

# Transcryptfs

A Dynamically Loadable Kernel-space Filesystem Architecture

*A Thesis Submitted  
in Partial Fulfilment of the Requirements  
for the Degree of*

**Master of Technology**

*by*

**Adarsh J**

**Roll No. : 10111004**

*under the guidance of*

**Prof. Rajat Moona and Prof. Dheeraj Sanghi**

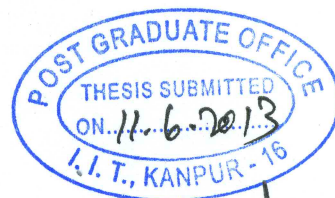


Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

June, 2013

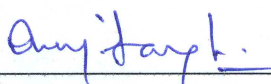
## CERTIFICATE



It is certified that the work contained in this thesis entitled "*Transcryptfs, A Dynamically Loadable Kernel-space Filesystem Architecture.*", by Adarsh J(Roll No. 10111004), has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

---

(Prof. Rajat Moona)  
Department of Computer Science and Engineering,  
Indian Institute of Technology Kanpur  
Kanpur-208016

  
\_\_\_\_\_  
(Prof. Dheeraj Sanghi)  
Department of Computer Science and Engineering,  
Indian Institute of Technology Kanpur  
Kanpur-208016

June, 2013

# Abstract

TransCrypt is an encrypting filesystem, indigenously developed at IIT Kanpur. It has been under continuous development for the past 6 years, by various students of IIT Kanpur at Prabhu Goel Research Centre for Computer and Internet Security. The codebase of Transcryptfs filesystem for Linux is spread across various subsystems of Linux kernel, viz. dm-layer, lsm, vfs, crypto, etc., making it difficult to keep Transcryptfs updated with changes in any of those subsystem in upstream kernel's mainline code. One of the main features of Transcryptfs for Linux file server is the use of Linux security module (LSM) for providing access control mechanism; Due to changes in kernel architecture, since late 2007, LSM has to be statically linked with kernel during its compile time, thereby imposing a restriction that Transcryptfs supported kernel should be statically compiled with the Transcryptfs-lsm module. This additionally imposed restriction has led to a complicated procedure for setting up of Transcryptfs filesystem and also has shifted the responsibility of patching kernel with new updates from the distribution managers to individual system administrators. These factors have resulted in non-adoption of Transcryptfs for real world usecase.

In this thesis, we re-engineer and come up with an architecture that aims to make Transcryptfs an easily deployable and dynamically loadable kernel module, in addition to decoupling codebase from the kernel source into a single manageable module, thereby easing maintenance and further development of Transcryptfs filesystem. The contribution of this thesis is in designing the kernel-space architecture and related implementation of Transcryptfs as an out-of-the-tree kernel module.



*Dedicated to  
Paranoids and the FOSS Community.*



# Acknowledgement

I would like to express my sincere gratitude towards my thesis supervisors, Prof. Rajat Moona and Prof. Dheeraj Sanghi, for their guidance, constant support and encouragement.

I thank Department of Computer Science and Engineering, IIT Kanpur, for providing the necessary infrastructure and a congenial environment for research work.

I also thank Satyam Sharma, Sourav Khandelwal, Prateek Mishra, Rahul Krishna and my batch mates of M.Tech 2010 for their valuable inputs and discussions. I am forever indebted to my family for supporting me throughout.

*Adarsh J*





# Contents

<b>Abstract</b>	<b>iii</b>
<b>List of Figures</b>	<b>xiii</b>
<b>Abbreviations</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Related Work . . . . .	2
1.2.1 eCryptfs . . . . .	2
1.2.2 TrueCrypt . . . . .	2
1.2.3 Other encrypting filesystems . . . . .	3
1.3 Contribution of this Thesis . . . . .	3
1.4 Organization of this Thesis . . . . .	4
<b>2 Background</b>	<b>5</b>
2.1 Stacked Filesystem . . . . .	5
2.2 Features of Transcriptfs . . . . .	6
2.3 Issues with existing Transcriptfs Implementation . . . . .	7
<b>3 Architecture</b>	<b>9</b>
3.1 Components of Transcriptfs Kernel Module . . . . .	9
3.1.1 Crypto Manager . . . . .	9
3.1.2 Communication Manager . . . . .	10
3.1.3 Keystore Manager . . . . .	11

3.1.4	Other supporting components . . . . .	11
3.2	User Space Components . . . . .	11
3.2.1	AuthServer . . . . .	12
3.2.2	Transcryptfsd . . . . .	12
3.2.3	Other support utilities . . . . .	12
<b>4</b>	<b>Metadata Organization and Key Management</b>	<b>15</b>
4.1	Filesystem Metadata File . . . . .	16
4.2	File System Key (FSK) . . . . .	16
4.3	File Encryption Key (FEK) . . . . .	16
4.4	Access Control List (ACL) . . . . .	17
4.5	In-Memory Key Management . . . . .	17
<b>5</b>	<b>Implementation of Transcryptfs Kernel Module and Results</b>	<b>19</b>
5.1	Mounting of Transcryptfs . . . . .	19
5.2	Opening of a File . . . . .	20
5.3	Read/Write from a File . . . . .	21
5.4	Adding an ACL Entry . . . . .	21
5.5	Performance Test Setup . . . . .	22
5.6	Results . . . . .	23
5.6.1	Write performance . . . . .	23
5.6.2	Read performance . . . . .	24
5.6.3	Interpretation of Results . . . . .	24
<b>6</b>	<b>Conclusion and Future Work</b>	<b>25</b>
6.1	Summary . . . . .	25
6.2	Future Work . . . . .	26
6.2.1	Better support for caching within kernel module . . . . .	26
6.2.2	Integration of data recovery process within kernel module . . .	26
6.2.3	SSHFS support for network access . . . . .	26

<b>A User's Manual</b>	<b>27</b>
A.1 System Requirements . . . . .	27
A.2 Installation . . . . .	27
<b>Bibliography</b>	<b>29</b>



# List of Figures

3.1	Transcryptfs Architecture . . . . .	10
4.1	Filesystem Metadata Format . . . . .	16
5.1	Write Performance . . . . .	23
5.2	Read Performance . . . . .	23



# Abbreviations Used In This Thesis

- FS: Filesystem Server. This is the server on which the Transcriptfs filesystem is hosted.
- PKS: Private Key Store. The PKS stores the private key of the user and provides an interface to perform private key related cryptographic operations on a given message.
- FEK: File Encryption Key. This key is used for symmetric encryption and subsequent decryption of file data.
- FSK: File System Key. This is unique for each Transcriptfs volume and is used to blind the FEK.
- $E_U^A(m)$ : Asymmetric key encryption of message  $m$  with the public key of the entity,  $U$ .
- $E_k^S(m)$ : Symmetric key encryption of message  $m$  with the key,  $k$ .
- $H^{MD5}(m)$ : MD5 hash of message  $m$ .





