In His Sublime Name

GPGPU-based Secure File System

Mohsen Koohi
Burge Computer Lab
http://www.burge.ir/GSFS

Chapter 3

Security Structure of GSFS

In this chapter, we consider the security structure of GPGPU-based Secure File System (GSFS). First, we explain the security requirements of the GSFS and then we attend the details and consider the integrity and confidentiality methods in different parts of the file system.

3.1 Security requirements for GSFS

The security requirements of GSFS are:

1. Integirty and confidentiality of the inodes: GSFS is a secure file system that provides both confidentiality and integrity of data. Therefore, each manipulation of superblock fields or tables of file system is detected at the beginning of the mounting and is represented to the user. Also, inodes manipulations are detected while accessing them and for reular inodes, the manipulation of data blocks is realized when a user accesses the changed block.

2. Selecting secure inodes: The users are responsible for requesting authentication and confidentiality and all information is not encrypted regularly. When a user needs authentication and confidentiality for a directory, he requests GSFS to make it secure. After that, all children of that inode will be secure. The data and metadata of each secure inode are encrypted and its integrity is evaluated. This user request-based security makes the file system more flexible and users will have secure and insecure inodes. For example, they can request security only for their private data and other executible files can be insecure. In this way, the total space of disk is not encrypted and therefore, the performance of the system is improved.

3. Secure inode sharing: GSFS provides sharing inodes between different users. The owner of inode can add other users to the users list of

a secure inode for reading or writing. The owner of inode can also revoke any users of that inode.

4. Increasing and decreasing the users of inodes in each level of the tree structure of the file system: In GSFS, the user access level can be increased or decreased in each level. The owner of a secure inode can add some users to a child of that inode or delete some users of a child. This action can be done in each level of the tree structure of the file system. Therefore, it is possible to add some users to a child of a secure inode and these new users may have no access to other children of the parent. Also, it is possible to delete some users of the parent in a child and in this way, some users of parent have no access to this child. GSFS uses Crust key revocation to speed up the key derivation and minimize the required memory.

5. Differentiating authentication from confidentiality: Authentication in GSFS is done by the root user public key and confidentiality is done by the users public key. Therefore, there is no need to discuss differentiating reader from writer.

6. Reducing the usage of the public key cryptography: GSFS only encrypts the twigs of the secure inodes by the public keys and uses Cryptree cryptographic links for the underlying inodes to encrypt their keys by the key of their parents. In this way, GSFS tries to minimize the usage of the public key cryptography.

3.2 Security of directory inodes

Each secure inode in GSFS has a hash value that is stored as the authentication code and used for integrity checking.
Each secure directory inode or directory inode with some secure directory inodes should be authenticated. Therefore, directory entries(dentries) of directory inodes should be authenticated. This is done by producing a hash value for each dentry that is stored in a particular place in dentries page of inode.
This page is an m-ary tree whose its leaves include the dentries and in each level, some children are selected to produce parent hash and this process continues to compute the hash of root. The root value of dentries page is stored in the inode. When the inode wants to access a dentry, the stored hash values on the path from the root to this dentry hash value, are considered to have correct values for that dentry. Also, the hash of the dentry is computed and compared with the hash value given from dentries page.
For confidentiality of secure inodes, each secure twig inode has an owner key that is selected randomly and a version for *Crust* that is increased after each revocation. Using the owner key and the version, we can produce a

Crust key that is able to produce all keys with a version equal or less than the Crust version.
After each revocation, the version of Crust is increased and a new Crust key is produced using the owner key. As stated in the previous chapter, the new Crust key is able to produce all previous keys, but the old Crust key can't produce the key with the new version.
The newest Crust key is encrypted by the public keys of all users who have access to the inode and is written in the user block. The owner key is only encrypted by the public key of the owner and is also written in the user block. The hash value of the user block is computed and stored in the inode and in this way, user block authentication is achieved.
GSFS doesn't necessarily allocate a different Crust key for each inode, and a Crust key can be shared between some inodes. Each inode first derives the key with its own version from Crust key and then uses that key to derive its own key. But there are two cases that a child inode has a different Crust key:

- The first case is when the access of some users in the parent inode is not allowed for the child inode. In this case, a new owner key is allocated to the child and the new Crust key is created for it.
- The second case is when the set of users who have access to the child inode is more general than the users who have access to the parent inode. In this case, as the previous case, a new owner key is allocated for the child inode and the new Crust key is produced for it. The new Crust key is only encrypted by the public keys of the new users and stored in the user block. In this case, because of access of the parent inode users, we use Cryptree cryptographic links method to encrypt the child crust key with the parent inode key and store it in the user block. Therefore, the parent users don't require public key cryptography to access the child inode.

