Keypad: An Auditing File System for Theft-Prone Devices

Roxana Geambasu John P. John Steven D. Gribble Tadayoshi Kohno Henry M. Levy
University of Washington
roxana, jjohn, gribble, yoshi, levy@cs.washington.edu

Abstract

This paper presents Keypad, an auditing file system for theft-prone devices, such as laptops and USB sticks. Keypad provides two important properties. First, Keypad supports fine-grained file auditing: a user can obtain *explicit evidence* that no files have been accessed *after* a device's loss. Second, a user can disable future file access after a device's loss, even in the *absence* of device network connectivity. Keypad achieves these properties by weaving together encryption and remote key storage. By encrypting files locally but storing encryption keys remotely, Keypad requires the involvement of an audit server with every protected file access. By alerting the audit server to refuse to return a particular file's key, the user can prevent new accesses after theft.

We describe the Keypad architecture, a prototype implementation on Linux, and our evaluation of Keypad's performance and auditing fidelity. Our results show that Keypad overcomes the challenges posed by slow networks or disconnection, providing clients with usable forensics and control for their (increasingly) missing mobile devices.

Categories and Subject Descriptors D.4.6 [*Operating Systems*]: Security and Protection

General Terms design, performance, security

Keywords Keypad, auditing, file system, theft-prone

1. Introduction

Laptops, USB memory sticks, and other mobile computing devices greatly facilitate on-the-go productivity and the transport, storage, sharing, and mobile use of information. Unfortunately, their mobile nature and small form factors also make them highly susceptible to loss or theft. As example statistics, one in ten laptops is lost or stolen within a year of purchase [Nusca 2009], 600,000 laptops are lost annually in U.S. airports alone [Ponemon Institute 2008], and dry cleaners in the U.K. found over 4,000 USB sticks in pockets in 2009 [Sorrel 2010]. The loss of such devices is most concerning for organizations and individuals storing confidential information, such as medical records, social security numbers (SSNs), and banking information.

Conventional wisdom suggests that standard encryption systems, such as BitLocker, PGP Whole Disk Encryption, and TrueCrypt, can protect confidential information. Unfortunately, encryption alone is sometimes insufficient to meet users' needs, for two reasons. First, traditional encryption systems can and do fail in the world of real users. As described in the seminal paper "Why Johnny Can't Encrypt" [Whitten 1999], security and usability are often at odds. Users find it difficult to create, remember, and manage passphrases or keys. As an example, a password-protected USB stick containing private medical information about prison inmates was lost along with a sticky note revealing its password [Savage 2009]. Encrypted file systems often rely on a locally stored key that is protected by a user's passphrase. User passphrases are known to be insecure; a recent study of consumer Web passwords found the most common one to be "123456" [Imperva 2010]. Finally, in the hands of a motivated data thief, devices are open to physical attacks on memory or cold-boot attacks [Halderman 2008] to retrieve passphrases or keys. Even physical attacks on TPMs and "tamper-resistant" hardware are possible [Anderson 1996, Robertson 2010].

Second, when encryption fails, it fails *silently*; an attacker might circumvent the encryption without the data owner ever learning of the access. The use of conventional encryption can therefore lead mobile device owners into a false sense of protection. For example, a hospital losing a laptop with encrypted patient information might not notify patients of its loss, even if the party finding the device has circumvented the encryption and accessed that information.

This paper presents the design, implementation, and evaluation of *Keypad*, a file system for loss- and theft-prone mobile devices that addresses these concerns. The principal goal of Keypad is to provide *explicit evidence* that protected data in a lost device either has or has not been exposed after loss. Specifically, someone who obtains a lost or stolen Keypad device *cannot* read or write files on the device with-

EuroSys'11, April 10–13, 2011, Salzburg, Austria.

out triggering the creation of access log entries on a remote server. This property holds even if the person finding the device also finds a note with the device's encryption password.

Keypad's forensic logs are detailed and fine grained. For example, a curious individual who finds a laptop at the coffee shop and seeks to learn its owner might register audit records for files in the home directory, but not for unaccessed confidential medical records also stored on the device. However, the professional data thief will register accesses to all of the specific confidential medical files that they view. Furthermore, Keypad lets device owners disable access to files on the mobile devices once they realize their devices have been lost or stolen, even if the devices have no network connectivity, such as USB memory sticks (in contrast to systems like Apple's MobileMe).

Keypad's basic technique is simple yet powerful: it tightly entangles the process of file access with logging on a *remote auditing server*. To do this, Keypad encrypts protected files with *file-specific* keys whose corresponding decryption keys are located on the server. Users never learn Keypad's decryption keys and thus they cannot choose weak passwords or accidentally reveal them; it is therefore computationally infeasible for an attacker to decrypt a file without leaving evidence in the log. When a file operation is invoked, Keypad logs the file operation remotely, temporarily downloads the key to access the file, and securely erases it shortly thereafter. Keypad is implemented on top of a traditional encrypted file system; obviously users should choose strong passwords (or use secure tokens, etc.) for that underlying file system, but Keypad provides a robust forensic trail of files accessed even if users choose weak passwords or the traditional system's keys are otherwise compromised.

While conceptually simple, making this vision practical presents significant technical challenges and difficult tradeoffs. For example, neither the user nor Keypad can predict when a device will be lost or stolen. As a result, the system must provide both an accurate fine-grained forensic record, which is critical after loss, and acceptable performance, which is critical prior to loss.

The tension between performance and forensics is pervasive. As an example, consider the creation of a file. For forensic purposes, a naïve Keypad architecture might first pre-register newly created files and their corresponding keys with the remote server prior to writing any new data to those files. However, pre-registration would incur at least one full network round-trip, which could be problematic for some workloads over slow mobile networks, such as 3G or 4G. Delaying the registration is an obvious optimization, yet doing so would leave a loophole that a device thief could exploit to access files without triggering a log entry in the remote server. Overall, our experience demonstrates that we can achieve both forensic fidelity and acceptable performance by combining conventional systems techniques with techniques from cryptography, including identity-based encryption [Boneh 2001, Shamir 1985].

We begin with a description of Keypad's motivation and goals in the following section. Keypad's architecture is presented in Section 3 and its implementation in Section 4. Section 5 provides a detailed evaluation of our prototype and Section 6 discusses its security. Section 7 reviews related work and we conclude in Section 8.

2. Motivation and Goals

Keypad is designed to increase assurances offered to owners of lost or stolen mobile devices. The mobile devices might have computational capabilities (e.g., laptops and phones) or might be simple storage devices (e.g., USB sticks). We view Keypad as particularly valuable to users storing personal or corporate documents, banking information, SSNs, medical records, and other highly sensitive data.

Examples. We provide two brief motivating examples. Alice is a businesswoman who carries a corporate laptop that stores documents containing trade secrets. Alice's IT department installs Keypad on the laptop, configuring it to track all accesses to files in her "corporate documents" folder. After returning to her hotel from a two-hour dinner, Alice notices that her laptop is missing. She immediately reports the loss to her IT department, which disables any future access to files in the corporate documents folder. The IT department also produces an audit log of all files accessed within the two-hour window since she last controlled her laptop, confirming that no sensitive files were accessed.

As a second example, at tax preparation time, Bob scans all of his tax documents, places them on a USB stick, encrypts it with a password, and physically hands the stick and password to his accountant. A few weeks later, Bob can no longer find his thumb drive and can't remember whether his accountant kept it or whether he lost it in the intervening weeks. Fortunately, Bob's stick was protected with Keypad and Bob uses a Web service provided by his drive manufacturer to view an audit log of all accesses to the drive. He sees that there were many accesses to his tax files over the previous week and he learns the IP addresses from which those accesses were made. Bob therefore places fraud alerts on his financial accounts and notifies the appropriate authorities.

In these scenarios, users benefit from additional advantages that Keypad has over traditional encrypted file systems. First, Keypad provides highly accurate, remotely readable forensic records of which files were accessed post-loss. If a file does not appear in those records, that suggests that no one accessed the file after device loss; if a file does appear in those records, this suggests that data was accessed and the owner should take appropriate mitigating actions. Second, by preventing key access, Keypad can prevent adversaries from accessing protected files post-loss, even in the absence of network connectivity, e.g., for a disconnected USB stick or an extracted laptop hard drive.

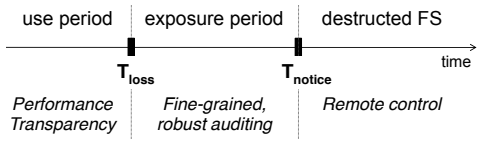


Figure 1. Timeline of theft/loss. This timeline shows the two critical events during the lifetime of a device: the device loss and the user noticing that the device has been lost. For each period, we enumerate the Keypad properties that matter in that period.

