems cache everything locally or not at all; there is no possibility to cache files on nearby nodes.

In this paper we present the design of a secure cache system called CryptoCache that allows roaming users to cache files on untrusted file hosting servers. The system allows flexible sharing of cached files among unauthenticated users, i.e. unlike most distributed file systems CryptoCache does not require a global authentication framework.

Files are encrypted when they are transferred over the network and while stored on untrusted servers. The system uses public key cryptography, which allows roaming users to selectively grant read and write access to others by entrusting them with respectively the public key or the private key.

1 Introduction

Mobile computers, laptops, palmtops, personal digital assistants (PDAs) and smart cards are rapidly becoming small enough to carry around and at the same time powerful enough to be useful for roaming users. Unfortunately, the storage capacity of these devices is small compared to fixed installations, so mobile users generally rely on connectivity with their home network for storage of larger amounts of data, e.g. by using a distributed file system like AFS [8], Coda [3] or Echo [1]. This dependence on a remote system introduces reliability problems (e.g. network unreachable), quality of service problems (e.g. jitter in audio or video) and security problems (e.g. eavesdropping) that have to be solved.

The problems of reliability and quality of service are greatly reduced by caching or prefetching frequently used files to a server close to the user. However, unless the user's organization owns a global network of file servers, cached files have to be temporarily stored by renting disk space on an untrusted file server. Renting temporary disk storage from local file hosting services (current examples of such services are HostFile (`www.hostfile.com`) and X:drive (`www.xdrive.com`)) amplifies the security problems of remote access. In addition to interfering with the files in transit on the public network, a successful attacker could manipulate the data stored in the cache. Cached files must therefore be stored in the same way they are transferred securely over the network, i.e. confidentiality, integrity and authenticity can be assured by encrypting and signing each file on the user's home system, transmitting encrypted files across the network and caching encrypted files on the untrusted file hosting server. The files are only decrypted when they are used on the mobile user's mobile computer.

While this scheme offers secure and reliable delivery of data to the mobile user's platform, it does not allow him to grant local users read-only access to cached data. The only way to share a file with a local user is to give him the encryption/decryption keys. Sharing keys raises a number of issues that are not adequately addressed by existing cryptographic file systems [2, 4, 9]. In this paper, we identify some of these shortcomings and present the design of a sharable cryptographic file cache for roaming users.

The rest of this paper is organized as follows: Section 2 presents our threat model, identifies the requirements for a cryptographic cache and analyses the applicability of existing cryptographic file systems. Our design of the sharable cryptographic file cache is presented in Section 3. Finally, we present a discussion of our system in Section 4.

2 Secure Caching

2.1 Motivation

Computing is becoming increasingly mobile and at the same time mobile computing is now entering the field of consumer electronics with wireless access from PDA's and keyboards that can be attached to mobile phones for writing SMS messages and browsing the web, e.g. the Ericsson Chatboard. Mobile users will come to expect ever more from smaller and lighter devices. However, these devices will have limited computing and storage resources, so they need to be strongly integrated with the user's home system enabling him to avail of all of his resources while on the move.

In this paper we focus on the storage requirements of roaming users. Current handheld devices can typically hold a few megabytes of data. The storage capacity of such devices is unlikely to change dramatically with current technology, because battery power and lifetime are limiting factors.

Roaming users will therefore either require fast and reliable connections to their home system or the ability to cache data close to their current location. Price may well be a deciding factor in deciding between these alternatives. In Europe, mobile telephone companies charge little for a local call, but international calls are very expensive. It is therefore much cheaper for the mobile user to access a cache than his home system, so caching allows the development of a larger variety of applications.

2.2 Application Scenarios

In the following we consider a few application scenarios, that further motivate mobile users' need for a secure file cache.

Consider a traveling salesman who is demonstrating a product to a client. Part of this demonstration consists of projecting a video and giving the client access to online manuals or training videos on his own PDA. It is vital that the projection runs smoothly and that the online access is fast and reliable. This information may not be highly confidential, but it should not be freely available to people who have not bought the product, and early disclosure to competitors may deprive the salesman's organisation of its leading edge. Prefetching such substantial amounts of data and caching them close to their place of use fulfills the quality of service requirements, while signatures and encryption assures confidentiality, integrity and authentication of the downloaded data.