3.3 Security of regular inodes

GSFS, uses *GCM*[1] for security of data blocks of secure regular inodes. GCM [Dwo07] is a block cipher that provides assurance of confidentiality and authenticity of the data and is recommended by *NIST* in 2007.
To encrypt a block with GCM, an Initial Vector(IV) and a key are required and an Authenticated Tag(AT) is produced for assurance of authenticity. The key of the block is derived form Crust key; therefore, we should store the triple value of [Version, IV, AT](*VIA*) for each block.
The VIAs are stored in blocks of the filesystem and for assurance of integrity, we use hash of each block to form hash pages. In the next level hash pages is

[1] Galois/Counter Mode

formed from previous level hash pages. This continues until the two values of the roots of the last two hash pages are reached and stored on the inode. When one user reads one page, the integrity of hash pages is evaluated by parsing this tree structure reversely until the integrated value of VIA of the target page is reached. Then VIA is used by GCM to decrypt and authenticate the requested page.
The vaule of the IV for each page of the data should be increased by one in each time of the writing.

3.4 Security of other parts

Each secure inode produces a hash value as the authenticity assurance code (AAC) and the AACs of all inodes are stored in the Inode Hash Pages(IHP) table. The hash value of this table and other tables are computed and stored in the superblock.
The hash value of the superblock with a time stamp is signed by the root user private key and stored at the end of the super block after each filesystem synchronization. This value is examined at the beginning of mounting and in this way, the integrity of all parts of the file system is evaluated. If there are some manipulations of superblock or one of the tables, an alert is sent to the user.
To defeat replay attacks, the hash value of the superblock and time stamp is wirtten to root user at the beginning and at the end of the mounting, and root user will be aware of the last file system time stamp and hash value of the superblock.
If we can't trust on the system time, we can use trusted timestamping servers to send the hash value of the super block and store the returned timestamped value of that hash which is signed by the private key of the server. In this case, we need only the public key of the server for integrity evaluating. It should be noted that in the current version, we have trusted the time of system, and using trusted timestamping servers can be added to GSFS in future versions.
GSFS uses RSA-1024 for public key cryptography and *Skein*[2] as its hash function. Skein is one of five hash functions selected for the last round of the SHA3 competition[3].

3.5 Conclusion

In this chapter, we considered the security structure of the GSFS. First, we stated the security requirements of the GSFS and then, explained how to

[2] http://www.skein-hash.info

[3] http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/Round2_Report_NISTIR_7764.pdf

provide confidentiality and integrity for secure directory inodes and secure regular inodes. Finally, we presented the method for evaluating integrity of the superblock by using timestamp.

Chapter 4

GSFS Implementation

In this chapter, we will consider the implementation of the GSFS. First, we introduce the general structure of the GSFS and then we explain the details of each part. After that, we consider some functions and then the synchronization method in GSFS is described.

4.1 General structure

There are several ways to implement a file system. By using *FUSE*[1], it is possible to implement a file system in user mode. Also, it is possible to implement a file system by NFS. But the most efficient way is to implement system file as a kernel module. In this way, in addition to more security because of running functions in kernel mode, the consecutive context switches are avioded.
GSFS is impleneted as a kernel module for linux 2.6.34. It doesn't require a base file system and is able to mount independently on each block device. The first mount requires sending "-o create" optional parameter to initialize the tables and superblock.
At the beginning of mounting, the root user should tell his private key to the GSFS. It is needed for signing the superblock hash and time stamp for changes that will occur.
The general structure of GSFS is shown in Figure 4.1.

4.2 GSFS tables

GSFS uses some tables for accessing different parts of the file system.
Block Allocation Table(BAT) is responsible for allocating disk blocks. It devotes bit to each block. The 0 value of this bit says that the related block is free