Goals. Figure 1 shows a high-level timeline of three periods in the life of a lost or stolen device, along with the properties the user requires during each. First is the normal use period during which the user has control of her device. The user loses control of the device at T_{loss}; however, the user may not know exactly when this occurs, so she must consider T_{loss} to be the last point at which she remembers having control. T_{notice} is the time at which the user realizes that she has lost her device, at which point she should take action. In our Alice scenario, the exposure period (T_{loss} to T_{notice}) is the full two-hour dinner window.

Our primary Keypad goal is to provide strong *audit security*. If an adversary gains control of a device and accesses a Keypad-protected file, at least one audit log entry should be produced on a remote audit server. Further, the adversary cannot tamper with the contents of the audit log or otherwise make it unavailable to the victim. Specifically, our goals are:

- *Robust auditing semantics:* Keypad must provide robust semantics by preventing unrecorded file accesses. To achieve this, the remote auditing server must observe data and metadata operations performed on the client.
- *Performance:* File access latency and throughput should be acceptable for Keypad-protected data. We mainly target office productivity and mobile workloads, rather than server- or engineering-oriented workloads. We also assume multiple network environments: at the office (LANs), at home (broadband), and on the road (3G or 4G). We seek minimal overhead at work or home, but will tolerate some increased latency in challenging mobile environments in exchange for Keypad's properties.
- *Fine granularity:* Keypad should produce detailed access logs of read and write accesses to individual Keypad-protected files. Administrators can control the granularity and coverage of these logs; e.g., configuring Keypad to produce audit logs for an entire file system or only for specific files identified as sensitive.
- *User transparency*: We assume that users are not technically sophisticated; therefore, Keypad's operation should be largely transparent to them and its auditing security should be independent of users' technical competence.
- *Remote access control:* The victim should be able to disable access to protected files after device loss, even if the device has no network or computational capabilities. If an adversary has not yet accessed a protected file, then disabling access prevents any access to the file in the future. If an adversary has already accessed the file, we provide no guarantees about repeat accesses.

These goals mean that device owners will have accurate information about which files have been accessed post-loss. While we will consider optimizations that may introduce extra entries in the audit log, maintaining a *zero false-negative rate* is critical. If a file does not appear in the audit log, then one can confidently say that the file was not accessed. In addition, these audit goals must hold after T_{loss} even if an attacker uses his own software and hardware (and not Keypad) to access the files stored on the device.

We also have several non-goals for Keypad. First, we do not attempt to ensure the device's physical or software integrity after theft/loss. If a user recovers a lost device, he should assume that it has been tampered with, and inspect and reinstall the device from scratch to ensure that no keyloggers or malware have been installed. Second, Keypad deals with device theft/loss that is detectable by a user, and not with surreptitious attacks where an adversary might undetectedly access data on a user's device while he is away. This excludes evil-maid attacks from our threat model [Rutkowska 2009].

Third, Keypad ensures auditability and remote control solely at the file system interface level and below (e.g., the buffer cache). Auditability and control of clear-text data cached in applications' memories is out of Keypad's scope. Fourth, we do not seek to improve the confidentiality of protected files over traditional encryption. Instead, Keypad provides a secure audit log of file accesses if that traditional encryption fails. Finally, we do not guarantee that users can always access Keypad-protected files in the absence of network connectivity (which we consider increasingly rare, given ubiquitous cellular and WiFi networks). However, we do introduce a "paired-device" mechanism to mitigate the impact of disconnection while still maintaining auditability.

3. Keypad Design

Keypad augments encrypted file systems with two properties: auditability and remote data control. The basic idea is simple yet powerful. Keypad: (1) encrypts each file with its own symmetric key, (2) stores all keys on a remote audit service, (3) downloads the key for a file each time it is accessed, and (4) destroys the key immediately after use. This approach supports our auditability and remote data control goals. By configuring the audit service to log all storage accesses, we obtain fine-grained auditability; by disabling all keys associated with a stolen device on the service, we prevent further data access.

Despite its simplicity, designing a practical file system to achieve our goals poses three challenges. First is performance: each file access requires a blocking network request,

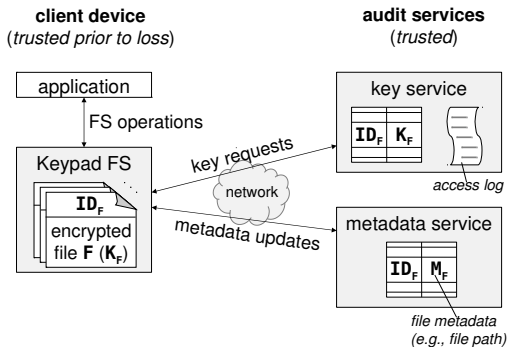


Figure 2. Keypad Architecture. Each file is encrypted with its own random symmetric key. Keys are stored remotely on a key service. To enable forensics, a (separate) metadata service stores file metadata.

which could harm application performance and responsiveness over high latency cellular networks. Second is disconnection: involving the network on all file accesses prohibits file use during network unavailability. While we treat this as an exception, we still wish to support disconnected operation. Third is metadata: an auditor requires user-friendly, up-to-date metadata for each key to interpret access logs appropriately. As will be shown, efficiently maintaining metadata is complex, but possible. This section shows how Keypad's design addresses these three challenges.

3.1 Keypad Overview

Figure 2 shows Keypad's architecture. On the client device, each file F has a unique identifier (called the *audit ID* – ID_F) stored in its header, and the file's data is encrypted with a unique symmetric key, K_F. A remote *key service* maintains the mappings between audit IDs and keys. When an application wants to read or write a file, Keypad looks up the file's audit ID in its header and requests the associated key from the service. Before responding to the request, the service durably logs the requested ID and a timestamp. This process ensures that after T_{notice}, the user will be able to identify all compromised audit IDs for which there is a log entry after T_{loss}.

In addition to the key service, Keypad contains a *metadata service* that maintains information needed by users to interpret the logs. The information (called *file metadata*) includes a file's path, the process that created it, and the file's extended attributes. The metadata and key services fulfill conceptually independent functions; they could be run by a single provider or by distinct providers. Using distinct providers helps to mitigate privacy concerns that could arise if a single party tracked all file access information. The key service sees only accesses to opaque IDs and keys, while the metadata service learns the file system's structure, but not the access patterns. Thus, privacy-concerned users can avoid exposing full audit information to any audit service by using different key and metadata providers.

To meet our goal of robust auditing semantics, Keypad must carefully manage file metadata. For example, when an application creates a new file with name G, Keypad: (1) locally allocates an ID_G for the file, (2) sends a request to the key service to create a new key K_G and bind it to ID_G, and (3) sends a request to the metadata service to register the name G with ID_G. While steps 2 and 3 can occur concurrently, Keypad must confirm that both requests complete before it allows access to the new file. This ensures that file metadata is associated with keys prior to T_{loss}, so that any compromised keys can be correlated with their metadata after T_{notice}.

Similarly, during a file's lifetime, Keypad must keep the service's metadata current to ensure that a user will have fresh information in case of compromise. For example, whenever an application renames a file, Keypad sends a metadata-update request to the metadata service. Keypad must ensure that a thief cannot overwrite the user's metadata with bogus information after theft. For this reason, we implement the metadata store as an append-only log.

3.2 Semantics and Challenges

Keypad provides users with strong auditing semantics at audit time (i.e., post T_{loss}). We formulate an *ideal* invariant describing these semantics as follows:

> For any file **F** with identifier $\mathbf{ID_F}$ that was accessed after $\mathbf{T_{loss}}$ the following properties hold:
>
> (1) the key service shows an $\mathbf{ID_F}$ log entry after $\mathbf{T_{loss}}$, and
>
> (2) the metadata service shows all metadata updates that occurred on $\mathbf{ID_F}$ before $\mathbf{T_{loss}}$.

For (2), the metadata server must contain the latest file metadata (such as file pathname or other attributes) that the user assigned to the file. For example, suppose a user has downloaded a blank IRS tax form into `/tmp/irs_form.pdf`, renamed it as `/home/prepared_taxes_2011.pdf`, and filled it with sensitive information. Then, at forensics time, the user will need to have this latest path available on the service side to interpret a compromise of the taxes file accurately. Hence, maintaining up-to-date service-side metadata is vital to enable meaningful forensics.

In theory, we could achieve semantics arbitrarily close to this ideal invariant. If Keypad downloaded a file's key *every time* a block in the file is accessed and erased the key from memory immediately after using it, then we would obtain the first part of the invariant. Similarly, if Keypad waited for *every* metadata update to be acknowledged by the metadata service before completing that operation on the local disk, then we would obtain the second part.

In practice, however, achieving the ideal invariant is challenging at best. If Keypad must wait a full network round-trip for every block access and for every metadata operation (e.g., `rename`), then the system would be unacceptably slow over high-latency networks. Similarly, disconnected ac-

cess would be impossible. The remainder of this section describes a combination of new techniques and re-purposed traditional mechanisms that help overcome these challenges. While each technique slightly weakens the invariant, we believe that the semantics remain clear and easy to grasp, and that we achieve our goals in nearly all realistic cases.

