I. J. Computer Network and Information Security, **2012, 8, 33-39**
Published Online August 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2012.08.04

Distributed Encrypting File System for Linux in User-space

U. S. Rawat, Shishir Kumar
Department of Computer Science & Engineering, Jaypee University of Engineering & Technology, Guna (MP), India
umasrawat@gmail.com, dr.shishir@yahoo.com

***Abstract* —** Linux systems use Encrypting File System (EFS) for providing confidentiality and integrity services to files stored on disk in a secure, efficient and transparent manner. Distributed encrypting file system should also provide support for secure remote access, multiuser file sharing, possible use by non-privileged users, portability, incremental backups etc. Existing kernel-space EFS designed at file system level provides all necessary features, but they are not portable and cannot be mounted by non-privileged users. Existing user-space EFS have performance limitations and does not provide support for file sharing.

Through this paper, modifications in the design and implementation of two existing user-space EFS, for performance gain and file sharing support, has been presented. Performance gain has been achieved in both the proposed approaches using fast and modern ciphers. File sharing support in proposed approaches has been provided with Public Key Infrastructure (PKI) integration using GnuPG PKI module and Linux Pluggable Authentication Module (PAM) framework. Cryptographic metadata is being stored as extended attributes in file's Access Control List (ACL) to make file sharing task easier and seamless to the end user.

Index Terms — Encrypting File System (EFS), File System in User-space (FUSE), Network File System (NFS), Public-Key Infrastructure (PKI), Access Control List (ACL), Pluggable Authentication Module (PAM)

I. INTRODUCTION

While considering file system security, several aspects should be taken into account such as authentication, authorization, access control, confidentiality and integrity. Linux systems provides authentication, authorization and access control services using Pluggable Authentication Module (PAM) [1] ; policy language that defines file owner and group, along with the owner/group/world read/write/execute attributes of the file; Posix Access Control Lists (ACL's) [2] that provides more stringent access control on a per-file basis etc. For confidentiality and integrity services, Encrypting File System (EFS) have to be used that provides file encryption/decryption along with integrity mechanisms, in a secure, efficient and transparent manner to the user. Distributed encrypting file system should also provide secure remote access over Network File System (NFS), file sharing among multiple users, possible use by non-privileged users, portability, incremental backups etc.

Encryption services by encrypting file systems can be placed at file system level or device layer level. In device layer systems like dmCrypt [3] and cryptsetup [4], encryption/decryption takes place at device layer in kernel-space, using Linux kernel device mapper infrastructure that provides a generic way to create virtual layers of block devices. These systems perform encryption with a single key on entire block device, so file sharing is not possible among multiple users. They are also not convenient for incremental back-ups, cannot be mounted by non-privileged users and cannot be used remotely over NFS.

At file system level, EFS can be implemented either in user-space or in kernel-space. eCryptfs [5] is the most popular kernel-space EFS, integrated with the Linux kernel since 2.6.19. It uses stackable file system interface approach [6] to introduce a layer of encryption that can fit over any underlying file system. eCryptfs has been implemented using File System Translator (FiST) [7], a tool that can be used to develop stackable file systems using template code. eCryptfs is more efficient than existing user-space encrypting file systems, discussed subsequently. It performs encryption on a per-file basis and provides support for file sharing among multiple users using Public Key Infrastructure (PKI) support. It also provides support for file integrity using keyed hashes. It can be used remotely on top of networked file systems. The limitations of eCryptfs are that, it cannot be ported across different platforms and do not provide any options for non-privileged users to mount a file system.