# Chapter 1

## Introduction

### 1.1 Motivation

Storage devices are used in almost every electronic computation device today. Over the years, the cost per gigabyte of storage has been decreasing exponentially[19]. The secondary storage devices can now store terabytes of data and yet, cost only a small fraction of the value of the data they store. The premise that data is much more valuable than the cost of media on which it resides, sets us up with a challenge of securing it within the justifiable cost. This problem was partly answered by having encrypted storage systems, one such solution is a filesystem designed by IIT Kanpur, known as Transcriptfs[35].

Transcriptfs is a kernel-space encrypting filesystem that incorporates an advanced key management scheme to provide a high grade of security while remaining transparent and easily usable. The implementation of Transcriptfs has evolved over multiple years from originally being a modification over ext3 filesystem[34], followed by a key management implementation[3], then moving encryption process to device mapper[38] and later making Transcriptfs filesystem independent[26], further effort were directed into allowing filesystem to be accessible from a networked environment[18][1][24], followed by addition of support for windows client[2] as well as supporting Transcriptfs server on windows operating system[5][27].

## 1.2 Related Work

### 1.2.1 eCryptfs

eCryptfs is an enterprise cryptographic stacked filesystem for Linux. It was initially conceived as a stackable filesystem by Erez Zadok[43] and later implemented and merged into mainline kernel by Michael Halcrow[10]. eCryptfs is based on symmetric key cryptography and it stores the cryptographic meta data in the underlying encrypted filesystem. Our work on Transcryptfs's architecture has been inspired by eCryptfs, the notable differences are the use of in-kernel public key cryptography for key-management and the use of asynchronous block encrypting APIs in Transcryptfs. The trust model of eCryptfs includes the userspace of Linux operating system, which we see as a drawback in its security model as is later explained in Section 2.2 of this thesis. The functional feature set of eCryptfs is the closest to Transcryptfs, compared to any other encrypting filesystem. Hence, we will compare with eCryptfs while evaluating the performance of Transcryptfs.

### 1.2.2 TrueCrypt

TrueCrypt is a userspace software which does on-the-fly encryption of files in a TrueCrypt volume[37]. TrueCrypt operates completely in the userspace and uses XTS-AES mode of operation for encrypting data. Our work is inspired by the practical demonstration of effectiveness of XTS-AES[4] encryption mode in TrueCrypt. TrueCrypt supports Windows, Linux and OS X operating systems, thereby providing the user with a choice of platform and providing the compatibility of encrypted volume across various platforms. Even though it theoretically provides strong encryption and decryption, the fact that it does so in userspace makes it insecure and vulnerable to various userspace attacks when it is mounted and is in operation.

### 1.2.3 Other encrypting filesystems

Apart from the above noted encrypting storage solutions, there are few other commonly used encrypting filesystems, including:

- dmccrypt/LUKS

This is a block device based disk encryption system. It does not provide a per file encryption key and instead encrypts the whole disk with same key. It supports use of any encryption mechanism supported by Linux Kernel.

- FreeOTFE

This is an on-the-fly disk encryption, which supports both Windows and Linux Operating Systems. It operates in userspace and performs its operations similar to TrueCrypt. One of the notable features of FreeOTFE is that it can be used in a portable manner without requiring a full fledged installation.

## 1.3 Contribution of this Thesis

The primary contribution of this thesis is in design of architecture and implementation of Transcryptfs. We achieve the objective of having a dynamically loadable kernel module for easy deployment and configuration of Transcryptfs on a Linux server. We build upon the stackable filesystem model on the lines of ecryptfs. We have designed an architecture so as to keep up with the spirit of initial design of Transcryptfs, that is, to provide a trust model with least number of trusted entities, thereby making it more secure. We also extend the mode of encryption used in implementation of Transcryptfs to a new standard defined by NiST for storing encrypted data on secondary storage, XTS-AES[25].

In the course of implementation, we have ported public key infrastructure supporting functionality into kernel. We ported a popular user space library PolarSSL[36] into kernel to provide various functionalities like big integer operations, signing and verification operations.

## 1.4 Organization of this Thesis

The rest of this thesis is organized as follows. Chapter 2 sets the background by explaining the concept of stacked filesystem and briefly describing the features of Transcriptfs and its existing problems. Chapter 3 describes the architecture of kernel space and various kernel space modules of Transcriptfs. Chapter 4 explains the metadata format and its organization along with key-management within the kernel space. Chapter 5 delves into details of working of Transcriptfs kernel module and compares its performance with ecryptfs. Chapter 6 concludes the work done in this thesis and provides suggestions towards the possible extensions for future work.

# Chapter 2

## Background

The concept of overlaying one filesystem over another is known as stacked filesystem. It is achieved by registering the filesystem callbacks in Virtual File System (VFS) layer[30]. Stacked filesystem was first introduced by Heidemann et al[11]. WrapFS is a bare-bone template of a stacking filesystem which is generated by FiST framework[42]. WrapFS in its default state is a null layer which offers copying of filename and page data across the stacked layer[44]. During our development, we chose WrapFS as our initial template. We built various features around it to suit the needs of Transcryptfs architecture.