[1]File system in user space

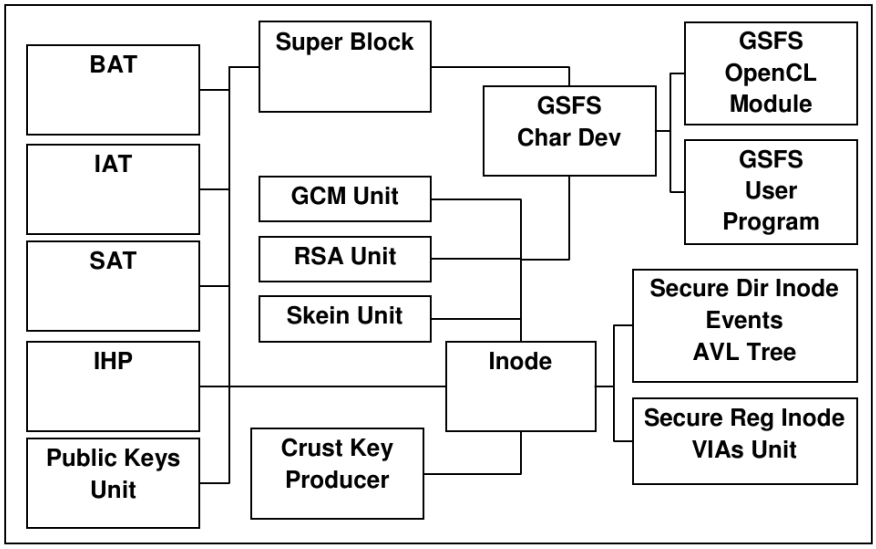

Figure 4.1: GSFS structure

and the 1 value states that is allocated. In this way, GSFS searches the BAT when it needs a block. The start and end block number of BAT and also the last allocated block number are stored in the superblock. The hash value of BAT is written in superblock for next time authentication.
Inode Allocation Table(IAT) is responsible for allocating inode number and also specifying the block number of each inode. It allocates 4 bytes for each inode that makes it possible to store a 32 bit unisigned integer that states the block number of the inode. The zero value of this integer states that the related inode number is not allocated. The start and end block number of IAT, the last allocated inode number, and the hash value of IAT are written to the superblock.
Secure indecis Allocation Tabel(SAT) is responsible for allocating index for each secure inode to store its hash value in the *Inodes Hash Pages(IHP)*. This index specifies the location of the hash value of the inode in the IHP. SAT allocates one bit for each secure index such as BAT. The start and the end block number of SAT and IHP, the last index allocated by the SAT, and the hash values of SAT and IHP are stored in the superblock.
Access to each page of these tables is done through a LRU[2] and therefore, the last pages accessed by these tables are presented. Thus, accessing and modifing the pages can be done without reading them from disk.
Public Keys Unit the manager of the users pages. Each user has a page that stores his public key. Public keys unit uses one block of the disk to store the binary values of user ID and its user page block number. The hash value of this page, in addition to the users pages, is stored in the superblock and

[2]Least Recently Used

used for integrity evaluation.

4.3 Security units of GSFS

GSFS implements Crust key functions in the *Crust Key Producer Unit*. This unit is responsible for:

1. Producing Crust keys from the owner key and requested version.
2. Deriving a previous key with requested version from a Crust key.

In the current implementation, GSFS uses $d = 4, m = 16$ and therefore, it is possible to revoke $16^4 = 2^{16} \approx 65K$ key.
The *RSA Unit* is responsible for encyptinh/decrypting by RSA-1024 public key cryptography algorithm. This unit is a part of *PolarSSL*[3] package and a little changes are done on it to be used in kernel and user modes.
GCM Unit encrypts/decrypts data blocks of regular secure inodes. GCM is a counter mode block cipher that is able to authenticate, in addition to encryption. It uses *Galois Hash* function to compute the hash value of the block and use this value for integrity checking. Further information about GCM can be found in [Dwo07].
GCM Unit uses kernel AES function and if the OpenCL module is ready, it uses OpenCL module to encrypt/decrypt in parallel.
Hashing is done in the *Skein Unit*. We used its code from its web site.
To authenticate the hash values of the dentries, GSFS uses a merkle tree to speed up dentry hash value integrity checking, and also updating processes.

4.4 GSFS character device and user program

Users commands in GSFS are sent from user mode to the kernel mode by using *GSFS Character Device*. These commands are:

- User login
- User logout
- Making an inode secure
- Adding some users to a secure inode
- Revoking some users access from a secure inode

GSFS character device is a virtual character device and GSFS uses it to create a character device inode to send instructions from user mode to kernel mode. In addition to this character device, a user level program is needed to write instructions. This program is *GSFS User Program (GUP)*. GUP can

[3] http://www.polarssl.org

be implemented in each programming language and in the current version, it is implemented in the C language. Figure 4.2 represents GUP.

```
linux:/home/Programs/GSFS/GSFS_v0.5 # ./gsfs_user_prog.o ./gsfs_cdev
Input the number:
        1) Login
        2) Logout
        3) Generate RSA-1024 keys for new user
        4) Make a directory secure (more special)
        5) Add users to a secure directory (more public)
        6) Revoke users access to a secure directory
        7) Exit
```