Existing user-space EFS like CFS [8] and EncFS [9] are implemented using NFS approach and File System in User-space (FUSE) [10] respectively. CFS is implemented entirely in user-space as a modified NFS server. A userspace daemon, cfsd, acts as a pseudo-NFS server, and NFS client in the kernel makes RPC calls to the daemon. The CFS daemon performs transparent encryption/decryption of file contents during write and read operations.CFS can be mounted by any user on the system and does not require any modifications to the kernel so can be easily portable. CFS is capable of acting as a remote NFS server, so it can be accessed remotely without requiring an additional NFS mount. The limitation of CFS is its poor performance due to frequent context switches and data copies between user-space and kernel-space. Also, it uses DES algorithm for file encryption/ decryption, which further degrades its performance. EncFS [9] is another popular user-space EFS for Linux, written using FUSE library. FUSE has been integrated into the Linux kernel tree and provides a good way to write virtual file systems. FUSE exports all file system calls within the kernel to the user-space through a simple application programming interface (API) by connecting to a daemon that is running in the user-space. In EncFS, this user-space daemon has been modified to perform transparent encryption and decryption of file contents during write and read system calls respectively. EncFS is portable as FUSE has ports available for other major operating systems. EncFS also has provisions to permit non-privileged users to mount the file system. FUSE provides an efficient userspace-kernel interface, so performance of EncFS is somewhat better than CFS. EncFS can be used remotely, mounted on top of NFS. It also provides support for file integrity using keyed hashes. Both CFS and EncFS perform encryption with a single key on entire directory, so sharing of files is not possible among different users.

As mentioned above, performance, file sharing, portability and availability to non-privileged users, all cannot be achieved together. Existing user-space EFS have performance limitations and does not provide support for file sharing; and kernel-space EFS are not portable and cannot be mounted by non-privileged users.

Through this work, design and implementation of user-space EFS using two approaches: one based on CFS and another based on EncFS, with performance improvements and file sharing support, has been presented. In modified CFS approach, Blowfish algorithm has been used instead of DES for improving performance. Blowfish gives high performance that DES, Triple DES and AES ciphers [11]. In modified EncFS approach, XTS (XEX-based Tweaked codebook mode with ciphertext Stealing) mode of the AES algorithm [12, 13] has been used for performance gain. EncFS uses CBC (Cipher Block Chaining) mode for file encryption with keyed hashes, like HMAC, for file integrity. In modified EncFS, XTS-AES itself provides more protection than CBC-AES against unauthorized manipulation of the encrypted data, thus curtails the need for separate integrity mechanism [14, 15]. Thus, XTS-AES mode is suitable choice for encrypting data stored on hard disks where there is not additional space for an integrity field. It also provides random access to encrypted data. It can also be implemented as parallel algorithm. Parallel implementation of XTS-AES algorithm is 90 % more efficient than the serial algorithm [16].

In both the proposed approaches, file sharing support is being provided by PKI integration and performing encryption on a per-file basis with storing cryptographic metadata as extended attributes in file's ACL.

The rest of this paper is organized as follows. Section II presents the modified CFS and modified EncFS architectures with PKI integration for file sharing support. Section III and section IV explains the cryptographic operations taking place in modified CFS and EncFS respectively. Section V describes the implementation of proposed designs. Section VI provides performance comparison of both proposed approaches with existing user-space and kernel-space encrypting file systems. Section VII concludes the paper with identified future work.

II. Modified Architectures of CFS and EncFS

Modification made in the existing architectures of CFS and EncFS, for providing support for file sharing, have been shown as dotted portions in Fig. 1 and Fig. 2, respectively. Existing CFS and EncFS perform encryption of entire directory contents with a single key, with storing cryptographic metadata in special files in that directory, so file sharing is not possible in these systems. For multiuser file sharing support, modified CFS and modified EncFS performs encryption of each file with a different File Encryption Key (FEK) that itself is encrypted with the public keys of the users who are authorized to access that file. Public key cryptographic support is being provided by Pluggable Authentication Module (PAM) [1] and GnuPG PKI module [17]. PAM provides a discretionary access control mechanism whereby superuser can parameterize how a user is authenticated and what happens at the time authentication. In both modified approaches, PAM captures the user's login passphrase and stores it in the session keyring. GnuPG PKI module has been used to access the user's GnuPG keyring. GnuPG keyring stores public key and private key pair corresponding to all the users on the system. GnuPG PKI module utilizes the user's login passphrase stored in user's session keyring to decrypt and access the user's private key stored on the GnuPG keyring. It provides the user's private key and public key to the CFS daemon and EncFS daemon when the user logged in.