### 2.1 Stacked Filesystem

The concept of stacking filesystem provides us with a strong abstraction of functionality provided by each stacking layer. Further, since the stacked filesystem is implemented via VFS layer callbacks, one can insert the kernel module during runtime, providing the stacking functionality dynamically. This makes it possible for us to introduce new kernel module operations via inserting corresponding kernel module for our VFS, without a need for rebooting the kernel. Such modules are known as Dynamically Loadable Kernel Modules (DLKM)[41][32].

## 2.2 Features of Transcriptfs

Transcriptfs has been designed with the following features:

- Minimal Trust Model

Transcriptfs trusts only the kernel-space components with the handling of secret data. It does not rely even on root (superuser) for operations other than mounting passphrase, which is a one time operation. Since it trusts very few entities, it is hard to compromise the security of data.

- Protection of data with offline attacks

Transcriptfs does not store any key material in the secondary storage. All the unencrypted data resides only in the main memory during run time. The secondary storage consists of encrypted data and metadata which is also either encrypted or hash of key material. Hence, it makes the offline attacks ineffective for decrypting secure data.

- Use of PKI for securing key material

Transcriptfs introduces Public Key Infrastructure into kernel space. By introducing PKI, we can perform strong cryptographic operations within kernel thereby making brute force attacks ineffective on RSA encrypted key material. We also make use of public certificates to trust entities. Entities (Users) identify themselves via their private key store (PKS).

- Modeled for Enterprise Usecase - Backup and Key escrow

In order to have a usable filesystem at an enterprise level, filesystem should support basic functionalities like regular backup and an escrow mechanism in case of unfortunate eventualities. Transcriptfs filesystem can easily be backed up without compromising data by accessing encrypted filesystem via lower filesystem path. Since Transcriptfs stores metadata associated with encrypted file within the file data itself, there are no special mechanisms required for taking backups. Regular archiving utilities will work without any changes. For

escrowing data from a file, Transcryptfs provides an option to store a Data Recovery Agent's (DRA) token for every file, the private key store of DRA is shared using m-out-of-n secret sharing scheme[33] to n different entities, such that m out of n entities can come together to recover the key for decrypting data according to the organization's data recovery policies.

## 2.3 Issues with existing Transcryptfs Implementation

The existing implementation of Transcryptfs on Linux has various sub systems of Transcryptfs spread throughout the various subsystems of kernel source code. Broadly, PKI related operations are implemented as asymmetric crypto operations within `crypto` subsystem of kernel; access control mechanisms of Transcryptfs are implemented in `security` subsystem as an LSM (Linux Security Modules) module - which imposed an additional restriction of static integration with kernel at the compile time. This would mean that anyone who wishes to run a customized version of kernel, would have to recompile the kernel to support Transcryptfs; Device mapper related functions were implemented in `dm` (device mapper) subsystem as a driver within kernel; Core of Transcryptfs required modifications in kernel's `fs` (filesystem) subsystem. Clearly, with such a distribution of code, any minor changes in any of those subsystems in mainline kernel might break the Transcryptfs implementation, hence maintenance of Transcryptfs becomes a liability. Given the fast pace at which various subsystems of kernel change it would be a problem for us to keep up with the latest version of kernel.

The other major problem or rather lack of feature is that, one has to create a new filesystem altogether with the existing approach. Thereby, losing out on various features that could have been capitalized by using a different filesystem at base and having Transcryptfs only as an encryption layer instead.

The earlier implementation of Transcryptfs makes the installation process tedious

and requires administrator to have an in-depth understanding of how to compile kernel. This has been an impediment in wide acceptance of Transcriptfs as secure filesystem. These issues have been the focus of our work in current thesis. We have addressed these problems in our new architecture as described in the later chapters.



# Chapter 3

## Architecture

This chapter describes the overall architecture of Transcriptfs kernel module including its interfaces to the userspace. We briefly list userspace components which have been described in detail by Sourav Khandelwal in his thesis[17]. The architecture follows the loadable kernel module pattern[41], which primarily allows the filesystem module to be dynamically added to the kernel space in runtime, without a need for system restart. The kernel module of Transcriptfs is made up of several subsystems, which handle specific operations as described in the following sections.

### 3.1 Components of Transcriptfs Kernel Module

#### 3.1.1 Crypto Manager

Crypto Manager subsystem of Transcriptfs implements interfaces for cryptographic operations provisioned by kernel[7] along with other cryptographic primitives which are ported from userspace library, PolarSSL[36]. The ported functions include implementation of RSA[29], PKI[20][40] related functions like X.509 certificate[13] parsing and big integer operations.

Crypto Manager is responsible for setting up the cryptographic contexts. These cryptographic contexts are used to perform cryptographic operations within kernel space. These cryptographic contexts are associated in `mount_crypt_stat` structure