Figure 4.2: GSFS User Program

When a new user uses GSFS for the first time, he uses the 3rd option to create public and private key for himself. These two keys are stored in the user defined location and the public key is sent to the kernel and stored in the Public Keys table.
After this, in the begging of each usage, the user can login by the 1st option and he should specify the location of his private key. GUP reads the private key, encrypts a random value by the private key, and sends the random value and encrypted value to the kernel.
If the GSFS in kernel is able to decrypt the encrypted value and give an equal value to that random value, the kernel will be assured that the private key is correct and allows the user to login. Also, it sotres the private key of the user in its memory for the next usages.
In the exit time, the user logs out by the 2nd option. It should be noted that GUP should not be presented everytime, and the user can run it when he has some orders.
To make an inode secure, the user selects the 4th option. To add some users to a secure inode, he uses the 5th option. Finally, to revoke accesses of some users to a secure inode, he selects the 6th option. The details of these three options will be presented in the next sections.

4.5 GSFS OpenCL Module

In the current version, GSFS uses a user level OpenCL program to enhance the cryptography rate by GPU parallel processing. *GSFS OpenCL Module* is a C program that uses OpenCL library for parallel cryptography and because of writing in the user mode in the current version, we named it *GSFS User Module(GUM)*.
GUM connection with kernel is done by the GSFS character device. It begins to work in the background as a daemon and waits until a command is sent to it; then, it creates a thread by the *"pthread"* library to run functions

needed for the entered command and returns itself to wait for another kernel command. In this way, the kernel can send commands to GUM quickly and there is no need to wait for completeing the previous commands.
GUM is able to do commands in parallel by CPU and GPU using OpenCL. Each 4096 byte page is divided to 256 16-byte pieces and each work-item is responsible for specifing the value of each piece. In the current version, the encrpytion and decryption are done by the GUM and authentication is done in the kernel mode.
To delete the memory copying time from kernel mode to the GUM virtual memory, GSFS maps the pages to GUM virtual memory and thereby, the only memory copy is done from main memory to the GPU memroy.
This process starts by creating a mapping using mmap system call in the GUM. The address of this mapping is sent to the kernel. When GUM writes to kernel, it is blocked by using a semaphore. Everytime kernel has some pages to encrypt/decrypt, first it adds the pages to the GUM virtual memory and then releases the GUM semaphore and gets a new semaphore to block itself.
Now, GUM returns to run and when the results are ready, GUM again writes to the kernel; in this case the kernel removes the pages from GUM virtual memory and releases another blocked semaphore to allow the blocked process to use the ready results. Then, GUM unmaps the created mapping.
The OpenCL kernel written for GUM is an AES cipher that is responsible for creating GCTR pages of GCM by using IV, and rounds keys. These GCTR pages iare used in the kernel to be xor-ed with the plain text to create cipher text or vice versa.

4.6 AVL tree for events of directory inodes

GSFS uses an AVL tree to manage the events and changes in the children of the directory inodes. It uses AVL tree because an AVL tree is able to balance itself, and the order of adding children is not important for tree searching. Thus, the search complexity will be $\log(n)$ in the worst case forever.
The children information are dentry hash value, key, and users. When GSFS requires one of these fields, it first searches the AVL tree of the inode, and if it isn't found, GSFS reads the field from the disk and stores it in the child node of the AVL tree.

4.7 Secure regular inodes VIAs unit

As stated in the previous chapter, each data block of a regular secure inode has a 3-ary VIA value formed from Version, IV, AT. These VIAs are stored in the disk, and their integrity should be checkd in each access. Therefore, GSFS uses a merkle tree to manage the VIA values in the *Secure Regular Inode*

VIAs unit (SRI).
In this way, GSFS in the first level stores each of 128 32-bytes pieces bytes of the VIA values in a block and computes its hash value. Next, in the second level, 256 16-bytes pieces of the block number and hash values of the first level are stored in a block and the hash value is computed. In the third level, also, 256 16-bytes pieces of the second level block number and hash values are stored in a block and its hash is computed. Two pieces of hash value and block number of the third level are stored in the inode.
Consequently, GSFS is able to write $4096 * 128 * 256 * 256 * 2 = 2^{12+7+8+8+1} = 2^{36}$ bytes or 64GB for each secure regular inode.
To accelerate the access to the VIA values, first 8 values of VIAs that are for the first 8 blocks are stored in the inode. Also, the first 6 values of the block number and hash value of the second level and the first 4 values of the block number and hash value of third level are also stored in the inode.
By using this merkle tree, the authentication and modification processes are accelerated. Also, by storing the first values of each level in the inode, the access time is enhanced.