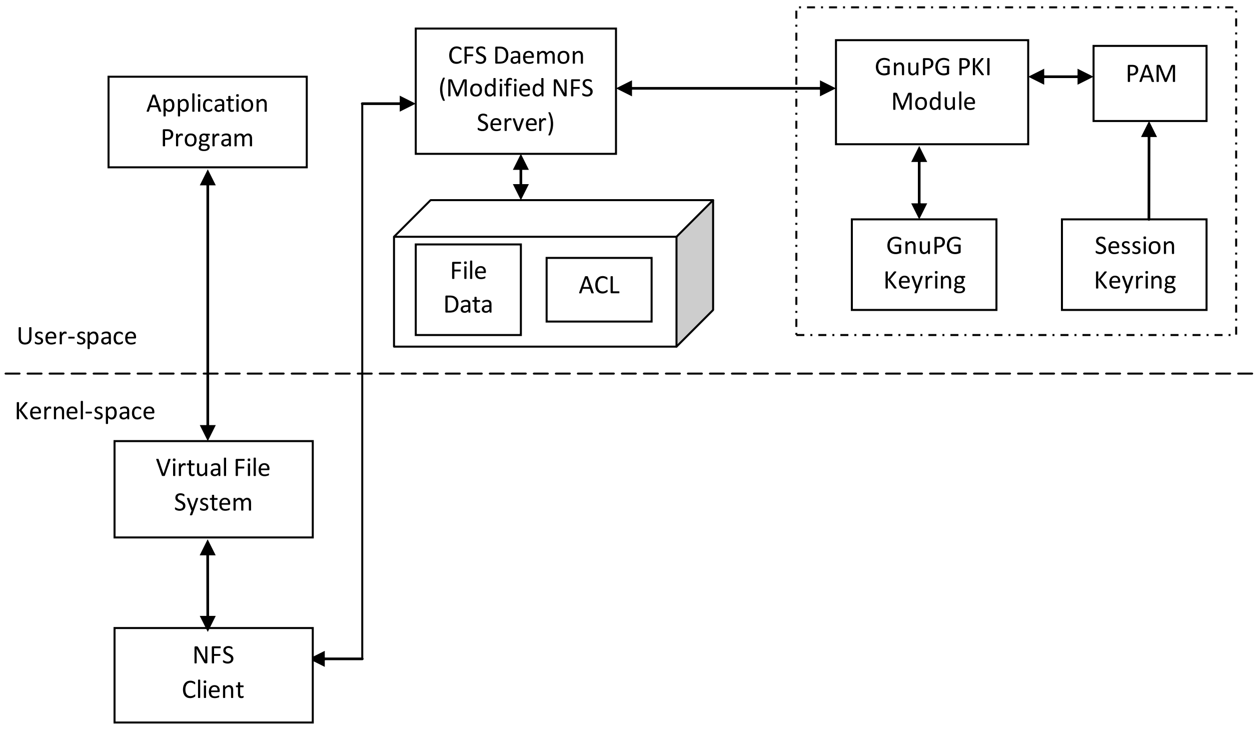

Figure 1. Modified CFS Architecture

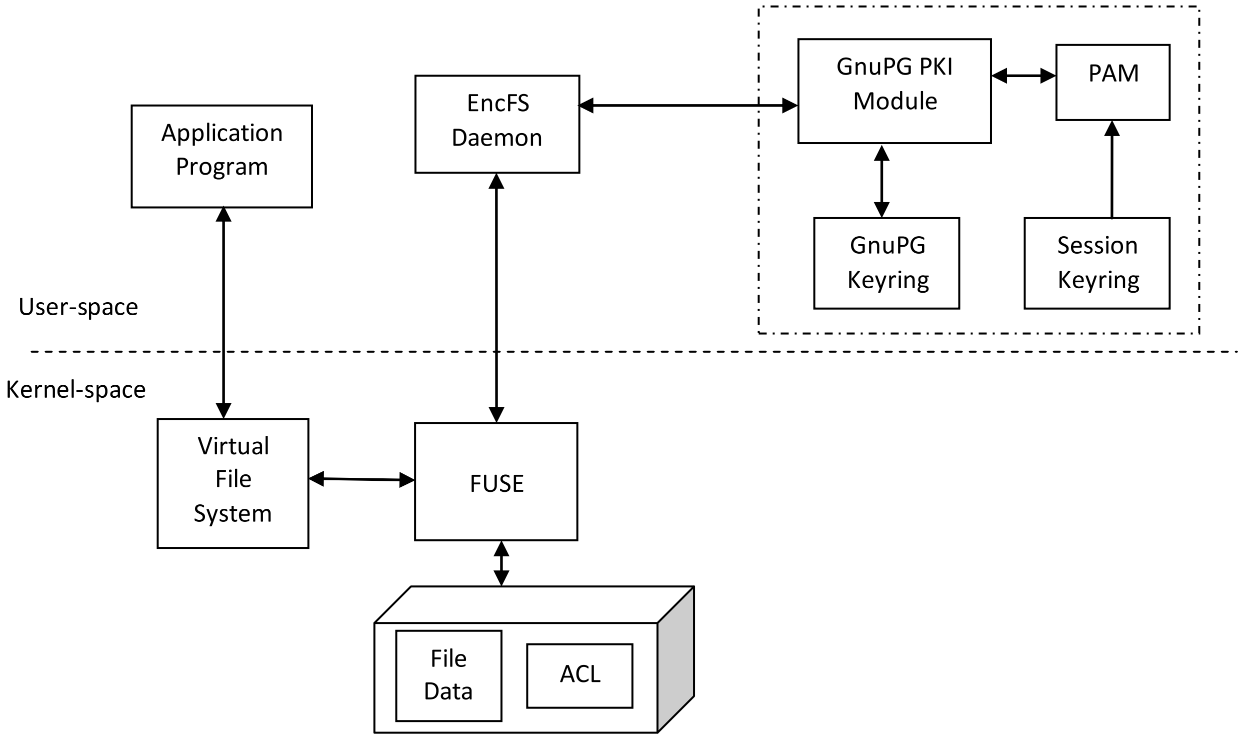

Figure 2. Modified EncFS Architecture

In both the proposed approaches, cryptographic metadata (FEK encrypted with user's public key) is stored as extended attributes in file ACL to ease the file sharing task, detailed explanation in section III. In addition to ease of file sharing, there are several other benefits of per-file encryption with storing cryptographic metadata as extended attributes of a file [18]:

Performance: In general, only certain files need to be encrypted. Encryption is not required for files like system libraries and executables. By limiting the encryption to only those files that really need it, system performance can be improved.

Backup utilities: Incremental backup utilities can operate without having access to the decrypted content of the files.

Transparent operation: Individual encrypted files can be easily transferred from the block device without any extra transformation, and others with authorization will be able to decrypt those files.

III. Cryptographic Operations in Modified CFS

When a new file is created, the CFS daemon generates a new 128-bit symmetric File Encryption Key (FEK) for the encryption of the file contents using Blowfish algorithm. The owner of the file is automatically authorized to access the file, and so the FEK is encrypted with the public key of the owner of the file (K_{owner}), which was passed to CFS daemon at the time when the user logged in by GnuPG PKI module, as mentioned in section II. The encrypted FEK is then stored in the file ACL along with other user information. An ACL entry will look like as:

UID: username: permissions: E_{Kowner} *(FEK)*

Suppose that the owner at this point wants to grant access to the file to another user whose public key, K_{user}, is in the GnuPG keyring. GnuPG PKI module extracts K_{user} from the GnuPG keyring, and passes it to the CFS daemon. CFS daemon now encrypts FEK with K_{user} and adds to the extended attribute set of the file. The ACL for the file now contains two entries, with two copies of FEK encrypted with different public keys: E_{Kowner} *(FEK)* and E_{Kuser} *(FEK)* along with other user information.