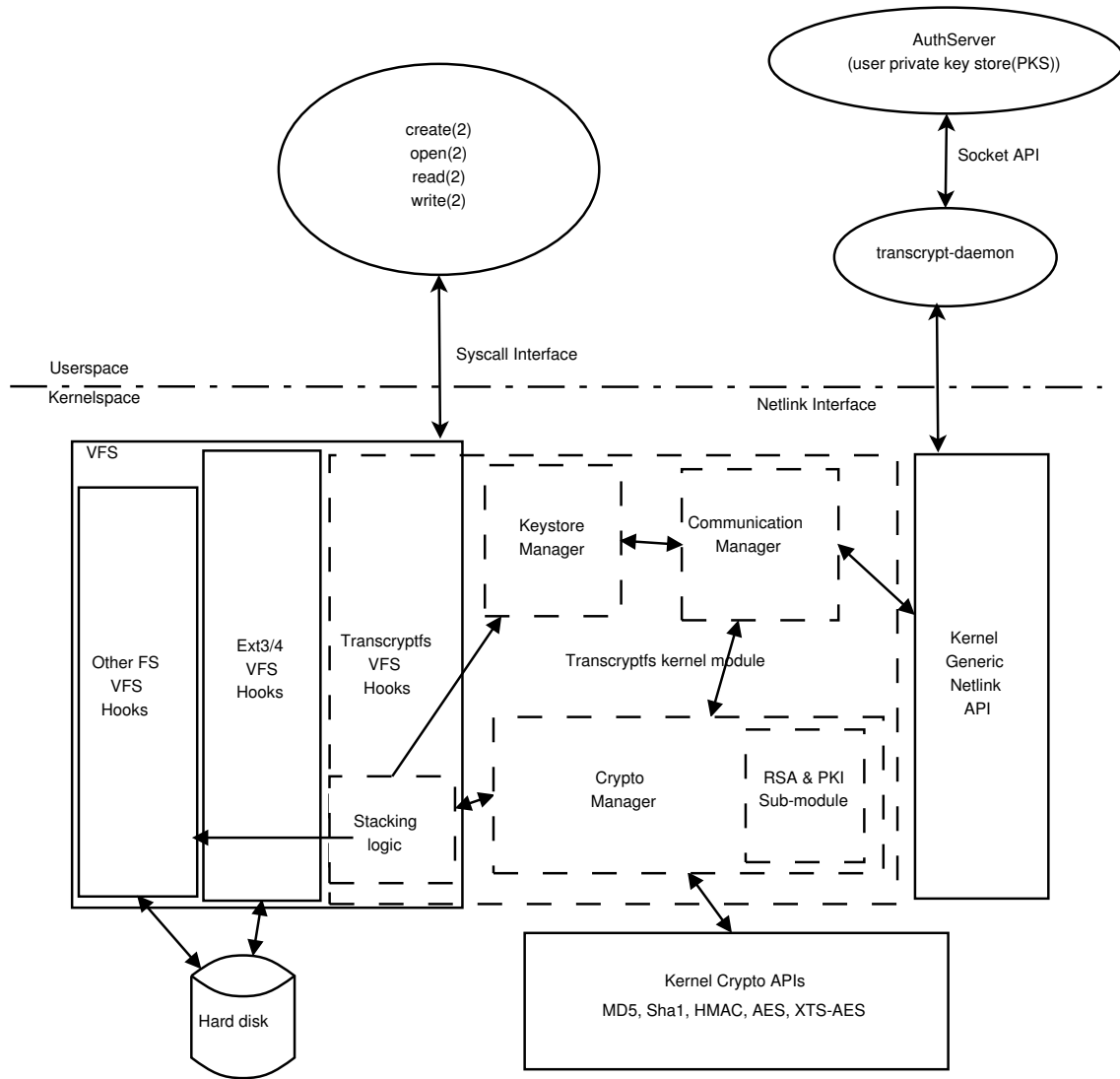


Figure 3.1: Transcriptfs Architecture

for filesystem related cryptographic operations and `crypt_stat` structure associated with every `inode` for file related cryptographic operations.

### 3.1.2 Communication Manager

Communication manager subsystem provides the interface between the kernelspace and userspace for communicating with userspace utilities. Communication manager implements a netlink interface through which messages are received and replied to from within the kernel space. The calls to userspace are implemented via completion objects[22], which are asynchronous and are executed in a separate kernel thread. Hence, they don't block the other parallel operations of Transcriptfs mod-

ule and provide good scalability when multiple files are operated on simultaneously. The netlink[15] interface is utilized for communicating all private key store (PKS) operations of the user.

### 3.1.3 Keystore Manager

Keystore manager subsystem is responsible for establishing session with the user's PKS and maintains a list of established sessions. Keystore manager also manages the users' public key certificates. It acquires users' public key certificates from the filesystem and caches them within the kernelspace for faster subsequent accesses.

Keystore manager also maintains the kernel keyrings[9], which stores the session keys with the userspace. Each session key uniquely identifies the userspace session that was setup at the time of user login. These session keys are available to programs that execute in userspace and are isolated from other userspace sessions. This unique constraint helps us associate symmetric encryption keys with every user session. All the messages sent from a specific userspace session to kernel via netlink interface uses these keys to encrypt communication and hence prevent masquerading attack.

### 3.1.4 Other supporting components

The Transcryptfs kernel module consists of other supporting components, which perform operations required for stacking logic of filesystem and implements various entry point for VFS operations, including open, read and write system calls[6]. The other functionalities of this subsystem include handling of directory entry (`dentry`) and `inode` for various directory and file operations on VFS layer[21].

## 3.2 User Space Components

User space components of Transcryptfs provide functionality complimenting the kernelspace for various operations involving the proper functioning of Transcryptfs, these components are discussed in detail in Sourav Khandelwal's Thesis[17]. They

are briefly explained here for sake of completeness.

### 3.2.1 AuthServer

AuthServer Component runs on a host that is physically connected to user's PKS. AuthServer negotiates the connection between user's PKS and Transcriptfsd daemon. It registers itself with Transcriptfsd to receive any messages related to private key operation. It establishes a secure session with Transcriptfsd to prevent the man-in-the-middle attack.

### 3.2.2 Transcriptfsd

Transcriptfsd is the userspace daemon that interacts with Transcriptfs kernel module via netlink[31] interface. This daemon runs on the same system as that of Transcriptfs kernel module. The main functionality of this component includes routing of kernel messages to corresponding users PKS via AuthServer and vice versa. This daemon listens on pre-determined port for incoming connection from AuthServer (UDP port 3002 in our current implementation).

### 3.2.3 Other support utilities

The userspace also consists of the following support utilities:

- *setfacl*

This tool allows user to modify ACL of a file in Transcriptfs volume.

- *mount.transcriptfs*

This tool is a mount helper which asks for passphrase to mount a Transcriptfs volume and pass it on to the mount system call as a base64[14] encoded value.

- *PAM Module*

This module initiates the session establishment between the user's PKS and filesystem kernel at the time of login. This module is part of the authentication

infrastructure under PAM[39].