3.3 Encryption Key Caching and Prefetching

Many of Keypad's critical-path operations are remote key-fetching requests, e.g., issued whenever an application performs a file `read` or `write`. The number of such key requests can be minimized using standard OS mechanisms, such as caching and prefetching. For instance, instead of erasing a key immediately after use, Keypad can cache it locally. Similarly, on access to a file F, Keypad can prefetch keys for other related files, such as those in the same directory. Key caching and prefetching remove key retrieval from the critical path of many file accesses, dramatically improving performance (Section 5).

While caching and prefetching are well understood, they have non-standard implications in our system. First, these techniques cause keys to accumulate in the device's memory, affecting what users can deduce from the audit log of a lost device. Keys that are cached at time T_{loss} are susceptible to compromise: if an adversary can extract them from memory he can permanently remember those keys and bypass audit records for those files. The victim must thus make the worst-case assumption that all keys cached at T_{loss} are compromised. Second, key prefetching creates false positives in the audit log: some prefetched keys may not be used, although records for those keys will appear in the logs.

Keypad must therefore use caching and prefetching carefully to ensure good auditing semantics. For caching, we impose short lifetimes (T_{exp}) on keys and securely erase them at expiration. This bounds key accumulation in memory; the shorter the T_{exp}, the fewer keys will be exposed after T_{loss}. Experimentally, we find that key expirations as short as 100 seconds reap most of the performance benefit of caching, while exposing relatively few keys in memory at a given time. For prefetching, we designed a simple scheme to prefetch keys only when a file-scanning workload is detected (e.g., recursive file search or file hierarchy copying). This benefits file-system-heavy workloads where prefetching is the most useful, while maintaining high auditing precision for light workloads (e.g., interacting with a document). We discuss further prefetching alternatives in Section 4.

Key caching and prefetching alter Keypad's auditing semantics in a clear way: a user must now consider as compromised all files with audit records after $T_{loss} - T_{exp}$. Doing so ensures that the user will never experience false negatives. Hence, these techniques alter the invariant introduced in Section 3.1 in the following way: key and metadata service information must be present for any file F that was accessed after $T_{loss} - T_{exp}$. In Section 5.2 we quantify the effects of caching and prefetching on auditing.

3.4 Identity-Based Encryption for Metadata Updates

Metadata-update file system operations (such as file `create` and `rename`) account for a significant portion of file system operations in many workloads. For example, an OpenOffice file save invokes 11 file system operations, of which 7 are metadata operations that create and then rename temporary files. This large number of metadata operations would result in poor performance over slow networks if Keypad were to wait for an acknowledgement from the metadata service upon *every* metadata update before committing the update to disk, as required by our ideal auditing semantics. Figure 3a shows this scenario.

Overlapping local metadata updates with remote metadata service updates seems like a tempting optimization, however, it opens Keypad to possible attacks and frustrates our semantics. For example, consider a user who creates a new file called `/home/taxes_2011`, writes sensitive tax information inside, and closes the file and editing application. Suppose that due to network failures the create request does not reach the metadata service and therefore the service does not learn the new file's name. If a thief steals the device and reads the tax file ten minutes later, the access will produce an audit trail on the key service; however, no file metadata will be available for the user to interpret the log. Worse, the thief could block Keypad's metadata retries and send a bogus request to the service, e.g., declaring the new file's path as `/tmp/download` to mislead the user.

To respond to this challenge, Keypad leverages identity-based encryption (IBE) [Boneh 2001, Shamir 1985] in a way that both eliminates the network from the critical path of metadata updates and retains its strong auditing semantics. IBE allows a client to perform public-key encryption using any key string it chooses as the public key. A server called a private key generator (PKG) is required to generate the decryption key for the arbitrary public key. Most importantly for our use, the PKG need not know the public key string in advance, but the public key string must be provided to the PKG to learn the decryption key.

We modified Keypad to use IBE as follows. First, we add a level of indirection for file encryption keys. A file F's content is encrypted using a locally-generated random *data key* (denoted K_F^D) stored in the file's header. The data key is *itself* encrypted under the remote key, which in turn is stored on the key server. Section 4 provides more detail.

Second, Keypad's metadata service acts as a PKG, as shown in Figure 3b. When an application invokes a metadata operation (such as `rename`) for a file F, Keypad "locks" its encrypted data key K_F^D in the on-disk file header by encrypting it with IBE, using the new file's pathname as the public key string. While the metadata request is in flight, reads and writes can proceed *as long as* a copy of the file's cleartext data key K_F^D is cached in memory. Because files with metadata updates in flight are vulnerable to attacks, we reduce the key expiration time for such files to the bare minimum nec-

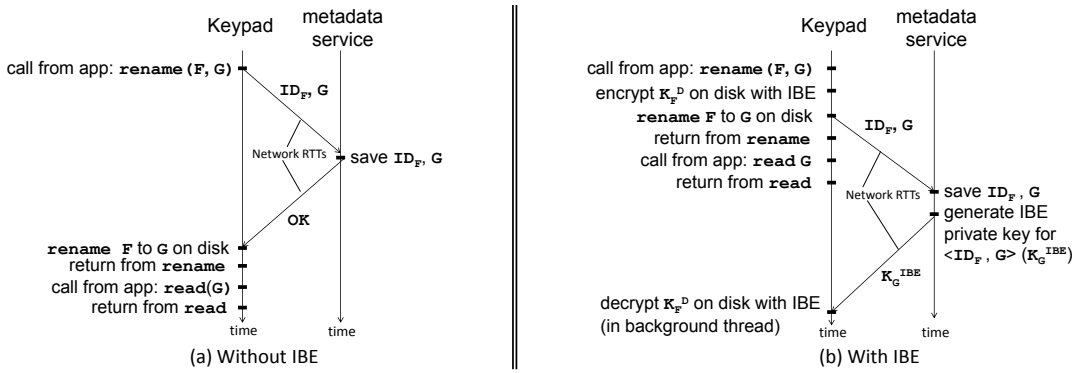


Figure 3. Timelines for handling metadata-update operations without IBE (a) and with IBE (b). The application is assumed to issue a `rename(F, G)` followed by a `read(G)`. Assuming that a copy of F's decryption key is cached in memory, IBE allows overlap of accesses to F with the metadata service request until the cached key times out (1 second in our system).

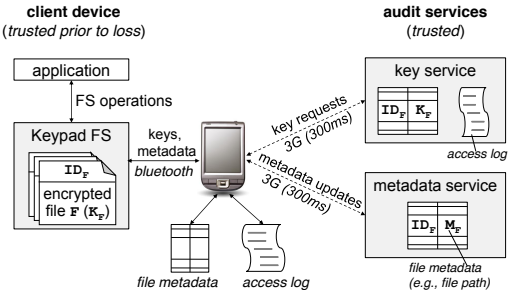


Figure 4. Paired-device architecture. By pairing a laptop with a mobile phone, Keypad supports disconnected operation and may even improve performance.

essary to hide network latencies on cellular networks. For example, our prototype expires cached keys with in-flight metadata updates in *one second*, minimizing attack opportunity. After the cached key times out, the file is essentially "locked" on disk by the IBE encryption, preventing subsequent file accesses until the metadata service confirms its success. On confirmation, the metadata service returns the IBE private key, allowing Keypad to "unlock" the file.

Suppose an attack or network failure prevents the service from registering the new metadata and subsequently the device is stolen. In the (extremely likely) case that the theft occurred more than one second after the user's rename request, the file's cached data key will have expired and the thief will need to obtain the IBE private key in order to unlock the file for access. As a result, the thief is forced to supply the *correct* file pathname to the metadata service if he desires to read the file; lying or avoiding the metadata update will prevent him from gaining access. Therefore, the thief cannot access the file without causing an audit record associated with correct and up-to-date metadata to be logged on the corresponding audit services.

3.5 Using Paired Devices for Disconnected Access

Although disconnected operations are assumed to be the exception rather than the rule, Keypad must still support them. One option is to cache keys for an extended period of time and accumulate metadata registrations locally. However, this forces the user to give up auditability for the disconnected duration, which can be dangerous. Further, caching is not applicable to storage-only devices like USB sticks. To address this issue, we developed a *paired-device* extension to Keypad that supports disconnected operations without sacrificing auditability semantics.

Many of today's users carry multiple devices when they travel, such as a laptop as well as a smart phone or a tablet. These devices support short-range, low-latency networks, such as Bluetooth. The paired-device architecture, shown in Figure 4, uses a cell phone as a transparent extension of the Keypad key and metadata services. Keypad on the laptop is configured as usual, using strict caching, prefetching, and metadata registration policies to ensure fine-grained auditing. The phone is configured to hoard [Kistler 1991] any recently used keys, cache them until connectivity is restored, log any accesses and metadata updates to the local disk, and upload the logs when connectivity returns. If only the laptop is lost, the phone is used along with the audit service logs to provide a full audit trail. If the phone is stolen along with the laptop, then the audit service will list more files as exposed than if the laptop were stolen alone.