4.8 Some functions of GSFS

In this section, we consider some functions of the GSFS. Firstly, we consider the key computation method for each block of data, and then, we states the read and write functions. After that, we explain the functions of making an inode secure, adding users to a secure inode, and revoking a user access to a secure inode.

4.8.1 Block key computation

The key of each block is derived from the inode key in these states:

1. The curst key producer unit derives the version key by using the entered version.
2. The inode number is concatenated to the version key and the hash of this value is computed as the inode key.
3. The block number is concatenated to the inode key and the block key is computed by hashing this value.

It should be noted that the various versions of the inode keys are stored in a LRU in that inode to accelerate the access to them.

4.8.2 Read function

In the read function, a number of blocks are read from disk and stored in the address space. The levels of this function are:

1. Restoring the VIA values of the requested blocks from SRI unit.
2. Computing the block keys of each of requested blocks concerning their version.
3. Sending the read command from disk in bio level.
4. Comparing the AT stored in the VIA and that computed from read block of disk for each of the blocks in the GCM unit.
5. Computing the GCTR pages and decrypting the authenticated blocks in the GCM unit.

4.8.3 Write function

The states of the write function are:

1. Restoring the VIA values of the blocks from SRI unit.
2. Increasing the IV value of each VIA and setting the version field of each VIA to the maximum version of the Crust key of the inode.
3. Computing the AT of each block in the GCM unit.
4. Sending the new VIAs to the SRI to store on disk.
5. Computing GCTR pages of each block and encrypting the blocks in the GCM unit.
6. Writing the encrypted pages on the disk

4.8.4 Make an inode secure function

Making an inode secure is user for two cases. In the first case, we want to make an insecure inode secure. In the second case, we want to limit the users of the parent inode to access that inode. In both cases, we want to make an inode access level more specific. The levels of this function are:

1. Allocating a block for user block from BAT.
2. Allocating a secure index from SAT.
3. Allocating a random key as the owner key of the inode.
4. Producing the Crust key in the Crust key producer unit.
5. Completing the user block by writing the encrypted values of the owner key and the Crust key by the public key of the owner.
6. Encrypting the inode dentry in its parent.
7. Allocating secure index for all parents up to the root inode for authenticating them from SAT.

4.8.5 Add users to a secure inode function

This function is also used for two cases. In the first case, we want to add some users to a secure inode that has user block. In the second case, we want to add some users to a secure inode that doesn't have the user block. In this case, the new added users have access to this inode and don't have access to the parent inode, and the parent inode users have access to this inode. In both cases, we want to make the inode access level more general. The levels of this function are:

1. Allocating user block, owner key, Crust key, and completing the user block if it doesn't exist.
2. Storing the encrypted Crust key of the inode by the last version of the Crust key of the parent in the parent link field of the user block, if the user block is allocated now (the second case).
3. Adding the encrypted Crust key by the public keys of the new users to the user block.
4. Decrypting the inode dentry in its parent and doing it for all secure parents inode, if the user block is newly allocated.
5. Updating the users list of the children inodes without user block to allow the new users to access them.

4.8.6 User revocation function

This function is used to revoke some users access to an inode and its children. The levels of this function are:

1. Increasing the version of Crust key by one and creating a new Crust key by using the owner key in the Crust key producer unit.
2. Updating the user block of the inode to encrypt the new created Crust key by public keys of the only allowed users.
3. Updating the parent link if it exists.
4. Updating the users of inode to delete the revoked user access.
5. Encrypting the inode dentry in its parent inode by the new version of Crust key
6. Increasing the version of Crust key and encrypting the parent link and Crust key for any child that has a user block.

4.9 GSFS synchronization

As we saw in functions of the previous section, we require a technique for accessing inodes while the mutual exclusion should be adhered to and dead-

lock must be prevented. Therefore, we need some semaphores for this purpose.
Each inode has a *Read/Write Semaphore* and there are two rules in GSFS to synchronize the access of the inodes:

1. Each function can get the semaphore of each inode and then, it can get the semaphore of its parent but not vice versa.
2. Semaphores of inodes can be released only in the reverse direction that had been gotten.

4.10 GSFS code

The code of GSFS is formed from 3 parts:

1. Skein: About 1000 lines (adopted from Skein website).
2. RSA: About 4000 lines (adopted from PolarSSL package).
3. About 16000 lines written for GSFS in this project and includes the implementation of:
   a) All units stated in this chapter.
   b) All layers of file system down to read/write in bio level from/to disk.
   c) Inode, file, and address space operations.

4.11 Conclusion

In this chapter, we introduced the internal structure of the GSFS and its implementation. We explained different units of the GSFS and some functions. Finally, we stated the synchronization method of the GSFS and the code .