## Chapter 4

# Metadata Organization and Key Management

Transcryptfs filesystem maintains a set of metadata associated with the encrypted data. This chapter lists and describes each of the metadata and its format. There are multiple approaches to storing metadata along with the file, we primarily explored two ways of doing so:

- As part of extended attributes of file
- As part of data of file in the lower encrypted filesystem

Earlier implementation of Transcryptfs used extended attributes to store file specific metadata[26], but we found a serious limitation in terms of maximum size of metadata that could be stored in extended attribute. The maximum size depends on type of filesystem and block size, for ext4 with 4KB block size, maximum data that could be stored in extended attributes is limited to 4KB, that translates to approximately 15 ACL entries per file. The other limitation was with various file archival utilities, which did not recognize the presence of extended attributes and hence produced a potential threat of non-recoverable data while archiving files, due to loss of metadata. To overcome these limitations, we chose to store the metadata as part of underlying filesystem's file data.

## 4.1 Filesystem Metadata File

Transcryptfs maintains a hidden metadata file in the root path of encrypted lower filesystem, with the name ‘.transcryptfs.fsk.hash’. This file contains a salt value, generated randomly at the time of filesystem creation, for use with PBKDF2[16] (20 bytes), followed by encrypted FSK (16 bytes) and MD5[28] hash of FSK (16 bytes) as represented in Figure 4.1. The metadata file is accessible only via underlying filesystem path and is hidden in the mounted partition to prevent accidental modification and corruption of metadata.

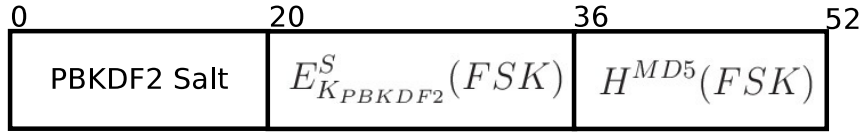


Figure 4.1: Filesystem Metadata Format

## 4.2 File System Key (FSK)

A secret key unique to a particular filesystem is maintained. This filesystem key is generated as part of filesystem creation process. FSK is derived by PBKDF2[16] function taking a passphrase from the administrator as the input. The generated FSK’s MD5[28] hash is stored in the filesystem metadata file as described in Section 4.1. On every mount operation, user is asked for passphrase and PBKDF2 is applied on passphrase to reconstruct FSK, to verify the validity; the MD5 hash is computed and matched with the earlier stored value in filesystem metadata file. The FSK is used as an encryption key for carrying out symmetric encryption while storing or retrieving token within file metadata.

## 4.3 File Encryption Key (FEK)

Every file is encrypted using the XTS-AES scheme which uses a symmetric key for encryption and decryption process. This key, of size 32 bytes, is generated at



the time of file creation using a random generator by Kernel which is identified as File Encryption Key (FEK). FEK is never stored in plaintext in the storage device. Instead, it only resides in plaintext in the kernel's memory space. FEK is used as symmetric key for encryption of data in XTS-AES[4] mode.

## 4.4 Access Control List (ACL)

Every file maintains an associated list of tokens for the purpose of access control restrictions. The token is generated by blinding FEK via FSK using symmetric encryption and further encrypted via asymmetric encryption using the current user's public key. The token is a per-user per-file based entry along with which user's UID and his public certificate's cert-id. It is stored as  $UID|cert-id|tokensize|E_U^A(E_{FSK}^S(FEK))$  in metadata of ACL is stored in the lower filesystem's file data.

## 4.5 In-Memory Key Management

Throughout the operation of Transcryptfs, various secret data is maintained within the kernel space memory. To allow easy access to this sensitive data within kernel, we augment the `crypto_stat` structure to the virtual `inode` structure of kernel. This allows us to maintain FEK associated with a particular file in its `inode` structure, hence providing a granular access mechanism to secret data. The filesystem specific key material is maintained in the super-block structure. Apart from these, we also maintain a list of established sessions with users' Private Key Store (PKS). The session list is a global structure within Transcryptfs kernel module and maintains session specific key material in `transcryptfs_session_key_list`. To perform the operations on this list, we cache it in `kmem` caches for faster access and manipulation.



# Chapter 5

## Implementation of Transcryptfs

### Kernel Module and Results

This chapter explains the internal working of Transcryptfs kernel module. We explain the control flow of various operations performed on a filesystem. All the operations described below are transparent to user space, i.e. there is no change in regular file operations that an application performs. The various operations that we will explain in following sections are:

- Mounting of Transcryptfs
- Opening of a File
- Read/Write from a File
- Adding an ACL entry

#### 5.1 Mounting of Transcryptfs

The mount operation is initiated from the `mount.transcryptfs` utility in userspace. The userspace utility asks for mount passphrase and passes it in a secure way as mount parameter to kernel. The kernel module defines a callback function for mount operation within Transcryptfs, which handles mounting operation.

The options passed as mount parameters, **pass** (passphrase) and **ca\_path**, are extracted. The AES encrypted passphrase is decrypted using session key. The un-encrypted passphrase is used as input to PBKDF2 function to derive key,  $K_{fsk}$ . The underlying filesystem's metadata is read to recover **encrypted\_FSK** and MD5 hash of FSK,  $H^{MD5}(FSK)$ . The FSK is calculated by performing  $E_{K_{fsk}}^S(encrypted\_FSK)$ . The validity of the FSK thus generated is ascertained by matching the MD5 hash of FSK with  $H^{MD5}(FSK)$ . The generated FSK and **ca\_path** are then stored in a kernel datastructure called **mount\_crypt\_stat**.

The superblock **inode** and **dentry** for the mounted path is registered with the kernel. Any file operations performed on this mounted path will then pass through the Transcryptfs kernel module. Further calls to decrypt file metadata (ACL tokens) requiring access to FSK will lookup **mount\_crypt\_stat**.

## 5.2 Opening of a File

Transcryptfs module registers with VFS callback for open system call. The callback function performs the following operations in sequence:

- Check the validity of session with userspace

On successful validity of session, the **session** structure is populated with the UID stored in session. The current user's UID is checked with session UID and on failure, access to file is denied.