In addition to supporting (increasingly rare) disconnected cases, the paired-device architecture has another advantage: it can improve performance over slow mobile networks without sacrificing auditing. Because the laptop–phone link is relatively efficient, the paired phone can improve laptop performance by acting as a cache for it. Here the phone is configured to perform aggressive directory-level key prefetching and caching. On a key miss, the laptop contacts the phone via bluetooth and the phone returns the key, if available; otherwise the phone fetches the missed key and other related keys from the key service and returns the key to Keypad. Section 5 evaluates the performance improvement for this solution. As before, auditing properties are preserved if only

the laptop is stolen. If both devices are stolen, then auditing is at a directory-level granularity.

3.6 Partial Coverage

Not all files necessarily require audit log entries. For example, as a trivial optimization we could exclude non-sensitive files such as binaries, libraries, and configuration files from Keypad's audited protection domain. In this scenario protected files are encrypted locally and their keys and metadata are stored remotely; unprotected files are (optionally) encrypted locally, but their encryption keys are derived from the user's login credentials.

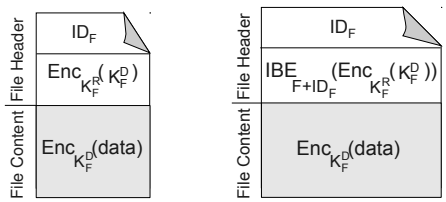
The benefits of this optimization are obvious: Keypad's performance and availability costs are only incurred for protected files. There is also a risk: if a sensitive file is accidentally placed in an untracked file or directory, the audit logs will not reveal accesses to that sensitive data. One reasonable protection policy is to track accesses to any file in crucial directories, such as the user's home and temporary directory (e.g., `/home` and `/tmp` on Linux).

3.7 Summary

Keypad provides strong guarantees to its users. If a protected file is accessed, then at least one record related to that access will appear in the remote audit logs, and up-to-date metadata about the file will be available online. As we have shown, one challenge Keypad faces is preserving this strong property while overcoming the performance impact of communicating with remote services in the critical path of file accesses. We introduced a series of novel techniques to meet this challenge. Though some of these techniques have an impact on the quality of the information in the audit logs, we show in Section 5.2 that this impact is small.

4. Implementation

We implemented a Keypad prototype including the client-side Keypad filesystem, the key service, and the metadata service as shown in Figure 2. All components are coded in C++ and communicate using encrypted XML-RPC with persistent connections. Our client-side Keypad file system is an extension of EncFS [EncFS 2008], an open-source block-level encrypted file system based on FUSE [Fuse 2004]. EncFS encrypts all files, directories, and names under a single volume key, which is stored on disk encrypted under the user's password. Keypad extends EncFS in two ways. First, we modified EncFS to encrypt each file with its own per-file key. The single volume key is still used, however, to protect file headers and the file system's namespace, e.g., file and directory names. Second, Keypad stores all file keys on a remote key server and maintains up-to-date metadata on a metadata server. To support forensic analysis we built a simple Python tool; given a T_{loss} timestamp and an expiration time, T_{exp}, the tool reconstructs a full-fidelity audit report of all accesses after $T_{loss} - T_{exp}$, including full path names and access timestamps.



(a) Keypad File. (b) IBE-Locked File.

Figure 5. Keypad File Formats. Keypad on-disk file structure for the normal case (a) and the IBE-locked case (b).

Keypad File Structure. Figure 5(a) shows the internal structure of a Keypad file F, which consists of two regions: the file's header and its content. The file's header is fixed size and is encrypted using EncFS' volume key. For the file's content, our implementation adds a level of indirection for encryption keys to support techniques such as IBE efficiently. Specifically, file F's content is encrypted using a 256-bit random *data key*, denoted K_F^D. The data key is stored in the file's header encrypted under the *remote key*, denoted K_F^R. The remote key is stored on the key server and is identified by the file's audit ID (ID_F), which is a randomly generated 192-bit integer that is stored in the file's header along with the encrypted data key. This internal file structure is transparent to applications, which see only the decrypted contents of a file.

FS Operations. Keypad intercepts and alters two types of EncFS operations: file-content operations (`read`, `write`) and metadata-update operations (`create`, `rename` for files or directories). When an application accesses file content, Keypad: (1) looks up the file's audit ID from its header, (2) retrieves the remote key K_F^R, either from the local cache or the key service, (3) decrypts the data key K_F^D using K_F^R, (4) caches K_F^D temporarily, and (5) decrypts/encrypts the data using K_F^D.

When an application creates or updates file metadata, Keypad: (1) locks the data key using IBE, if enabled, and (2) sends the new metadata to the metadata service. The metadata is the file's path reported as a tuple of the form `directoryID/filename`. The names of Keypad directories are also kept current on the metadata service. While our current prototype applies IBE for file metadata update operations (e.g., file `create`, `rename`), it does not apply it to directory metadata operations (e.g., `mkdir` or directory `rename`), although this should be possible to add.

Key Expiration. Keypad caches keys for a limited time to improve performance. A background thread purges expired keys from the cache. If a key has been reused during its expiration period, the thread requests the key from the key service again, causing an audit record to be appended to the access log for that audit ID. If a response arrives before the key expires, the key's expiration time is updated in the cache, otherwise the key is removed. As a result, absent network failures, keys in Keypad never expire while in use. This

ensures that long-term file accesses, such as playing a movie, will not exhibit hiccups due to remote-key fetching.

Key Prefetching. Key prefetching attempts to anticipate future file accesses by requesting file keys before the files are accessed. For our prototype, we sought a simple policy that would have both reasonable performance and little impact on auditability. We have experimented with two policies: (1) a random-prefetch scheme that prefetches random keys from the local directory upon every key-cache miss and (2) a full-directory-prefetch scheme that prefetches all keys in a directory when it detects that the directory is being scanned by an application. Our experiments indicated that the latter policy provided equally good performance, while incurring fewer false positives in the audit logs. Hence, our Keypad prototype uses it by default. The intuition behind our full-directory prefetch design is to avoid producing false positives for targeted workloads (such as interacting with a document, viewing a video, etc.) and to improve performance for scanning workloads (such as grepping through the files in a directory or copying a directory). Our full-directory-prefetch scheme avoids recursive prefetches to ensure that any false positives are triggered by real accesses to (related) files in the same directory. While other more effective prefetching policies may exist, our results show that our full-directory-prefetch policy, combined with our caching policies, reduce the number of blocking key requests to a point where the performance bottleneck shifts from blocking key requests to metadata requests (see Section 5).

IBE. To avoid blocking for metadata-update requests, our prototype implements IBE-based metadata registration, using an open-source IBE package [Boneh 2002]. On a metadata-update operation, Keypad locks the file until the metadata service confirms the receipt of the new file path; however, file operations can proceed for a one-second window, as previously described, to absorb the registration latency. Figure 5(b) shows the structure of an IBE-locked file. Its encrypted data key is further encrypted using IBE under a public key consisting of the file's path (`directoryID/filename`) and the audit ID (ID_F). Embedding ID_F into the public key strongly binds ID_F and the path together at the metadata server. Handling updates for other types of file metadata functions (such as `setfattr`) works similarly, although our current prototype only supports pathnames as metadata.

Android-Based Paired-Device Prototype. We implemented a prototype of the paired-device architecture (Figure 4) using the Google Nexus One phone. A simple daemon (431 lines of Python) on the phone accepts key requests from the laptop over Bluetooth, saves accesses to a local database, responds to the laptop, and uploads access and metadata information to Keypad servers in bulk over wireless.

5. Evaluation

This section quantifies Keypad's performance and auditing quality. Keypad must be fast enough to preserve the usability of desktop and mobile applications, even in the face of adverse network conditions (e.g., 3G), while providing high quality auditing.

For our experiments, we used an eight-core 2GHz x86 machine running Linux 2.6.31 as our client. Our key service and name service daemons ran on 8 core 2.6GHz servers with 24GB of RAM, connected via gigabit Ethernet. We used Linux's traffic control utility to emulate different network latencies. We did not emulate different bandwidth constraints, however, Keypad's bandwidth requirements are very low. During a 12-day period in which one of our authors used Keypad continuously, average Keypad bandwidth was under 5 kb/s, with occasional spikes up to 45 kb/s.

