International Journal of Computer and Internet Security.
ISSN 0974-2247 Volume 4, Number 2 (2012), pp. 43-50

http://www.irphouse.com

Keypad Based Security for Networking Devices to Regain Control over Meta Data

[1]M.Vijaya Bhaskar Rao, [2]M.Arjun Goud and [3]Prof.V.Purnachandra Rao

[1]Kakatiya University Andhra Pradesh, India
[2]Lecturer, Satavahana University, Karimnagar Andhra Pradesh, India
[3]Professor,TKR Institute of Technology and Science Andhra Pradesh, India
E-mail: arjun.muthyala2000@gmail.com

Abstract

This paper discusses about Keypad, an auditing file system for theft-prone devices, such as laptops, tablets, and USB sticks. While emerging computing technologies such as cloud computing and small, powerful, mobile devices offer previously global access to data and applications, they also threaten user's control over data ownership, distribution, and properties. This paper examines the broad data security, and management challenges raised by keypad technology and proposed a set of techniques to address these issues. Key pad offers remote access control and auditability for data stored on a stolen devices. Firstly we presented detailed description about keypad architecture and implemented a prototype for keypad. Secondly we discussed about the key pad technology performance and finally the security analysis for keypad technology is discussed.

Keywords: Mobile Devices, Keypad, Keys, Encryption, Security, File system, Information

Introduction

The mobile devices, such as laptops, tablets, and USB memory sticks, create not only great advantages but also significant risks due to their susceptibility to theft and loss. 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 Bit Locker [1], PGP Whole Disk Encryption [2], and True Crypt [3], can protect confidential information. However, encryption alone is sometimes

insufficient to meet users' needs. Two reasons are relevant for this discussion. First, traditional encryption systems can and do fail in the world of real users. 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 [4]. 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" [5]. Finally, in the hands of a motivated data thief, devices are open to physical attacks on memory or cold-boot attacks [6] to retrieve passphrases or keys.

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 without 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 accidental 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 traditional system's keys are otherwise compromised.

Keypad Architecture

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, 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.

Architectural Overview

Figure 1 shows Keypad's architecture. On the client device, each file F has a unique identifier (called the audit ID – IDF) stored in its header, and the file's data is encrypted with a unique symmetric key, KF. 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 Tnotice, the user will be able to identify all compromised audit IDs for which there is a log entry after Tloss.

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

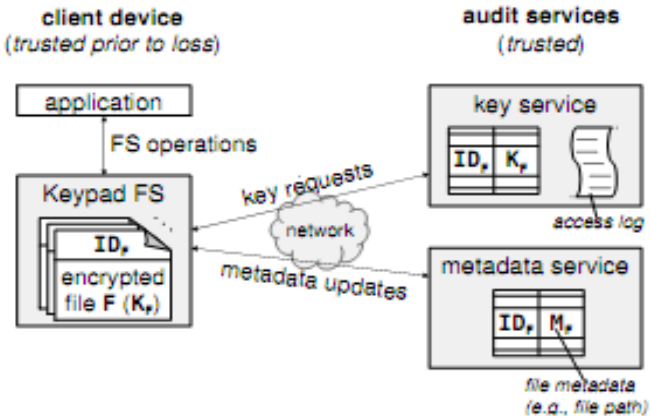


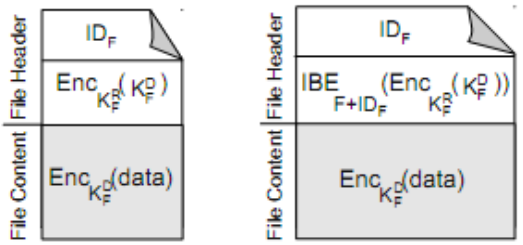
Figure 1: The Keypad System Architecture. Each file is encrypted with its own random symmetric key. Keys are stored remotely on a key service. To enable forensics, a meta data service stores file metadata

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.

Prototype 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 [7], an open-source block-level encrypted file system based on FUSE [8]. 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.

Keypad File Structure: Figure 2(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 KFD. The data key is stored in the file's header encrypted under the remote key, denoted KFR. The remote key is stored on the key server and is identified by the file's audit ID (IDF), which is a randomly generated 192-bit integer that is stored



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

Figure 2: Keypad File Formats. Keypad on disk file structure for the normal case (a) and the IBE locked case (b)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 KFR, either from the local cache or the key service, (3) decrypts the data key KFD using KFR, (4) caches KFD temporarily, and (5) decrypts/encrypts the data using KFD .

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 directory ID/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 for 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 grapping 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.

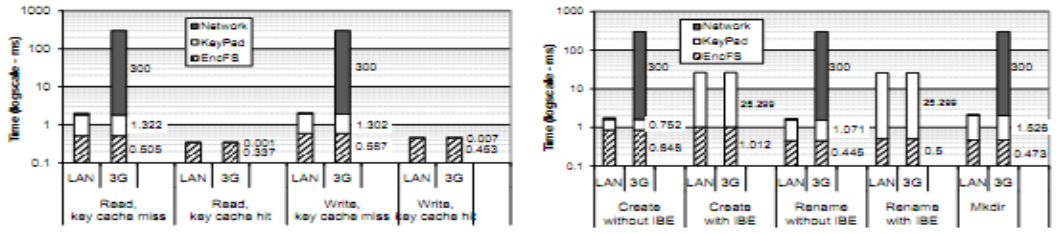
IBE: To avoid blocking for metadata-update requests, our prototype implements IBE-based metadata registration, using an open-source IBE package [9]. On a metadata-

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 network latency. Figure 2(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 (directory ID/filename) and the audit ID (IDF). Embedding IDF into the public key strongly binds IDF and the path together at the metadata server. Handling updates for other types of file metadata functions works similarly, although our current prototype only supports pathnames as metadata. An attacker cannot pre-obtain private IBE keys for popular file paths from the metadata server prior to stealing the device, because directory and file IDs are drawn at random from a gigantic space (2192).

Android Based Paired Device Prototype: We implemented a prototype of the paired-device architecture 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. It leverages the key derivation mechanism to easily fetch directory keys upon a key-cache miss. For example, when the laptop requests a file key, the Nexus will fetch the parent directory's key from the key server, cache it, compute the file's key by applying HMAC to the directory key, and return the file key to the laptop. Thus, when pairing with the Nexus, the key server provides directory-level auditing, while the phone offers fine-grained auditing.

Performance

To understand where the time goes for Keypad operations, we micro benchmarked 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.



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

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

Figure 3: 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.

Figure 3 shows the latency of file read and write operations for two cases: key-cache misses, which must fetch the key from 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.

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."

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.

Conclusion

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 wasnot 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 demonstrate that Keypad is usable and effective for common workloads on today's mobile devices and networks.

References

[1] Microsoft Bit Locker. Windows 7 Bit Locker Executive Overview.http://technet. microsoft.com/en-us/library/dd548341%28WS.10%29.aspx, 2009.
[2] PGP Corporation. PGP whole disk encryption. http://www.pgp.com/products/wholediskencryption/, 2008.
[3] T. Foundation. True crypt – free open-source on-the-fly encryption. http://www. truecrypt.org/, 2007
[4] M. Savage. NHS 'loses' thousand of medical records. http://www.independent.co.uk, 2009
[5] Imperva. Consumer password worst practices. http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf, 2010.
[6] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Proceedings of the USENIX Security Symposium, 2008.
[7] EncFS. http://www.arg0.net/encfs.
[8] FUSE: File system in User space. http://fuse.sourceforge.net/.
[9] PBC. http://crypto.stanford.edu/pbc/.