- Setup **crypt\_stat** structure for file

**crypt\_stat** structure is associated with the **inode** of file and populated with cryptographic metadata associated with file. The underlying file is read for populating the metadata and checks are performed for existence of current user's ACL token( $E_U^A(E_{FSK}^S(FEK))$ ). The ACL token is sent through netlink interface to user's PKS for decrypting via the userspace as explained in Sourav Khandelwal's thesis[17]. The decrypted token, *blindedFEK* -  $E_{FSK}^S(FEK)$ , is then un-blinded using the FSK stored in **mount\_crypt\_stat**. FEK thus

obtained is stored within `crypt_stat` for future read/write operations on the file.

### 5.3 Read/Write from a File

The read and write operations of VFS layer operate directly on the page cache in main memory[30]. Transcryptfs does not perform encryption and decryption of data at this point. To achieve higher efficiency of cryptographic operations, we perform encryption and decryption at page level. When the main memory is being flushed to the disk, we perform page level encryption. Similarly, when the pages are being loaded into page cache, we perform decryption. To implement these operations, we register our callback in `mmap` layer of VFS. Any page cache operation would result in data passing through our cryptographic layer. These operations make use of FEK available via the `crypt_stat` structure associated with the file. The Crypto Manager (Section 3.1.1) implements XTS-AES operations which is optimized for page level cryptographic routines. Hence the read and write operations performed from the userspace are instantaneous. However, once the file is to be synced to storage, the page level cryptographic routines encrypt the data before being written onto the underlying filesystem.

### 5.4 Adding an ACL Entry

Transcryptfs stores the ACL entries as part of the underlying filesystem's file data. Hence, it provides custom implementation of *setfacl* tool in userspace. The owner of the file can use this tool to add an entry to ACL. The user passes the UID of the new user to be added as a parameter to this tool. This tool then communicates with the kernel module via the netlink interface and does the following operations:

- Verifies the access of current user

The session is verified and existence of current user's ACL entry is ascertained.

If the current user does not have access then operation fails.

- Extract *blindedFEK* of current user

The current user's token is decrypted and *blindedFEK* is recovered as  $E_{FSK}^S(FEK)$ .

- Construct ACL token for new user

Transcryptfs module looks up the public key certificate of new user from the `certlist`. On successful retrieval of public key, token is constructed by encrypting the *blindedFEK* as  $E_U^A(\textit{blindedFEK})$ .

- ACL token for new user is added to file's metadata

The ACL token is then added to the underlying filesystem's file as a metadata entry.

## 5.5 Performance Test Setup

To analyze the performance of Transcryptfs, we perform multiple read and write operations. The read operations are performed by reading a 512MB file from Transcryptfs volume and writing it to `/dev/null`. Since the write time to `/dev/null` is negligible it can be safely ignored. The write operation is performed by copying a 512MB file from an un-encrypted partition to Transcryptfs volume. Care is taken to drop page caches before and after every operation. To normalize the reading, we perform the same operation 10 times and ignore the maximum and minimum values, then take average. The same operations are performed with eCryptfs and ext4 (non-encrypting filesystem) for comparative analysis. These tests are performed on a virtual machine using User Mode Linux (UML)[8]. The virtual machine is configured with following specifications:

- **CPU:** Single core, 2.8Ghz
- **Memory:** 1GB Ram
- **OS kernel:** Linux 3.5.0-rc2

The results are presented in the following section.

## 5.6 Results

The results obtained are plotted in the following figures. Figure 5.1 describes the performance of write operations and Figure 5.2 describes the performance of read operations. Following sections interpret and reason the outcome of these tests.

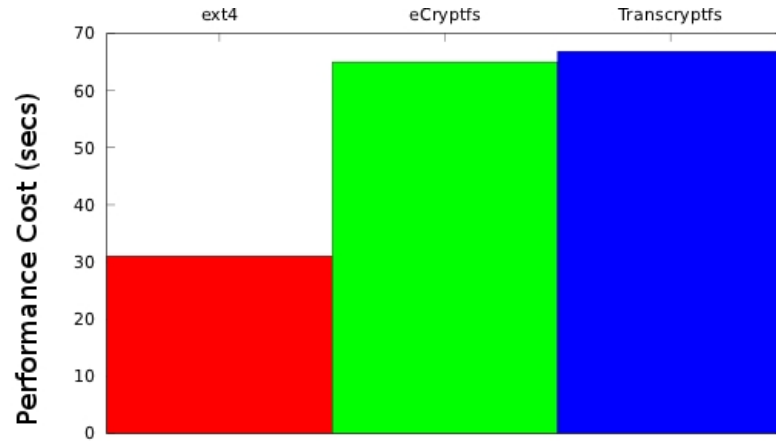


Figure 5.1: Write Performance

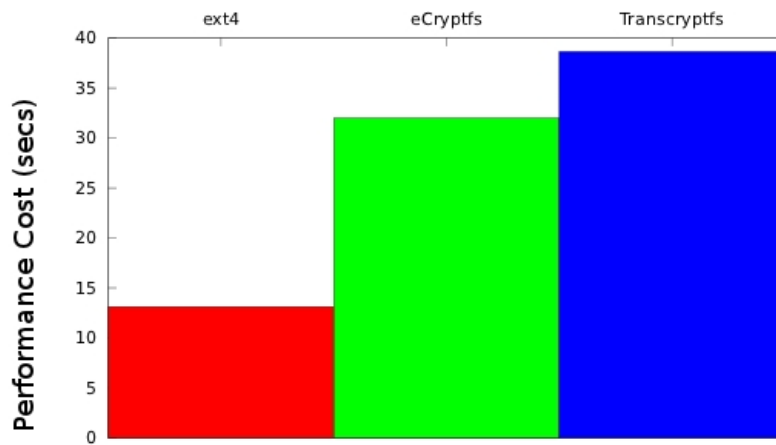


Figure 5.2: Read Performance

### 5.6.1 Write performance

The write performance is comparable to eCryptfs and costs marginally higher time. This marginal difference is due to Asymmetric key cryptography, which is significantly more CPU intensive. Compared to ext4, which is an un-encrypted filesystem, the write took about 100% extra time. The overhead incurred with unencrypted

filesystem is due to the cryptographic operations that we perform. This cost is an affordable price for security of data.