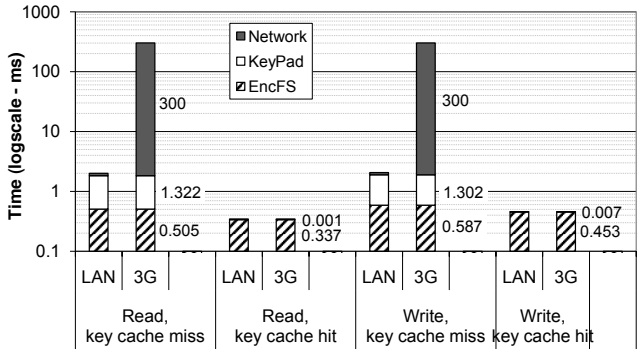
Throughout the evaluation, we emulate the following RTTs for various networks: 0.1ms RTT for a LAN, 2ms RTT for a wireless LAN (WLAN), 25ms RTT for broadband, 125ms RTT for a DSL network, and 300ms RTT for a 3G cellular network. To illustrate network latency effects on Keypad performance, we often use examples from extreme network conditions, such as fast LANs and slow 3G networks, even though popular mobile connections today rely on WLAN and 4G.

5.1 Performance

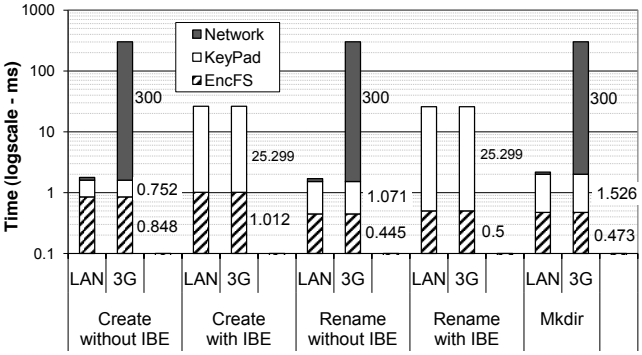
To understand where the time goes for Keypad operations, we microbenchmarked file content (`read` and `write`) and metadata (`create`, `rename`, and `mkdir`) operations. Our measurements included client, server, and network latencies, as well as latency contributions for EncFS and Keypad.

Figure 6(a) shows the latency of file read and write operations for two cases: key-cache misses, which must fetch the key from the the server, and key-cache hits, which use a locally cached key. For each case we show data for two extreme networks: a fast 0.1ms-RTT LAN and a slow 300ms-RTT 3G network. The results show that misses are expensive on both networks, but for different reasons. On a LAN, the network is insignificant, but Keypad adds to the base EncFS time due to the XML-RPC marshalling overhead. On 3G, network latency dominates. When the key-cache hits, both the network and marshalling costs are eliminated; a file read with a cached key is only 0.01ms slower than the base EncFS read time of 0.337ms. This shows the importance of key caching to avoid misses, which we accomplish by carefully choosing our expiration and prefetching policies.

Figure 6(b) shows the latency of file metadata update operations. For `create` and `rename`, we show latency with and without IBE; `mkdir` is shown only without IBE, since it does not benefit from this optimization in our prototype. Without IBE, metadata update latency is driven primarily by network RTT: file creation takes 1.618ms on a LAN, and 302ms over 3G. With IBE, metadata update latency is inde-



(a) FS Content Operations: `read`, `write`.



(b) FS Metadata Operations: `create`, `rename`, `mkdir`.

Figure 6. File Operation Latency. The latency of Keypad (a) content and (b) metadata-update operations. For each, we show the time spent in EncFS code, Keypad client and server code, and on the network. Labels on the graph show the latency for each component in the 3G 300ms RTT case. Results are averaged over 10 trials with a warm disk buffer cache.

pendent of network delay and is dominated by the computational cost of IBE itself. The figure shows that IBE meets its goal of improving performance of metadata updates over 3G. While IBE would add overhead for a LAN, it is unnecessary and would be disabled in the LAN environment.

5.1.1 Optimizations

We now demonstrate the effectiveness of our optimizations on a challenging workload: Apache compilation. While this workload is not characteristic of mobile devices, its complex nature make it ideal for evaluating the impact of our optimizations. In Section 5.1.2, we extend our evaluation to more typical workloads for mobile devices. As baselines, the Apache compilation takes 112s using the unmodified EncFS encrypted file system (i.e., with encryption but without auditing) and 63s on `ext3` (i.e., without encryption or auditing). Because Keypad enhances EncFS, the fair baseline comparison for Keypad is EncFS, and not `ext3`.

In what follows, we inspect the effect of optimizations as we enable one optimization after the other. We begin

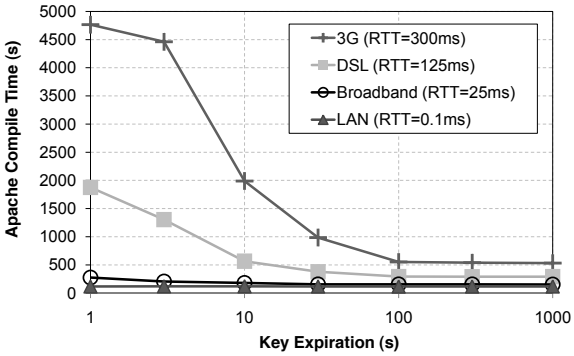


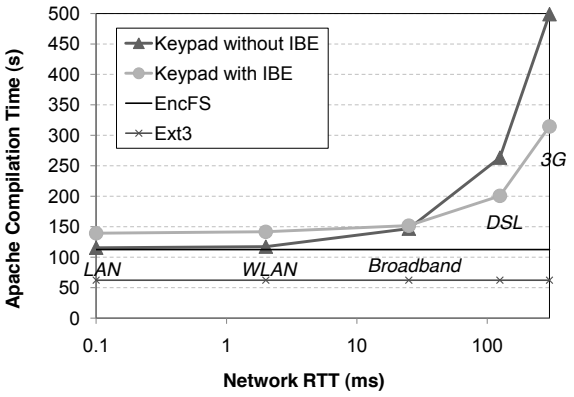
Figure 7. Effect of Key Expiration Time. This graph shows the effect of key expiration without any other optimizations enabled. A 100-s key expiration time is nearly optimal, and achieves compilation times of 115s, 153s, 292s, and 551s over a LAN, Broadband, DSL, and 3G, respectively. For comparison, the Apache compilation takes 112s on the unmodified EncFS and 63s on `ext3`.

by showing the effect of purely key caching with no other optimizations, then we add prefetching, then IBE, and finally we add the paired-device optimization.

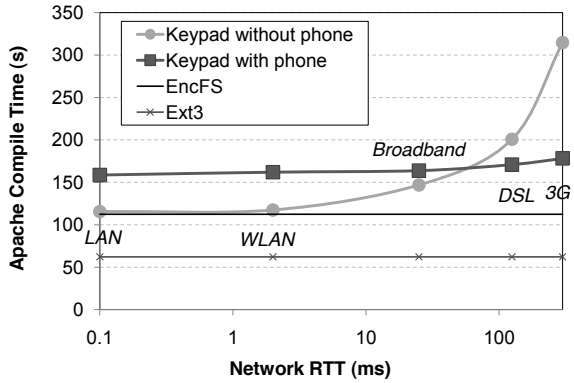
Key Caching and Expiration. Key caching is crucial to performance. Even a cache with one-second expiration time has significant impact: 18% improvement on a LAN and 4.9x on 3G, relative to no caching at all. Figure 7 shows additional improvements for Apache compilation time as expirations are lengthened beyond one second. No optimizations other than caching are enabled here. Our results suggest that short expiration times are sufficient to extract nearly all the benefits. For LAN, Broadband, or DSL latencies, an expiration of 10s or so is optimal. Over 3G, a 100s key expiration time achieves all the benefit and provides 8.6x improvement over 1s (from 79.4 minutes down to 9.2 minutes). In comparison to EncFS, Keypad's performance degradation for 100s expiration times is already small over a LAN (5.3% overhead over EncFS), while for the other network types, further optimizations are required for performance.

Note that a 100s timeout is extremely small. To benefit from cached keys, a thief needs to steal the device within 100 seconds of the user's last access. Even in such cases, the user will know which files were exposed. We therefore believe that we can achieve both good performance and accurate auditing with these parameters.

Directory-Key Prefetching. Key caching alone avoids many key service requests: of the 75,744 reads and writes in the Apache compilation, only 486 involve the server when using a 100s expiration time. Directory-key prefetching avoids additional server requests. Prefetching a directory key on the first, third, or tenth miss in a directory results in 101, 249, and 424 key-cache misses, which translates into 63.3%, 24.1%, and 2.4% improvements, respectively, over not using directory-key prefetching over 3G. We adopted a prefetch-on-third-miss policy to strike a good balance between per-



(a) Effect of IBE.



(b) Effect of Device Pairing.

Figure 8. Effect of IBE and Device-pairing Optimizations. (a) Effect of applying the IBE optimization atop a 100-s key caching policy and a third-miss prefetching policy; no device pairing is used here. (b) Effect of applying the device-pairing optimization atop the optimization setup in (a).

formance and auditing quality (which is evaluated in Section 5.2). Over fast networks, such as a LAN and WLAN, the prefetch-on-third-miss policy coupled with 100-s key caching results in negligible performance overheads compared to EncFS: 2.8% for LAN and 4.3% for WLAN. Over slower networks, especially 3G, other smarter prefetching policies may improve performance by further eliminating blocking key requests. However, we find that with our simple prefetching policy, the dominating runtime component now becomes the blocking metadata requests (932 blocking metadata requests compared to the 249 blocking key requests). We next focus on optimizing metadata requests.