Another example is two researchers who meet during a conference and decide to write a paper or research proposal together. The paper/proposal is partly based on a large dataset of observations that researcher X made at his home laboratory. In this scenario both should be given read-write access to the file containing the paper, but only researcher X should be allowed write access to the dataset.

A final example is a person who brings his MP3 player with him on a holiday abroad. The capacity of current MP3 players is fairly low (e.g. the Diamond Rio has 64MB, which translates into 16–20 titles. The selection of music stored in the MP3 player is easily changed if a selection of MP3 files has been prefetched to a nearby server. This server provides the traveller with fast and reliable access to a large selection of his favorite music, without incurring the extra costs and power consumption of a hard disk drive in the MP3 player.

These examples illustrate some applications of a secure file caching system. Moreover, the diversity of applications illustrated above indicates that this system may have many other uses, e.g. multiplayer game consoles, portable MPEG players, pay per view TV, etc.

2.3 Threat Model

Files transferred between the home network and the mobile device are subject to eavesdropping, deletion, modification, and fabrication on the public network and on the cache system.

The roaming user should therefore not be forced to trust users or administrators of either the public network or the cache server. In fact, the security of his mobile computation should depend only on the security of his mobile device and home system.

2.4 Secure Cache Requirements

The following requirements should be met by the secure cache system.

1. All data transfers must be signed to prove authenticity.
2. All data must be encrypted when they are transferred and stored outside the home network to prevent unauthorised access to the data.
3. Roaming users must be able to share files in the cache with people they meet and trust.
4. Sharing files must be simple and flexible, allowing the roaming user to share files with little or no intervention from the system administrator of his home system, and to share some files read-only and other files read-write.

Most file systems protect data by limiting user access to stored files. This implies the ability to establish an authenticated identity of the requesting user and compare that to the access control information for that file. Access is denied if the requesting user is not known to the file system, e.g. accesses from users that the roaming user encounter on his way. The system will deny these users access unless the system administrator of the home system injects information about the user into the file system, e.g. by creating an account for that user. This introduces a delaying factor in the collaboration and may destroy the spontaneity. It is therefore important, in this context, to separate access control decisions from the authenticated identity of the requesting user.

2.5 Related Work

Cryptographic file systems are not a new idea. Systems like CFS [2], TCFS [4] and Cryptfs [9] allow encrypted storage of files on disks. In these systems, files are transparently encrypted/decrypted using a key stored in the memory of the process operating on the file. These systems use symmetric cryptography which means that encryption and decryption use the same key. This means that the separation of read and write access must be enforced by the access control mechanisms of the underlying file system. Neither of these systems provide the flexibility of sharing required by roaming users.

Truffles [7] is a distributed file system that attempts to make file sharing between users in different administrative domains both simple and secure. It uses the Ficus file system [6] to offer replication and sharing of files and Privacy Enhanced Mail (PEM) as a secure transport mechanism. Truffles allows users to share volumes (subsets of the entire file system) with little intervention from the system administrators once the volumes have been defined. Ficus replicates volumes using an optimistic one-copy policy and all systems sharing the volume may hold its own replica. However, Truffles does not allow users to prefetch or push data to conveniently located servers. Furthermore, file sharing relationships and file data transfer rely on Privacy Enhanced Mail (PEM) which requires a nonempty intersection among the public key infrastructures that each user belongs to; this may hinder collaboration among users.

None of the cryptographic file systems allow files to be selectively shared read, write or read-write with users outside the administrative domain of the server. Truffles allow easy sharing once the volumes are defined, but it does not allow file caching near the collaboration. It is therefore better suited for long term relationships than short collaborations among roaming users.