### 5.6.2 Read performance

The read performance is 20% higher than eCryptfs. This overhead is due to the delay incurred in userspace communication. The ACL token decryption happens over an asynchronous call to user's PKS store which is present in userspace. Compared to ext4 (un-encrypted filesystem), the read took about 200% extra time. The overhead can be reduced further when we use `kmem` cache for resulting *blindedFEK*, as noted in section 6.2.

### 5.6.3 Interpretation of Results

From the above performance metrics, we can conclude that Transcriptfs performs on par with existing cryptographic filesystem solutions in write operations. The additional overhead in read operations is justified by the stronger trust model offered by Transcriptfs.



# Chapter 6

## Conclusion and Future Work

### 6.1 Summary

Over the years, right from its inception, Transcriptfs has evolved to become better and be in sync with the evolving Linux kernel. Transcriptfs started out as a Proof-of-Concept at a time when there were no popular encrypting filesystems for Linux. The initial version was implemented as a modified ext3 filesystem by Satyam Sharma [34] in 2007, the initial key management was designed by Abhijit Bagri [3]. The initial version was further refined by decoupling metadata into extended attributes by Arun Raghavan [26], further work was carried out to make it accessible over network by Abhay Khoje[18] and Salih K A [1]. Further, with the work contributed by this thesis, we have been able to decouple the Transcriptfs from kernel's codebase and have successfully implemented it as a dynamically loadable kernel module, which is maintained as an out-of-the-tree kernel module, with dependency only on the callback APIs of VFS layer. We have been successful in making the installation process a lot easier without the complications of re-compilation of whole kernel, thereby lowering the barrier to easier adoption of Transcriptfs. With the new re-engineered code base, we have made Transcriptfs more modularized and decoupled from other subsystems of kernel, improving the maintainability of the code for years to come. To keep up with the latest technology in encrypted storage, we have also upgraded the encryption scheme to use XTS-AES[23][4], which is approved standard

for offline storage of encrypted data by NIST[25].

## 6.2 Future Work

The following section describes the possible future work within the kernel module. Some of these features also require support from userspace components which have been explained in Sourav Khandelwal’s thesis[17].

### 6.2.1 Better support for caching within kernel module

In the current implementation, we have `kmem` cache being utilized for most frequently used cryptographic metadata. To achieve higher performance, one can cache all the internal kernel data structures, which will increase the performance by not performing userspace interaction multiple times. We expect that use of such caching mechanism would make the performance of Transcryptfs significantly better than other encrypting filesystems.

### 6.2.2 Integration of data recovery process within kernel module

Currently the data recovery process is performed using standalone utilities. For better user experience and enterprise adoption, it is advisable to integrate the data recovery as part of core Transcryptfs module. This feature also requires support from userspace in form of utilities that can initiate data recovery process.

### 6.2.3 SSHFS support for network access

SSHFS is a `fuse` filesystem, that enables users to mount a remote filesystem locally over SSH interface[12]. Transcryptfs kernel module can potentially be extended to support SSHFS, since, there is minimal changes required on the filesystem server. But, for metadata operations on file, we need to expose interfaces from kernel module to allow userspace client utilities to interact over SSH interface.

# Appendix A

## User's Manual

This appendix lists the recommended system requirements to run Transcriptfs. In section A.2, we detail the step-by-step procedure to install Transcriptfs on a linux server.

### A.1 System Requirements

The following is the minium requirements to run a Transcriptfs filesystem:

- *Processor:* Any 32 bit or 64 bit based processor with recommend clock speed of 1.6Ghz.
- *OS:* Any Linux based kernel with minium version v3.5.0-rc2.
- *RAM:* minium of 1GB recommended.
- *Underlying filesystem:* ext4 filesystem is recommended for underlying filesystem.

### A.2 Installation

Download `transcriptfs.ko` module corresponding to your version of linux kernel. Download `transneofs-utils.tar.gz` and extract the binaries.

Copy `mount.transcryptfs` to `/sbin/`. Add the extracted path to `$PATH` environment variable. To activate the module execute:

```
modprobe transcryptfs
```

Once the module has been successfully loaded by above command, new Transcryptfs volume can be created by following command:

```
mount -t transcryptfs /path/to/lower/filesystem \  
      /path/to/mount/transcryptfs
```

File operations performed on `/path/to/mount/transcryptfs` is transparently encrypted onto underlying filesystem at `/path/to/lower/filesystem`. To unmount the filesystem execute the following command:

```
umount /path/to/mount/transcryptfs
```

After all the Transcryptfs volumes are unmounted, Transcryptfs kernel module can optionally be removed by executing:

```
modprobe -r transcryptfs
```

# Bibliography

- [1] Salih K A. TransCrypt File Server Enhancements for Secure Remote Access. Master's thesis, Indian Institute of Technology, Kanpur, Jul 2009.
- [2] Rohit Kumar Agrawal. Implementation of Network File System Client Library for Microsoft Windows. Master's thesis, Indian Institute of Technology, Kanpur, Jun 2010.
- [3] Abhijit Bagri. Key Management for Transcript. Master's thesis, Indian Institute of Technology, Kanpur, May 2007.
- [4] Matthew V. Ball, Cyril Guyot, James P. Hughes, Luther Martin, and Landon Curt Noll. The xts-aes disk encryption algorithm and the security of ciphertext stealing. *Cryptologia*, 36(1):70–79, 2012.
- [5] Kapil Bhadke. WinTransCrypt: A Secure Encrypting File System for Microsoft Windows. Master's thesis, Indian Institute of Technology, Kanpur, Aug 2011.
- [6] Daniel P. Bovet & Marco Cesati. *Understanding the Linux Kernel*. O'Reilly Media, third edition, 2005.
- [7] Jean-Luc Cooke and David Bryson. Strong Cryptography in the Linux Kernel. In *Proceedings of the Linux Symposium*, pages 139–144, Ottawa, Canada, Jul 2003.
- [8] Jeff Dike. A user-mode port of the linux kernel. In *Proceedings of the 2000 Linux Showcase and Conference*, volume 2, pages 2–1, 2000.
- [9] Jake Edge. Kernel Key Management, Nov 2006. URL <http://lwn.net/Articles/210502/>.
- [10] Michael Austin Halcrow. eCryptfs: An Enterprise-class Encrypted Filesystem for Linux. In *Proceedings of the Linux Symposium*, pages 201–218, Ottawa, Canada, July 2005.
- [11] John S Heidemann and Gerald J Popek. File-system development with stackable layers. *ACM Transactions on Computer Systems (TOCS)*, 12(1):58–89, 1994.
- [12] Matthew E Hoskins. Sshfs: super easy file access over ssh. *Linux Journal*, 2006 (146):4, 2006.
- [13] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC 3280). 2002.