IBE. IBE tolerates the latency of metadata service requests over slow mobile networks. Figure 8(a) shows the impact of IBE on Apache compilation as a function of network RTT. As we see in the figure, IBE provides dramatic improvements on high-latency networks, including 3G- and 4G-class networks. For example, IBE improves the benchmark's performance on 3G by 36.9%. The crossover for IBE is around 25ms, i.e., it should be used only for networks with RTTs over 25ms and disabled otherwise. As mentioned above, for faster networks, such as LANs or WLANs, IBE is not even necessary, as Keypad's overhead is already negligible after applying key caching and prefetching.

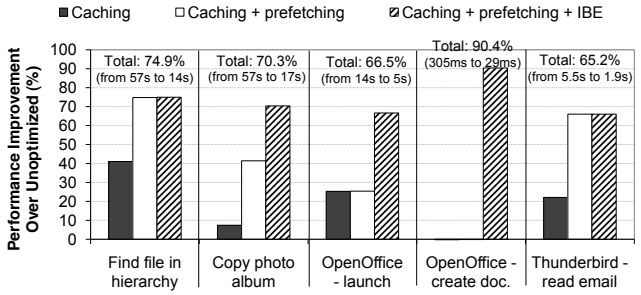


Figure 9. Impact of Optimizations on Various Applications. Impact of three of the optimizations on an emulated 3G network; labels indicate the total performance improvement when using all three optimizations over the unoptimized case, as well as the absolute numbers for the unoptimized and optimized.

The Paired Device. Our paired device design is aimed at facilitating disconnected operation, but it can also provide performance benefits for high-latency network environments. Figure 8(b) shows the effect on the Apache workload of using a paired device as a caching proxy for key and metadata services. Two conclusions can be reached from the figure. First, performance for disconnected operation over Bluetooth should be similar to or better than that of a broadband connection (the latencies are similar). Second, pairing with another device is always beneficial for performance over cellular networks, because most operations only traverse the lower latency Bluetooth link. Obviously the paired device should not be used if fast networks are available, where Keypad is already efficient enough compared to EncFS.

5.1.2 Office-Oriented Workloads

Figure 9 shows the impact of our optimizations on more typical office-oriented workloads. We add optimizations incrementally, reporting additional improvement as more optimizations are added. The labels on top of each bar group show the total improvement with all three optimizations enabled. Different workloads benefit the most from different optimizations, depending primarily on the relative frequency of those operations. For example, caching and prefetching are important for a read-intensive workload such as a recursive grep ("Find file in hierarchy"). IBE provides large improvements for workloads that create files ("OpenOffice – create doc"). For mixed content/metadata workloads, such as copying a photo album across directories, all optimizations are important.

To better understand performance across many applications, we benchmarked the time to perform a number of pop-

Application	Task	Time (seconds)					
			Keypad				
		EncFS	LAN (RTT=0.1ms)	WLAN (RTT=2ms)	Broadband (RTT=25ms)	DSL (RTT=125ms)	3G (RTT=300ms)
OpenOffice Word Processor	Launch	0.5	0.5 | 0.5	0.6 | 0.6	1.3 | 1.3	2.7 | 2.7	4.6 | 4.6
	New document	0.0	0.0 | 0.0	0.0 | 0.0	0.0 | 0.0	0.0 | 0.1	0.0 | 0.3
	Save as	1.4	1.4 | 1.4	1.4 | 1.4	1.5 | 1.5	1.6 | 1.8	2.0 | 2.3
	Open	1.7	1.7 | 1.7	1.8 | 1.8	2.0 | 2.2	2.1 | 4.0	2.1 | 7.5
	Quit	0.1	0.1 | 0.1	0.1 | 0.1	0.3 | 0.4	0.4 | 0.7	0.4 | 1.2
Firefox	Launch	3.7	3.7 | 3.7	3.8 | 3.8	4.4 | 4.4	6.0 | 6.0	8.8 | 8.8
	Save a page	0.7	0.7 | 0.7	0.7 | 0.7	0.7 | 0.8	0.9 | 1.5	1.3 | 2.8
	Load bookmark	4.5	4.5 | 4.5	4.5 | 4.5	4.5 | 4.6	4.5 | 5.0	4.5 | 5.7
	Open tab	0.2	0.2 | 0.2	0.2 | 0.2	0.2 | 0.2	0.2 | 0.4	0.2 | 0.8
	Close tab	0.0	0.0 | 0.0	0.0 | 0.0	0.0 | 0.0	0.0 | 0.1	0.0 | 0.3
Thunderbird	Launch	1.3	1.3 | 1.3	1.3 | 1.3	1.4 | 1.4	2.0 | 2.0	3.1 | 3.1
	Read email	0.3	0.4 | 0.4	0.4 | 0.4	0.5 | 0.6	1.0 | 1.5	1.9 | 2.5
	Quit	0.2	0.2 | 0.2	0.2 | 2.2	0.2 | 0.4	0.2 | 1.3	0.2 | 2.9
Evince PDF Viewer	Launch	0.1	0.1 | 0.1	0.1 | 0.1	0.1 | 0.1	0.1 | 0.1	0.1 | 0.4
	Open document	0.1	0.1 | 0.1	0.1 | 0.1	0.1 | 0.1	0.2 | 0.2	0.4 | 0.4
	Quit	0.0	0.0 | 0.0	0.0 | 0.0	0.0 | 0.0	0.0 | 0.0	0.0 | 0.0

x | y: x = time with warm key cache
y = time with cold key cache

Table 1. Typical Application Performance Over Keypad. For Keypad, we show both warm and cold key-cache times, separated by a |.

ular tasks using EncFS and Keypad over several emulated networks (Table 1). For Keypad, we show both warm and cold key-cache times. A user will likely experience both, but with well-chosen key expiration times many operations will be absorbed by a warm cache.

From a user's perspective, Keypad performs roughly identically to EncFS over fast networks, such as a LAN and a wireless LAN. Hence, while at the office, the user should never feel our file system's presence, whether its key cache is warm or cold. With only a few exceptions, the user should perceive similar application performance over broadband with Keypad and the unmodified EncFS. Over mobile networks, the user may notice some application slowdown, especially after extended periods of inactivity.

The table and our own experience confirm that application launches are particularly expensive over 3G networks, as they often encounter a cold cache and many file system interactions. Keypad could optimize launch by profiling applications and prefetching needed keys; other file systems, such as NTFS, perform similar special-case optimizations to speed up application launch.

5.1.3 Comparison to Other File Systems

A networked file system might be an alternative to Keypad; instead of just storing keys remotely, all file system content would be remote. NFS provides a reasonably fair comparison to Keypad, since its short-term caching might provide audit properties comparable to ours. In contrast, for AFS and Coda, their long-term, coarse granularity caching policies might interfere more with precise audit semantics.

Figure 10 shows the relative performance of Keypad to (remote) NFSv3 and (local) EncFS for Apache compilation. We configured NFS with asynchronous batched writes and its default caching policy; this improves its performance, but

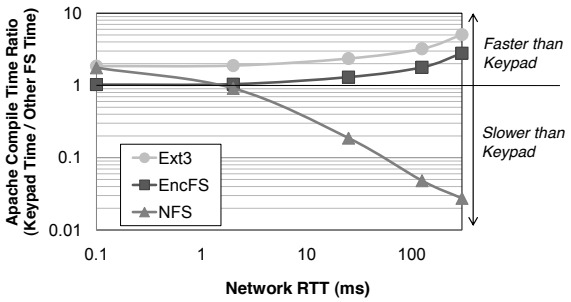


Figure 10. Comparison to EncFS and NFS.

would have some impact on auditing. Note that for these experiments, as before, we emulated different network RTTs but we did not constrain network bandwidth; thus, our results are upper bounds of NFS performance. Over actual 3G links, NFS performance would be significantly degraded because of wireless bandwidth constraints.

With LAN latencies, Keypad's performance is almost identical to EncFS with only a 2.78% increase in runtime, but worse than NFS, with a 75% increase. For reference, the unmodified EncFS itself is 71% slower compared to NFS with LAN-like latencies. As RTT grows, NFS degrades significantly. Even with an RTT of 2ms, NFS is 8.8% *slower* than Keypad, while for 3G network latencies of 300ms, NFS is 36.4x slower than Keypad! In contrast, Keypad is only 2.7x slower than EncFS over a 300ms network.

On large-RTT networks, NFS impacts interactivity. For example, launching OpenOffice over NFS with 3G latency takes 50.6 seconds, loading a bookmark in Firefox takes 27.6 seconds, and opening an email in Thunderbird takes 12.5 seconds, which we believe is unacceptable performance for these user-facing tasks.