3 CryptoCache Design

3.1 Overview

The CryptoCache is implemented as a special network file system in the same way as the cryptographic file systems mentioned above. For the

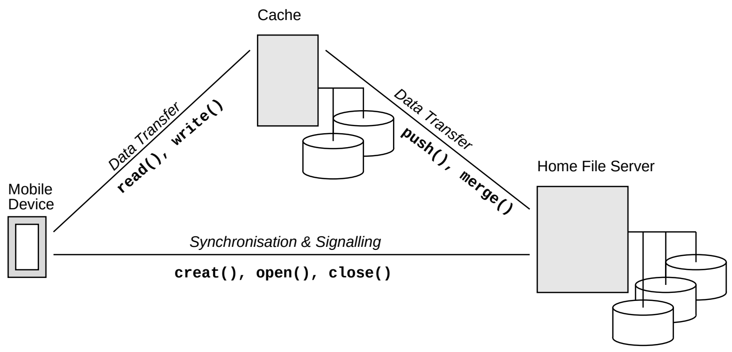

Figure 1: Overview of the CryptoCache System

sake of simplicity our design assumes that the user's home directory is mounted under `/cache` on a Unix-like system, but it could also be available as a folder on a network drive in a Windows or Mac environment. An overview of the system is shown on Figure 1.

The first time a user tries to access a file under `/cache`, the CryptoCache system determines the current location of the mobile device. It needs to be location-aware so that it can avoid caching files when it is connected directly to the home network and use already allocated disk space on a disk hosting server if it has not recently moved.

Disk space is allocated on a new file hosting server when the mobile device moves to a new location.[1] However, before allocating new disk space, the CryptoCache client ensures that the server merges the latest updates from the current cache with the files stored on the home system.

The CryptoCache client redirects vnode operations to a CryptoCache server on its home network (starting one if it is not already running). Thus, the server runs with the privileges of the roaming user.

Cryptographic file encryption keys are stored on a single key ring, with pointers from cached files to the corresponding key pairs. The mobile device is assumed to have sufficient capacity to store these keys permanently, e.g. 256 RSA 768 bit key pairs require about 45KB of memory. The device would also benefit from specialised hardware for encryption/decryption of data sent over the network.

Communication between mobile device and the home server must be secure, e.g. using SSL [5], but all communication with the CryptoCache server may use whatever protocol supported by that server.

3.2 File Cache Operations

The file cache is accessed in the same way as the cryptographic file systems and most of the file system operations are the same (except for the fact that CryptoCache uses public-key cryptography). The following operations differ from those of the cryptographic file systems.

`creat()` The creation of a new file is done as follows: First the mobile device calculates a key pair to be used for this file. It then encrypts the filename and sends a message to the home server containing the encrypted filename and the key pair for that file so that the CryptoCache server can create a copy on the home system. At the same time the CryptoCache client creates a new file on the cache server with the encrypted filename.

`open()` The open operation first checks a list of already cached files. If the file is cached, it opens the file from the CryptoCache server, otherwise it contacts the home system and requests a copy of the file. This request contains the location of the current cache server. The home system starts to copy the file to the CryptoCache server and replies with the key pair and the length of the file. Once this reply is received, the client can start to read from the CryptoCache server.

`read()` Each read operation is forwarded to the CryptoCache server. Each block, when received by the client, is decrypted using the decryption

[1] Disk space allocation may require a special trading service or human intervention.

key associated with the file and the block's signature is verified to ensure the integrity of the stored data.

`write()` The write operation encrypts the block, signs it and writes it to the cache server. Each block is encrypted and signed independently so that several users can update the file at the same time. This preserves the traditional "last write wins" semantics of NFS.

`close()` When a cached file is closed on the mobile device, the file is first closed on the cache server and a message is sent to the home system. If the client cannot reach the home system, the filename is written to a list of files, which is sent to the home system the next time that it becomes available.

3.3 File Server/Cache Interaction

The home server uses standard file system operations to push requested files to the cache and merge modifications from the cache with its own files. These logical operations are outlined in the following:

`push()` The first time a mobile user requests a file, this file is pushed to a CryptoCache server close to the client. The home system generates a key pair and uses the encryption key to encrypt and sign each disk block as it is written to the CryptoCache server. It then sends a reply to the client with the key pair and the length of the file. Knowing the length of the file allows the mobile device to start reading from the cache server before the home system has transferred the entire file.

`merge()` A message is sent to the home server every time a cached file is closed on the mobile device. This allows the home system to merge modifications from the cache with the original copy.

3.4 Sharing Files

When a roaming user wishes to give another user read access to one of his shared files, he gives the decryption key to that user along with any information needed to access the file on the cache server (typically a password, but different file hosting services may implement different access control mechanisms). The access control information allows him to access the file and the decryption key allows him to decrypt the contents of the file. Similarly, write access can be granted by giving the encryption key to the other user.