- [14] S. Josefsson. The Base16, Base32, and Base64 Data Encodings (RFC4648), Oct 2006. IETF.
- [15] Kevin Kaichuan. Kernel Korner - Why and How to Use Netlink Socket, Jan 2005. URL <http://www.linuxjournal.com/article/7356/>. [accessed 18 Sept-2012].
- [16] Kaliski, B. RFC 2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0, Jun 2000.
- [17] Sourav Khandelwal. TranscryptFS : User Space Support. Master's thesis, Indian Institute of Technology, Kanpur, Sept 2012.
- [18] Abhay Khoje. Framework for accessing TransCrypt File System over untrusted network. Master's thesis, Indian Institute of Technology, Kanpur, Jul 2009.
- [19] Matthew Komorowski. A History of Storage Cost@ONLINE, May 2013. URL <http://www.mkomo.com/cost-per-gigabyte>. [accessed 29 May 2013].
- [20] RSA Laboratories. Public-Key Cryptography Standards (PKCS) @ONLINE, October 2009. URL <http://www.rsa.com/rsalabs/node.asp?id=2124>. [accessed 18 Sept-2012].
- [21] Robert Love. *Linux Kernel Development*. Pearson Education, third edition, 2010.
- [22] LWN. Driver porting: completion events @ONLINE, February 2003. URL <http://lwn.net/Articles/23993/>. [accessed 18 Sept-2012].
- [23] Luther Martin. Xts: A mode of aes for encrypting hard disks. *IEEE Security and Privacy*, 8:68–69, 2010.
- [24] Dharmendra Modi. Design of a Secure Microsoft Windows based Distributed File System. Master's thesis, Indian Institute of Technology, Kanpur, Jun 2010.
- [25] SP NIST. 800-38e: Recommendation for block cipher modes of operations: The xts-aes mode for confidentiality on storage devices. *NIST Special Publication*.
- [26] Arun Ragahvan. File System Independent Metadata Organization for TransCrypt. Master's thesis, Indian Institute of Technology, Kanpur, Jun 2008.
- [27] Dhirendrakumar Ram. User-space Support for the WinTransCrypt Encrypting File System for Microsoft Windows. Master's thesis, Indian Institute of Technology, Kanpur, May 2012.
- [28] R. Rivest. The MD5 Message-Digest Algorithm (RFC 1321). April 1992. MIT.
- [29] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/357980.358017>.
- [30] Alessandro Rubini. The Virtual File System in Linux, May 1997. URL <http://www.linuxjournal.com/article/2108>.

- [31] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. Linux Netlink as an IP Services Protocol (RFC3549), Jul 2003.
- [32] Peter Jay Salzman, Michael Burian, and Ori Pomerantz. The Linux Kernel Module Programming Guide, May 2007. URL <http://tldp.org/LDP/lkmpg/2.6/html/index.html>.
- [33] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11): 612–613, 1979.
- [34] Satyam Sharma. TransCrypt: Design of a Secure and Transparent Encrypting File System. Master’s thesis, Indian Institute of Technology, Kanpur, May 2007.
- [35] Satyam Sharma, Rajat Moona, and Dheeraj Sanghi. TransCrypt: A Secure and Transparent Encrypting File System for Enterprises. In *8th International Symposium on System and Information Security*, 2006.
- [36] PolarSSL Team. PolarSSL library - Crypto and SSL made easy @ONLINE, 2012. URL <http://polarssl.org/>. [accessed 18 Sept-2012].
- [37] TrueCrypt Team. TrueCrypt - Free open-source disk encryption software for Windows Vista/XP, Mac OS X, and Linux, Sept 2012. URL <http://www.truecrypt.org/>. [accessed 18 Sept-2012].
- [38] Sainath S Vellal. A Device Mapper based Encryption Layer for TransCrypt. Master’s thesis, Indian Institute of Technology, Kanpur, Jun 2008.
- [39] Wikipedia. Pluggable authentication module @ONLINE, July 2012. URL [http://en.wikipedia.org/wiki/Pluggable\\_authentication\\_module](http://en.wikipedia.org/wiki/Pluggable_authentication_module). [accessed 18 Sept-2012].
- [40] Wikipedia. Public-Key Cryptography @ONLINE, 2012. URL [http://en.wikipedia.org/wiki/Public-key\\_cryptography](http://en.wikipedia.org/wiki/Public-key_cryptography). [accessed 18 Sept-2012].
- [41] Wikipedia. Loadable Kernel Module @ONLINE, Sept 2012. URL [http://en.wikipedia.org/wiki/Loadable\\_kernel\\_module](http://en.wikipedia.org/wiki/Loadable_kernel_module). [accessed 18 Sept-2012].
- [42] Erez Zadok and Jason Nieh. Fist: A language for stackable file systems. In *Proceedings of the Annual USENIX Technical Conference*, pages 55–70, 2000.
- [43] Erez Zadok, Ion Badulescu, and Alex Shender. Cryptfs: A stackable vnode level encryption file system. Technical report, Citeseer, 1998.
- [44] Erez Zadok, Ion Badulescu, and Alex Shender. Extending file systems using stackable templates. In *Proceedings of the Annual USENIX Technical Conference*, pages 57–70, 1999.