Assignment 4 Design Document

Benjamin Ross (bpross@ucsc.edu) , Matthew Musselman (mmusselm@ucsc.edu), Quentin Rivers(qrivers@ucsc.edu)
CMPS111, Spring 2012

1 Goals

The goal of this project is to implement file system encryption with a combination of system calls and user level code for the Minix 3 environment.

2 Available Resources

We have been given an AES sample program that takes in two keys and an input file and encrypts/decrypts the file. Because of the way AES works, encryption and decryption is done in the same way. This code was written by Professor Miller and will be used as a base for all of our encryption/decryption in this project. We also have hints attached to the specification documentation as to where we should place our files and the approximate location of our code in certain methods.

3 Design

Syscall:
The design of the syscall setkey(k0,k1) is straightforward. Because Minix uses an virtual file system (VFS) on top of the minix file system (MFS) the system call is going to be twofold. A system call must first be implemented in the VFS to call the appropriate system call in MFS. The system call is merely going to pass all of the appropriate information to VFS, which in turn is going to pass that to MFS. The MFS is going to keep track of all of the keys in a two dimensional array of ints. It will be MAX_NUMBER_OF_KEYS x 3. [i][0] will correspond to the UID, [i][1] will correspond to K0, and [i][2] will correspond to k1. We will convert the ints stored into the key on the fly.
This array will be held in main.c of the MFS. Entering the new key into the table will be done like this:

```
add_to_table(key,userid,key_table)
      if (entries == MAX_NUM_KEYS)
            print EPERM
      else if k0 and k1 == 0 // remove key
            iterate over table until entry found
            remove entry
            return success
            return error if no key found for UID
      else
            iterate over table until first entry found
```

```
            check to see if UID is already in table and set values
            set values in key_entry
            return success
```

To check if the table is full, we will use a global variable entries to keep track of the number of entries

For testing purposes, the syscall print_key_table() will be used. The purpose of this method is to test to make sure that keys are correctly being added to the table, and being held.

```
print_key_table()
      for entry
            print USERD: %d Key: %d\n
```

To encrypt the file in the Kernel, we will be modifying read.c in MFS. We will be modifying rw_chunk() to be exact. Before the file is read, we want to check to see if the OS thinks the file is encrypted. If the sticky bit is set, we will read the buffer into our encrypt function (held in mfs/encrypt.c) and decrypt the buffer. Then read.c's sys_safecopyto copies this buffer to user space. Then we decrypt the buffer so the changed buffer is not written back to the file system. For write, if the sticky bit is set, we call the encrypt function on the buffer after sys_safecopyfrom updates the buffer from user space. Our encryption in encrypt.c looks like this:
encrypt.c

```
encrypt_buf(uid,fid,buf,chunk)
            check to see if the user has a key
            if no key
                  return
            generate the key for the user
            setup AES encryption
            if chunk > 16 //this means that the buffer is writing more than
16 bytes
                  for ctr < chunk/16
                        encrypt 16 bytes at a time
            else // 16 or less bytes
                  for ctr < chunk
                        encrypt bytes
```

Protectfile will be designed like that AES example that Professor Miller gave to us. The only difference will be that we will use the actual file id of the file, as opposed to the dummy file id in encrypt.c
protectfile.c:

```
get_full_key(usr_key):
      usr_key_start = 16 - strlen(usr_key)
      key[16]

      from i = 0 to i = 7:
            byte = 0;
```

```
            if 2 * i < usr_key_len:
                  byte |= hex_value(usr_key[usr_key_len - 1 - 2 * i]))
            if 2 * i + 1 < usr_key_len:
                  byte |= hex_value(
                              usr_key[usr_key_len - 1 - (2 * i + 1)]) << 4)
            key[i] = byte

      return key

main:
      mode = argv[1]
      usr_key = argv[2]
      filename = argv[3]

      stat_err = stat(filename, file_stats)
      file_nr = file_stats->st_ino
      is_sticky = (file_stats->st_mode) & S_ISVTX

      if !is_sticky && mode == 'd'
            error, file not encrypted

      if is_sticky && mode == 'e'
            error, file already encrypted
      setup encryption using the provided file

      key = get_key (usr_key)

      turn off the sticky bit // so the system cannot encrypt/decrypt

      if mode == 'e'
            encrypt(file, key)
            turn on the sticky bit
      else
            encrypt(file, key) // encryption & decryption work the same way
```

Extra Credit:
Have chmod do this addtional behavior:

```
check if sticky bit set
      if set
            check if setting sticky bit
                  if true do nothing
                  else decrypt the file
      not set
            check if setting sticky bit
                  if true encrypt file
            else do nothing
```

What we would do is use VFS to change the mode and then encrypt the file using what we have written in the kernel or we would decrypt the file and then change the mode. This would allow us to leverage our code that we have in the kernel. We were not able to figure out how to get the file to copy itself in the kernel, because we did not have enough time.

4 Testing

Testing for this project will need to make sure that each component of the project works separately, before integrating them together.

Test Case 1:
Set Key for User

Allow the user to set their encryption key using the provided system call

Precondition: Must be a valid user on the MINIX System

Postcondition: User has an encryption key, which can be used for file en/decryption

Procedure: Have test program call the system call setkey(k0,k1) and have the system print out the User ID and the generated encryption key. Check to make sure User ID matches the user and check to make sure the encryption key matches the key passed in by the user

Test Case 2:
Store Key for User

Allow the user to set their encryption key and store this value in a table

Precondition: Must be a valid user on the MINIX System

Postcondition: User has an encryption key that is stored on the system, which can be used for file en/decryption

Procedure: Have test program call the system call setkey(k0,k1) and have the system store the User ID and key in a dictionary. Have a method that will print out the User/Key table and verify that the User ID is correct and that the encryption key is correct

Test Case 3:
Allow Up to 8 Users to have Encryption Keys

Allow up to 8 Users on a MINIX system to store encryption keys in the table

Precondition: Must be a valid user on the MINIX System. Less than 8 Users have stored encryption keys

Postcondition: User has an encryption key that is stored on the system, which can be used for file en/decryption

Procedure: Same as test case 2, but test for multiple users. Test for 9 users, which allows us to make sure that no more than 8 users can have an encryption key

Test Case 4
Test the Usage of protectfile

- Check to see if program responds to lack of key
  - Should Not Accept
- Check to see if program responds to lack of file
  - Should Not Accept
- Check to see if program responds to lack of mode [e|d]
  - Should Not Accept
- Check to see if program responds to correct usage
  - Should Accept

Test Case 5

Test the Key usage of protectfile

Check to see if program accepts a HEX key

Should Accept

Check to see if program accepts a NON-HEX key

Should Not Accept

Check to see if program accepts key longer than 16 chars

Should Not Accept

Check to see if program accepts key from 1-16 chars

Should Accept

Test Case 6

Test to see if encryption works

If sticky bit is not set, set it and encrypt the file

Should Accept

If sticky bit is set, don't change it and don't encrypt the file

Should Accept

Test Case 7

Test to see if decryption works

If sticky bit is set, change it and decrypt the file

Should Accept

If sticky bit is not set, don't change it and don't decrypt

Should Accept

Test Case 8

Test to see if FS level encryption works

If sticky bit is set, encrypt the file on write with User Key in Kernel

Should Accept

If sticky bit is not set, do not encrypt the file on write

Should Accept

Test Case 9

Test to see if FS level decryption works

If sticky bit is set, decrypt the file on read with User Key in Kernel

Should Accept

If sticky bit is not set, do not decrypt the file on read

Should Accept