Filenames and possible passwords for the file hosting servers can be transferred in clear text, but the encryption/decryption key has to be sent over a secure channel, e.g. sending an email using PEM.

3.5 Security of the Design

The cryptographic keys used in the CryptoCache are similar to capabilities in the sense that the decryption key allows the principal to read the file (read capability) and the encryption key allows the principal to write and sign the file (write capability). However, they are fundamentally different because capabilities form part of an access control mechanism and no access control is necessary with cryptographic storage.

CryptoCache shares many of the advantages of capability systems (flexible delegation, anonymous access, etc.), but also most of the disadvantages.

The CryptoCache trust model is inherently transitive, i.e. if you entrust somebody with a key you trust that he will not give it to somebody who cannot be trusted.

Revocation of keys poses the same problems as revocation of capabilities, but cache files are inherently short-lived, so all keys come with a maximum lifetime. Moreover, if a key is compromised, the roaming user is free to request a fresh copy of the file from his home system with a new set of keys. The compromised file can then be deleted.

The roaming user has to give all users that he collaborates with access to the cache. Confidentiality is guaranteed because only users who know the decryption key can understand the contents of the cached files. Integrity is guaranteed because an attacker cannot sign the modified disk blocks without knowing the encryption key. Authentication is guaranteed because all disk blocks are signed individually with the encryption key. However, availability cannot be guaranteed. Requesting a fresh copy from the home server does not help if a malicious user continuously deletes files in the cache. In this case the roaming user has to request a new file cache (account) with the file hosting service and be more careful about who he gives access to the new cache.

4 Discussion

We have presented the design of a cache mechanism that allows roaming users to temporarily store files on untrusted servers. This improves availability and quality of service for roaming users with compact mobile devices. Moreover, the system allows users to share files with selective access rights (read, write or both).

All files are encrypted before they leave the home server and only decrypted by the client. This ensures confidentiality, integrity and authenticity of the stored data.

The use of public key cryptography replaces traditional access control mechanisms. Files can be served to everybody, because only principals with the public key have the capability to decrypt and read the file and only principals with the private key have the capability to encrypt and sign the data that are written to the file.

CryptoCache has been designed for small mobile devices, but it appears to have applications outside the area of mobile computing. Intelligent caching of application data inside the network is presented as an attractive application for active networks. CryptoCache provides the same ability to push frequently used data onto strategically placed servers in the network. Collaborating servers can securely push data onto untrusted file hosting servers using CryptoCache. All servers participating in the collaboration must have decryption keys for all files, but only the server which actually hosts the file has the encryption key that allows it to update the file in the cache.

References

[1] A. D. Birrell, A. Hisgen, T. Mann C. Jerian, and G. Swart. The echo distributed file system. SRC Research Report 111, Digital Systems Research Center, 1993.

[2] M. Blaze. A cryptographic file system for Unix. In *Proceedings of the first ACM Conference on Computer and Communication Security*, Fairfax, VA, U.S.A, November 1993.

[3] P. J. Braam. The coda distributed file system. *Linux Journal*, (50), 1998.

[4] G. Cattaneo and G. Persiano. Design and implementation of a transparent cryptographic file system for Unix. Technical report, Dip. Informatica e Appl., Universita di Salerno, July 1997.

[5] A. O. Freier, P. Karlton, and P. C. Kocher. The ssl protocol : Version 3.0. Internet-draft, Internet Engineering Task Force (IETF), 1996.

[6] R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page Jr., G.J.Popek, and D. Rothmeier. Implementation of the ficus replicated file system. In *Proceedings of the Summer USENIX Conference*, pages 63–71, June 1990.

[7] P. Reiner, T. Page Jr., G. Popek, J. Cook, and S. Crocker. Truffles – a secure service for widespread file sharing. In *Proceedings of the Privacy and Security Research Group Workshop on Network and Distributed System Security*, 1994.

[8] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5), May 1990.

[9] E. Zadok, I. Badulescu, and A. Shender. Cryptfs: a stackable vnode level encryption file system. Technical report, Computer Science Department, Columbia University, 1998.