When a user opens an existing file, CFS daemon extracts the encrypted FEK from the file ACL, decrypts it using private key of the user and then decrypts the file contents using FEK. GnuPG PKI module provides the private key to the CFS daemon after retrieving it from GnuPG keyring when the user logged in, as already discussed in section II.

Note that this is not an access control directive; it is rather a confidentiality enforcement mechanism that extends beyond the Linux access control policy, based on file permissions and/or ACL's. Without authorized user's private key, no entity will be able to access the decrypted contents of the file. Linux access control policies will act over the decrypted file.

IV. Cryptographic operations in Modified EncFS

When a new file is created in modified EncFS, EncFS daemon generates a new 128-bit File Encryption Key (FEK) and 64-bit per-file tweak (TWK) that is used to carry out encryption of file contents using XTS-AES algorithm. XTS-AES allows for random access and encrypts file to the same length as their original size. XTS tweak of 128-bit is formed by concatenating 64-bit per-file tweak value (TWK) with 64-bit file offset. The idea behind using a tweak is to get unique, per-file ciphertext, thus protects from chosen ciphertext attack and copy-paste attack. If file size is not an integer multiple of the cipher block size, the XTS construction uses ciphertext stealing. Ciphertext stealing is a technique that can be used to encrypt data that does not comprise an integer multiple of the cipher's block size. XTS performs ciphertext stealing by combining the last two blocks of ciphertext. Sparse files are detected by examining if all 4096 bytes in a sector are zero before decryption, they will not be decrypted rather zero-filled sector is returned.

The public key cryptographic operations for file sharing take place in the similar way as discussed in section III. Here, EncFS daemon encrypts 128-bit FEK in concatenation with 64-bit TWK (FEK || TWK) with public key of the owner, and stores it in file ACL at the time of file creation. When a user wants to access a file, EncFS daemon extracts the encrypted FEK|| TWK from the ACL, decrypts it using private key of the user and then decrypts the file contents.

V. Implementation

In modified CFS, CFS daemon is extended to perform encryption of each file with a new File Encryption Key (FEK) using blowfish cipher; and encryption/decryption of FEK with public key/private key of the user using RSA algorithm. CFS daemon is also extended to extract and store encrypted FEK in file's ACL.

In place of DES, choice of blowfish cipher instead of AES has been made in modified CFS due to following two reasons. First, blowfish has high performance than AES, already mentioned in section I. Second, CFS can only be used with a 64-bit block cipher and AES uses fixed block size of 128 bits.

Standard implementation of blowfish and RSA ciphers has been added by providing a block encrypt/decrypt function and key encrypt/decrypt function respectively; and adding hooks for them in the following routines: *cipher(), mask_cipher(), pwcrunch(),*and *copykey().* These routines can be found in *cfs_cipher.c* and *getpass.c* in standard CFS implementation [19]. Modification has also been made in *cmkdir.c, cname.c, ccat.c, admproto.x,* and *cfs.h* files to refer to the new cipher, as well as references has been added in various places in the Makefile to refer the new cipher module.

Fig. 3 shows the various components of EncFS daemon [9]. Modifications have been made in various routines in files of *libencfs* component to perform to perform encryption of each file with a new file encryption key (FEK) using AES cipher in XTS mode; and encryption/decryption of FEK with public key/private key of the user using RSA algorithm. OpenSSL FIPS object module 2.0 [20] has been used as it provides support for XTS mode of AES in addition to RSA algorithm and key generation.