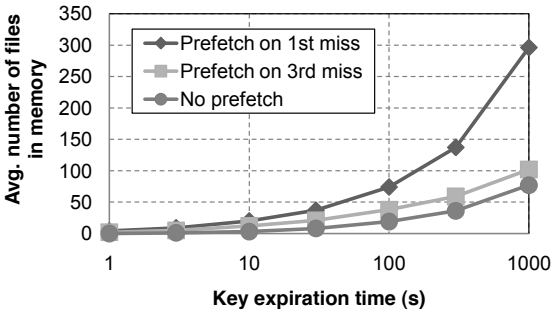


Figure 11. Effect of Optimization on Auditability. The average number of keys that reside in memory at any point in time, under various key expiration times and prefetching policies.

5.1.4 Anecdotal Experience

Anecdotally, one co-author used Keypad continuously to protect his laptop's `$HOME` and `/tmp` directories over a 12-day period, with an emulated 300ms client-to-server latency. Overall, the experience was positive: in most cases, there was no noticeable performance impact. Some activities, such as file system intensive CVS checkouts or recursive copies, were slower but usable. Other more typical activities, such as browsing the Web, editing documents, and exchanging email, had no noticeable performance degradation.

5.2 Auditing Properties

We now evaluate our optimizations' impact on auditability.

In-memory Key Sets. As described in Section 3.3, keys for recently-accessed or prefetched files stay in memory for their expiration period T_{exp}. This is not an issue for a thief who steals a passive storage device, such as a USB stick. For a laptop, because a thief can theoretically access cached-key files without triggering a server-side audit log, users must consider all files whose keys were retrieved between $T_{loss} - T_{exp}$ and T_{loss} as compromised. The size of this set at any point in time depends on the user's workload and on the aggressiveness of the caching and prefetching schemes.

To quantify this issue we used a trace gathered during our twelve-day deployment experience (Section 5.1.4) to calculate the impact of various optimizations on auditability. Figure 11 shows the size of the in-memory key set at any point in time averaged over use periods, for different key expiration times and prefetching policies. The graph shows that for reasonable key expiration and prefetching strategies, the average number of in-memory keys is small. For example, with a 100-second key expiration time and a prefetch-directory-on-third-miss strategy, on average there are 38 keys in memory at any instant. This is a small number and furthermore we observed that most of these keys exist as a side-effect of prefetching; i.e., they are files in the same directory as a file that was accessed by a user or program.

False Positives. Prefetching affects forensics by introducing false positives in the audit log. The rate of false positives depends on the prefetching policy as well as the thief's workload, since false positives only concern time post-T_{loss}. In the absence of an accepted "thief workload," we created a few scenarios that a thief might follow. Our goal was to gauge the impact of various prefetching policies on the rate of false positives, as a thief tries to find sensitive information on a captured device. We investigated three scenarios: (1) the thief launches Thunderbird, reads a few emails, browses folders, and searches for emails with a particular keyword; (2) he launches a document editor and looks at a few files; and (3) he inspects the history, bookmarks, cookies, and passwords in a Firefox window. For these workloads, our default prefetch policy (prefetch directory keys on the 3rd miss) leads to the following ratios between false positives and total accessed keys: 3:30, 6:67, and 0:12 for our Thunderbird, document editor, and Firefox workloads, respectively. Audit precision is high for these scenarios.

We have also discovered bad scenarios; if the thief navigates to a web page in Firefox, loading several files from the cache directory causes Keypad to prefetch the entire directory. While this causes several false positives, the user correctly learns that activity happened in the Firefox cache directory. Even in such cases, the auditing implications of our non-recursive prefetching policy are minimal, since all false positives are localized to one directory.

5.3 Summary

We measured the performance of our Keypad prototype on several workloads. Our measurement results and our experience using the system show that Keypad meets its goals of adding little overhead in the office or home environment, while remaining highly usable over cellular networks, such as 3G. Overall, our results show that with properly parameterized optimizations, Keypad can provide good performance while also maintaining good auditing fidelity. Furthermore, with current and future improvements of cellular network connectivity (e.g., 4G), we expect Keypad to have even better performance.

6. Security Analysis

Keypad is designed to provide strong audit guarantees for encrypted file systems if the first layer of defense, encryption with a password or cryptographic token, is breached. Keypad can additionally destroy the ability to read files after a mobile device is reported lost or stolen. Although we evaluated security properties extensively inline above, we now return for a unified discussion.

Context and Threat Model. We designed Keypad assuming that individuals who find or steal a mobile device range in sophistication, degree of planning, and interest. Curious individuals may insert a found USB stick into their computer, enter the password on the attached sticky note, and browse through a few files trying to find the device owner. Petty thieves may grab laptops opportunistically but have no

real interest in accessing confidential files. Corporate spies may plan and execute device theft carefully, with the goal of accessing confidential files before the victim reports the device missing. We refer to all such individuals as "attackers."

Because a user has no way of knowing the motivation and skill of a potential attacker, Keypad assumes the worst. We assume that an attacker has full access to the lost device's hardware (for laptops and USB sticks) and software (for laptops). The attacker can perform cold-boot attacks on laptops, install new software, and manipulate or sever the device's network traffic. The attacker can also perform lower-level activities, such as physically extracting the hard drive from a laptop or memory from a USB stick and interrogating it with custom hardware. However, we do not consider attacks in which the adversary gains control of the device, modifies it, and returns it to the victim without his knowledge (see our non-goals discussion in Section 2). Any attacker with control over a device while in the user's possession could mount a slew of malicious attacks outside the scope of a forensic file system, ranging from online data exfiltration to the installation of password key loggers. Botnets and other forms of malware are therefore also explicitly outside our threat model.

Analysis. We begin with the premise that the audit servers are trusted and secure. The key and metadata servers are trusted to maintain accurate logs, and they are assumed to incorporate strong defenses to adversarial comprise, routinely back up their state, and have their own audit mechanisms. Neither the key server nor the metadata server is, however, fully trusted with the private information about a user's file access patterns prior to T_{loss}; accessing that information requires collusion between both servers or the device owner's invocation of the Keypad post-loss audit mechanisms. The unavailability of servers can deny access to files; for highly sensitive data, we argue that users would prefer unavailability over the potential for unaudited future file disclosure. Further, although not implemented in our prototype, the communications between the Keypad file system and the servers should be encrypted to ward off attackers who intercept network communications prior to device theft. The keys must change every T_{exp} seconds to ensure that an attacker who extracts the current network encryption key from the device cannot decrypt past intercepted data.

Consider now an attacker who obtains a lost or stolen Keypad device. If the device is cold, such as a powered-down laptop or a USB stick, then any successful attempt to access a protected file must generate at least one log record on the Keypad audit servers. This is true whether the attacker uses the Keypad file system or his own hardware or software to perform the access. All of Keypad's mechanisms – the storing of K_F^R on remote servers, the entangling of the metadata server and key server states to ensure consistency, and our method for using IBE – enforce this property. Additionally, the selection of 192-bit audit IDs at random makes it infeasible for an attacker to request information about valid audit IDs from the key and metadata servers prior to physically obtaining the protected device; such requests are additionally thwarted by authenticating the device to the servers.

Attackers who obtain warm, computational devices – such as running or hibernated laptops – may seek to violate the properties of Keypad by directly accessing the device's memory. Cached keys K_F^R should be evicted from memory upon device hibernation, and such evictions should be recorded on the audit servers. For fully running devices, we must assume that an attacker has accessed any file with an audit log entry after $T_{loss} - T_{exp}$. Although Keypad's focus is on providing file system auditing, a forensic analyst must also acknowledge that applications may have sensitive data in memory. A conservative analyst might use various heuristics to identify potentially vulnerable cleartext data. For example, he might mark as compromised any file opened since the device's last boot or hibernation, events that could be recorded on the audit servers. A potentially better future solution to this problem might be to employ encrypted memory technology [Provos 2000], possibly coupled with auditing.

Most importantly, even against an attacker who obtains warm computational devices, Keypad preserves the following invariant: if an analyst does not mark a file as accessed, then one can confidently conclude that the file has indeed not been accessed by an attacker. Finally, because entries in the key service are identified per-device, the service can deny access to all relevant keys if a device is reported missing.

For completeness, we must also consider an attacker who attempts to generate spurious entries in the remote audit logs. While such spurious entries might complicate the task of a forensic analyst, an attacker cannot use such actions to hide their actual accesses of confidential data.

7. Related Work

Keypad is related to previous work in three areas: (1) theft-protection systems, (2) data-protection systems, and (3) distributed file systems.

Theft-Protection Systems. Commercial and research theft-protection systems, such as Apple's MobileMe and Adeona [Ristenpart 2008], rely on software running on a device that can monitor file accesses, report device locations and file accesses to a remote server, and delete files upon request. A determined attacker can circumvent these protections and analyze the device's media using his own hardware, without the associated monitoring software installed. Keypad provides strong forensic and data-destruction capabilities even against thieves who use their own hardware and software to attack a Keypad-protected file system.