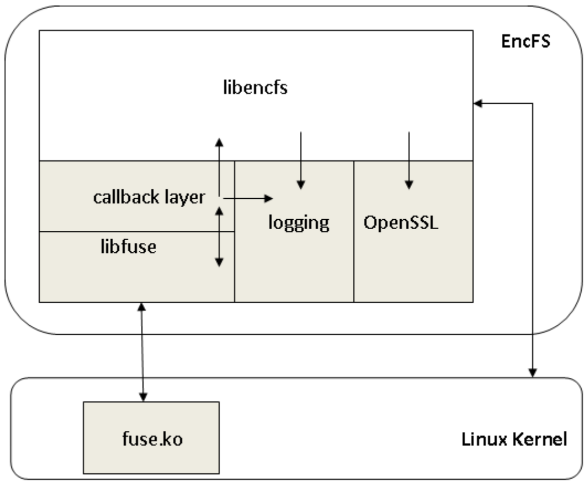

Figure 3. EncFS Components

In both the modified approaches, Access Control List (ACL) for storing encrypted FEK is implemented using extended attributes [2] inside the kernel data structures. The posix*_acl_entry* and *xattr_ acl_entry* structures, that define the kernel's representation of ACL entries, have been augmented with the 'encrypted FEK' field. On-disk format of *ext4* ACL entries, defined in the *ext4_acl_ entry* and *ext4_ acl_ entry_ short* structures have also been augmented with 'encrypted FEK' field. Suitable changes have also been made to various ACL manipulation functions.

VI. Performance

In this section, performance evaluation of modified CFS and modified EncFS with CFS [8], EncFS [9], eCryptfs [5] will be presented. Performance of these encrypting file systems and unencrypted ext4 file system, have been evaluated by running IOZone [21], a popular benchmarking tool that performs synthetic read/write tests to determine the throughput of the system. Tests have been conducted on a 3 GHz Intel core i-3 machine with 2GB RAM running Linux kernel 2.6.34.

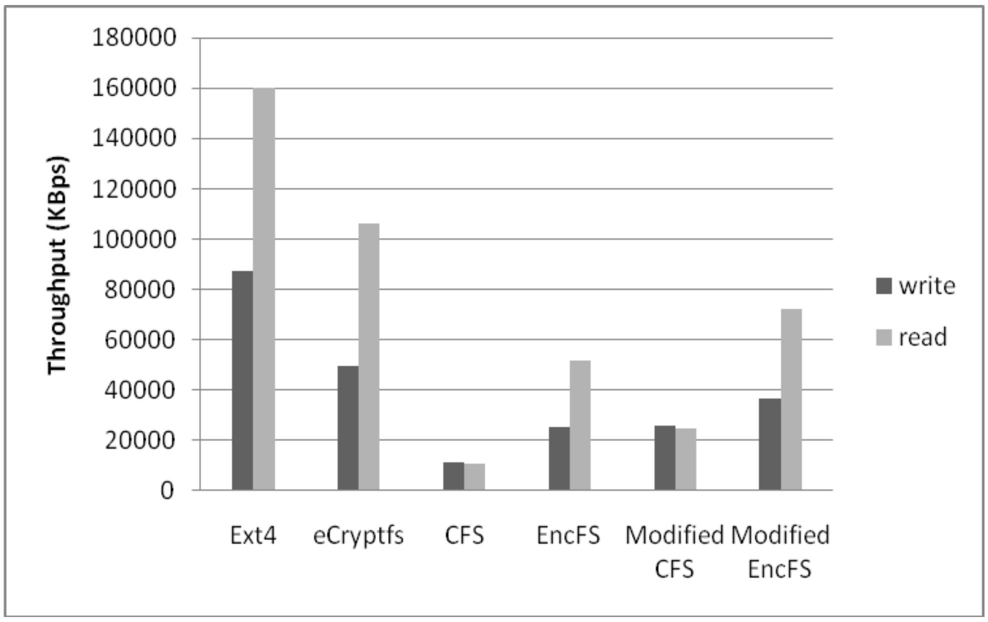

Figure 4. Throughput obtained for different file systems using IOZone benchmarking tool

Fig. 4 shows throughput obtained in Kbytes/sec for read and write operation on different file systems, by running IOZone on different file systems mount points, choosing file size of 1 GB with record size set to 128 KB. The *iozone* utility has been used with *–t* option for obtaining the throughput like:

# iozone -t 1 -i 0 -i 1 -s 1024m -r 128k –F ./f1 -b /home/output.xls

where *–i* option specifies the read and write tests (0 and 1 respectively); *-s* option specifies file size; *-r* option specifies record size; *-F* option specifies the path for creating temporary test file; and *–b* option specifies the output file in which obtained throughput will be stored after successful execution of the command. The path of file *f1* should be given inside a particular file system mount point, for which throughput has to be obtained.

Performance overhead is calculated for different encrypting file systems with respect to unencrypted Ext4 file system, and is being reported in Table I.

Performance overhead is more in case of CFS and EncFS as compared with eCryptfs, as both are implemented in user-space, so they have to perform many context switches and data copies between kernel-space and user-space as mentioned in section I.

TABLE I. PERFORMANCE OVERHEAD IN DIFFERENT EFS WITH RESPECT TO UNENCRYPTED EXT4 FILE SYSTEM

File System	% overhead in write operation w.r.t. ext4	% overhead in read operation w.r.t. ext4
eCryptfs	43 %	34 %
CFS	87 %	93 %
EncFS	71 %	68 %
Modified CFS	70 %	84 %
Modified EncFS	58 %	55 %

Performance overhead in modified CFS has been reduced significantly with the use of blowfish cipher. Performance gain of nearly 131 % for write and 136 % for read operation has been obtained when compared with CFS. Besides that, as shown in Table I, performance overhead in modified CFS is much more as compared with unencrypted Ext4 local file system, which makes concerns about its use as an encrypting file system. However, as mentioned in section I, it itself can act as a remote NFS server, so it is a better choice for remote file access than other encrypting file systems. Other encrypting file systems need to be mounted on NFS for remote file access, and then their performance degrades.

Performance gain of around 44 % for write and 39 % for read operation has been observed in modified EncFS as compared with EncFS. When compared with eCryptfs, performance overhead is around 50 % for both read operation and write operation in EncFS, which has been reduced to almost 26 % for write operation and 32 % for read operation in modified EncFS. This may be attributed to the reason, that both eCryptfs and EncFS uses keyed hashes, like HMAC, for file integrity, rather modified EncFS does not use any separate mechanism for integrity. Modified EncFS uses XTS mode that itself protects it from unauthorized manipulation of data, as discussed in section I.

VII. CONCLUSION AND FUTURE WORK

Existing implementations of CFS and EncFS have been modified for providing better performance and support for file sharing. Significant performance gain has been observed in modified CFS and modified EncFS implementations with the use of blowfish and XTS-AES ciphers respectively. Performance of modified EncFS is a bit inferior with respect to kernel-space encrypting file system (eCryptfs), although there are major benefits of implementation in user-space like portability, and possible mounting of file system by non-privileged users. File sharing support in both proposed approaches has been provided by PKI integration using GnuPG PKI module and performing encryption on a per-file basis with storing cryptographic metadata as extended attributes in file's ACL.

In future, modified EncFS can be implemented using parallel implementations of XTS-AES for further improving its performance. Hardware devices such as smart cards or USB connected disks can be used in modified implementations for storing the user's private key using openCryptoki PKCS#11 public-key infrastructure [22] support.

REFERENCES

[1] Andrew G. Morgan, "Linux Pluggable Authentication Module," http://www.kernel.org/ pub/linux/libs/pam.

[2] A. Grunbacher, "POSIX Access Control Lists on Linux," Proceedings of the USENIX Annual Technical Conference (ATC), FREENIX Track, San Antonio, Texas, USA, June 2003, pp. 259–272.