Data-Protection Systems. Encrypted file systems exist in academia (e.g., [Blaze 1993]) and industry (e.g., BitLocker, PGP Whole Disk, TrueCrypt). None provide remote auditing capabilities, therefore a security breach may go undetected. Keypad's forensic and data-destruction capabilities are or-

thogonal to work increasing the resilience of encrypted file systems to breach. Keypad can compose with new advances in encrypted file systems, providing both stronger barriers to access and a forensic trail if that barrier is breached.

ZIA [Corner 2002] and follow-on work [Corner 2003] protect files on a device with transient authentication. ZIA users wear small tokens that broadcast their presence. When a token is near a protected device, the device decrypts; when the token leaves, the device re-encrypts. Protection is lost if an attacker obtains both the device and the token, with no forensic guarantees. Keypad does not require a paired device, but if one is used, Keypad still provides a forensic trail of accesses even if both are lost or stolen. Keypad could be combined with ZIA for additional defense in depth.

Keypad's remote key-escrow architecture has been used frequently in the past to achieve a number of security and privacy goals. First, capture-resilient cryptography [MacKenzie 2001] uses a key server to prevent dictionary attacks against login passwords on stolen devices, as well as to enable remote wipe-out. Second, location-aware encryption [Studer 2010] uses a remote key server to dynamically adapt a device's data protection level based on its location. While the device is at a trusted location (e.g., at its owner's home), the server provides the decryption key; when the device is at an unknown or untrusted location, the server will require the user to enter a special password to return obtain the decryption key. Third, assured-delete systems, such as the Ephemerizer [Perlman 2005], revocable backup systems [Boneh 1996], and the Vanish distributed-trust self-destructing data system [Geambasu 2009] adopt the key-escrow architecture to ensure the deletion of sensitive data stored in backup systems or on Web services. Keypad resembles all of these systems in its remote key-escrow architecture and its secondary goal: post-theft data destruction. It differs from these systems in its primary goal: fine-grained auditability of mobile device data accesses.

In general, today's data-protection systems differ from our system in that they focus on data exposure *prevention*, whereas Keypad focuses on data exposure *detection* should prevention systems fail. In that sense, they should be considered as complementary rather than competitors.

Networked File Systems. Work in distributed file systems has aimed at providing shared and available remote storage (e.g., [Howard 1988, Lee 1996, Sandberg 1985]). Bayou [Peterson 1997] and Coda [Mummert 1995] support mobility, disconnected operation and data consistency. Coda's disconnected operation [Kistler 1991] relies on data caching, whereas Keypad uses device pairing, coupled with key caching, to support offline accesses. Coda supports encrypted communication but not storage. LBFS [Muthitacharoen 2001] uses compression to reduce latency for interactive file access over slow wide-area networks. SFS [Fu 2002, Mazieres 1999] is a network file system that supports secure network file transfers, avoiding the need for distributed key infrastructure by embedding public keys in file pathnames. SFS is concerned with secure communication, not with protecting a user's stored data from theft; it does not encrypt data on disk and does not support auditing.

In general, these systems do not support encryption and auditing. While they could be modified to support both on the server, there are significant performance issues, e.g., streaming an NFS-hosted video over 3G or wireless is slow and expensive. Finally, all of these systems are concerned with the transfer of *file data* between a client and server; in contrast, Keypad is concerned with *key management* and the transfer of encryption keys between a file system and a remote key server. Keypad is unique in its support for (and integration of) encryption and audit logging; it demonstrates the advantage of separating encryption and key management to enforce auditing for mobile device data.

8. Conclusions

This paper described Keypad, an auditing file system for loss- and theft-prone devices. Unlike basic disk encryption, Keypad provides users with evidence that sensitive data either was or was not accessed following the disappearance of a device. If data was accessed, Keypad gives the user an audit log showing which directories and files were touched. It also allows users to disable file access on lost devices, even if the device has been disconnected from the network or its disk has been removed. Keypad achieves its goals through the integration of encryption, remote key management, and auditing. Our measurements and experience demonstrate that Keypad is usable and effective for common workloads on today's mobile devices and networks.

9. Acknowledgements

We offer thanks to our shepherd Mahadev Satyanarayanan and the anonymous reviewers for their valuable comments on the paper. This work was supported by NSF grants CNS-0846065, CNS-0627367, and CNS-1016477, the Google Fellowship in Cloud Computing, the Alfred P. Sloan Research Fellowship, the Torode Family Career Development Professor, the Wissner-Slivka Chair, and a gift from Nortel Networks.

References

[Anderson 1996] Ross Anderson and Markus Kuhn. Tamper resistance: A cautionary note. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce (WOEC '96)*, 1996.

[Blaze 1993] Matt Blaze. A cryptographic file system for UNIX. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93)*, 1993.

[Boneh 2001] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '01)*, 2001.

[Boneh 2002] Dan Boneh, Matthew K. Franklin, Ben Lynn, Matt Pauker, Rishi Kacker, and Gene Tsudik. Identity-based encryption download. `http://crypto.stanford.edu/ibe/download.html`, 2002.

[Boneh 1996] Dan Boneh and Richard Lipton. A revocable backup system. In *Proceedings of the 6th USENIX Security Symposium*, 1996.

[Corner 2002] Mark D. Corner and Brian D. Noble. Zero-interaction authentication. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom '02)*, 2002.

[Corner 2003] Mark D. Corner and Brian D. Noble. Protecting applications with transient authentication. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys '03)*, 2003.

[EncFS 2008] EncFS. EncFS encrypted filesystem. `http://www.arg0.net/encfs`, 2008.

[Fu 2002] Kevin Fu, M. Frans Kaashoek, and David Mazieres. Fast and secure distributed read-only file system. *ACM Transactions on Computer Systems (TOCS)*, 20(1):1–24, 2002.

[Fuse 2004] Fuse. Filesystem in userspace. `http://fuse.sourceforge.net/`, 2004.

[Geambasu 2009] Roxana Geambasu, Tadayoshi Kohno, Amit Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proceedings of the 18th USENIX Security Symposium*, 2009.

[Halderman 2008] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[Howard 1988] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.

[Imperva 2010] Imperva. Consumer password worst practices. `http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf`, 2010.

[Kistler 1991] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In *Proceedings of the 13th ACM Symposium on Operating System Principles (SOSP '91)*, 1991.

[Lee 1996] Edward K. Lee and Chandramohan A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '96)*, 1996.

[MacKenzie 2001] Philip MacKenzie and Michael K. Reiter. Delegation of cryptographic servers for capture-resilient devices. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS '01)*, 2001.

[Mazieres 1999] David Mazieres, Michale Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, 1999.

[Mummert 1995] Lily B. Mummert, Maria R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of 15th ACM Symposium on Operating Systems Principles (SOSP '95)*, 1995.

[Muthitacharoen 2001] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, 2001.

[Nusca 2009] Andrew Nusca. How to: Keep your laptop from being stolen. `http://www.zdnet.com/`, February 2009.

[Perlman 2005] Radia Perlman. File system design with assured delete. In *Proceedings of the 3rd IEEE International Security in Storage Workshop (SISW '05)*, 2005.

[Peterson 1997] Karin Peterson, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97)*, 1997.

[Ponemon Institute 2008] Ponemon Institute. Airport insecurity: The case of lost and missing laptops; U.S. and EMEA result. `http://www.ponemon.org/data-security`, 2008.

[Provos 2000] Niels Provos. Encrypting virtual memory. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

[Ristenpart 2008] Thomas Ristenpart, Gabriel Maganis, Arvind Krishnamurthy, and Tadayoshi Kohno. Privacy-preserving location tracking of lost or stolen devices: Cryptographic techniques and replacing trusted third parties with DHTs. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[Robertson 2010] Jordan Robertson. `http://www.usatoday.com/tech/news/computersecurity/2010-02-08-security-chip-pc-hacked_N.htm`, 2010.

[Rutkowska 2009] Joanna Rutkowska. Evil maid goes after TrueCrypt! `http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html`, 2009.

[Sandberg 1985] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the Sun network file system. In *Proceedings of the USENIX Annual Technical Conference*, 1985.

[Savage 2009] Michael Savage. NHS 'loses' thousands of medical records. `http://www.independent.co.uk/news/uk/politics/nhs-loses-thousands-of-medical-records-1690398.html`, 2009.

[Shamir 1985] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of the 5th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '85)*, 1985.

[Sorrel 2010] Charlie Sorrel. Brits send 4,500 USB sticks to the cleaners. `http://www.wired.com/`, 2010.

[Studer 2010] Ahren Studer and Adrian Perrig. Mobile user location-specific encryption (MULE): Using your office as your password. In *Proceedings of the 3rd ACM Conference on Wireless Network Security (WiSec '10)*, 2010.

[Whitten 1999] Alma Whitten and J.D. Tygar. Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, 1999.