[3] "DMCrypt: Linux Kernel Device-Mapper Crypto Target," http://code.google.com/p/cryptsetup/wiki/DMCrypt.

[4] "Cryptsetup - Setup Virtual Encryption Devices under dm-crypt Linux," http://code.google.com/p/cryptsetup.

[5] M.A. Halcrow, "eCryptfs: An Enterprise-class Cryptographic Filesystem for Linux," Proceedings of the Linux Symposium, Ottawa, Canada, July 2005, pp. 201–218.

[6] E. Zadok, I. Badulescu, "A Stackable File System Interface for Linux," LinuxExpo, Raleigh, North Carolina, May 1999, pp. 141–151.

[7] E. Zadok, J. Nieh, "FiST: A Language for Stackable File Systems," Proceedings of the USENIX Annual Technical Conference (ATC), San Diego, CA, USA, June 2000, pp. 55–70.

[8] Matt Blaze, "A Cryptographic File System for UNIX," Proceedings of the ACM Conference on Computer and Communications Security (CCS), Fairfax, VA, USA, November 1993, pp. 9–16.

[9] Valient Gough, "EncFS Encrypted Filesystem Source Code," http://encfs.googlecode.com/files/encfs-1.7.4.tgz.

[10] Miklos Szeredi, "Filesystem in Userspace," 2012. http://sourceforge.net/projects/fuse/files/fuse-2.X.

[11] A. A. Tamimi, "Performance Analysis of Data Encryption Algorithms," Project Report, Washington University, St. Louis, USA, 2006.

[12] IEEE Standard 1619–2007, "The XTS-AES Tweakable Block Cipher," Institute of Electrical and Electronics Engineers, Inc., 2008.

[13] M. Dworkin, ''Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices,'' NIST SP 800-38E, 2009.

[14] M. Liskov, K. Minematsu, "Comments on XTS-AES", 2008.http://csrc.nist.gov/groups/ST/toolkit/BCM/doc

uments/comments/XTS/XTS_comments-Liskov_Minematsu.pdf

[15] Matthew V. Ball, Cyril Guyot, James P. Hughes, Luther Martin & Landon Curt Noll, "The XTS-AES Disk Encryption Algorithm and the Security of Ciphertext Stealing," Cryptologia, vol. 36, no. 1, pp. 70-79, January 2012.

[16] M.A. Alomari, K. Samsudin, A.R.Ramli, "A Parallel XTS Encryption Mode of Operation," IEEE Student Conference on Reseach and Development (SCOReD), UPM Serdang, Malaysia, November 2009, pp. 172-175.

[17] "GnuPG PKI Module," http://www.gnupg.org.

[18] M. A. Halcrow, "Demands, Solutions, and Improvements for Linux Filesystem Security," Proceedings of the Linux Symposium, Ottawa, Canada, July 2004, pp. 269–286.

[19] Matt Blaze, "CFS Encrypting File System Source Code," http://www.crypto.com/software/.

[20] "OpenSSL FIPS Object Module v2.0 Source Code," 2012. http://www.openssl.org/source/ openssl-fips-2.0.tar.gz.

[21] "IOZone," http://www.iozone.org.

[22] "OpenCryptoki v2.4.2 PKCS#11 implementation for Linux," http://sourceforge.net/projects/opencryptoki.

U.S.Rawat is Senior Lecturer and Ph.D. candidate in the Department of Computer Science and Engineering at Jaypee University of Engineering and Technology, Guna, India. He earned his M.E. in Computer Engineering from SGSITS, Indore in 2003. He is having 10 years of teaching experience. His current research interests include Information Systems Security and File Systems.

Shishir Kumar is currently working as Professor and Head in Department of Computer Science and Engineering at Jaypee University of Engineering and Technology, Guna, India. He has completed his PhD (Computer Science) in 2005. He is having around 13 years of teaching experience. His current areas of interest are Information Systems Security & Image Processing.