Intel® Protected File System Library

- A new feature called Intel Protection File System Library is introduced in the Intel SGX 1.7 Release. This Library is used to create, operate and delete files inside the enclave. To make use of the Intel Protected File system Library we need the following requirements:
    - Visual Studio 2015,
    - Intel SGX SDK version 1.7
    - Intel SGX PSW version 1.7
- The above requirements are essential for implementing Intel SGX Protected File System. In this document we will discuss regarding the architecture, API's, Implementation and Limitations of Intel Protected File System Library.

Overview of Intel® Protected File System Library:

- Intel® Protected File System Library provides protected files API for Intel® SGX enclaves. It supports a basic subset of the regular C file API and enables you to create files and work with them as you would normally do from a regular application.
- We have 15 file operation functions API's provided by Intel SGX. These API work almost the same as the regular C file API.
- With this API, the files are encrypted and saved on the untrusted disk during a write operation, and they are verified for confidentiality and integrity during a read operation.
- To encrypt a file, you should provide a file encryption key. This key is a 128 bits key, and is used as a key derivation key, to generate multiple encryption keys.
- The key derivation key used as an input to one of the key derivation functions is called a key derivation key, can be generated by an approved cryptographic random bit generator, or by an approved automated key establishment process. Another option is to use automatic keys derived from the enclave sealing key.
- This way we can keep our files secure and safe inside the Enclave. Since our files are encrypted and stored they are safe and secure inside the enclave.

Intel Protected File System API:

The Intel Protected File System Library provides the following functionalities.

- sgx_fopen
- sgx_fopen_auto_key
- sgx_fclose
- sgx_fread
- sgx_fwrite
- sgx_fflush
- sgx_ftell
- sgx_fseek
- sgx_feof
- sgx_ferror
- sgx_clearerr
- sgx_remove
- sgx_fexport_auto_key
- sgx_fimport_auto_key
- sgx_fclear_cache

The above mentioned API's are present in the SGX Protected FS trusted library. And these can be called only within the trusted enclave code which makes our files secure.

sgx_fopen():

- The sgx_fopen function creates or opens a protected file by the key provided by developer.
- Returns a file handle of the type SGX_FILE pointer, otherwise NULL is returned.

sgx_fopen_auto_key():

- The sgx_fopen_auto_key function creates or opens a protected file by the key provided by Enclave sealing key itself.
- Returns a file handle of the type SGX_FILE pointer, otherwise NULL is returned.

sgx_fclose():

- The sgx_fclose function closes a protected file handle.
- Returns 0 if file successfully closed else returns 1.

sgx_fread():

- The sgx_fread function reads the requested amount of data from the file, and extends the file pointer by that amount.
- Returns the no of blocks of size read from the file.

sgx_fwrite():

- The sgx_fwrite function writes the given amount of data to the file, and extends the file pointer by that amount.
- Returns the no of blocks of size written to the file.

sgx_fflush():

- The sgx_fflush function forces a cache flush, and if it returns successfully, it is guaranteed that your changes are committed to a file on the disk.
- Returns 0 if successful and 1 if not successful.

sgx_ftell():

- The sgx_ftell function creates or opens a protected file.
- Returns the current value of the position indicator of the file if successful, otherwise, -1 is returned.

sgx_fseek():

- The sgx_fseek function sets the current value of the position indicator of the file.
- Returns 0 if successful, otherwise -1 is returned.

sgx_feof():

- The sgx_feof function tells the caller if the file's position indicator hit the end of the file in a previous read operation.
- Returns 0 if end of file was NOT reached, otherwise 1 is returned if end of file reached.

sgx_ferror():

- The sgx_ferror function returns the latest operation error code.
- Returns 0 if no errors else returns the respective error code.

sgx_clearerr():

- The sgx_clearerr function attempts to repair a bad file status, and also clears the end-of-file flag.
- Return type: None

sgx_remove():

- The sgx_remove function deletes a file from the file system.
- Returns 0 if successful, otherwise 1 is returned.

sgx_fexport_auto_key():

- The sgx_fexport_auto_key function is used for exporting the latest key used for the file encryption.
- Returns 0 if successful, otherwise 1 is returned.

sgx_fimport_auto_key():

- The sgx_fimport_auto_key function is used for importing a Protected FS auto key file created on a different enclave or platform.
- Returns 0 if successful, otherwise 1 is returned.

sgx_fclear_cache():

- The sgx_fclear_cache function is used for clearing the internal file cache. The function scrubs all the data from the cache, and releases all the allocated cache memory.
- Returns 0 if successful, otherwise 1 is returned.

Note:

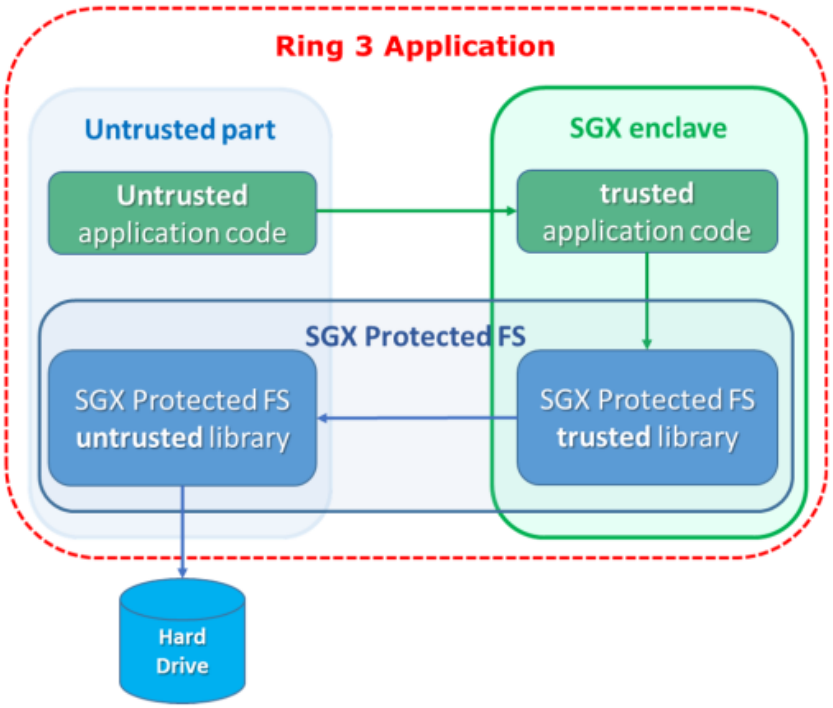
Please refer the “IntelProtectedFileSystem_Reference .pdf” attachment for more information about **Intel Protected File System API**

Intel Protected File System Application Layout:

To use the Intel SGX Protected File System libraries:

1. The enclave must be linked with sgx_tprotected_fs.lib
2. The application must be linked with sgx_uprotected_fs.lib
3. The enclave's EDL file must 'import' all the functions from sgx_tprotected_fs.edl
4. The source files should 'include' sgx_tprotected_fs.h

The following figure demonstrates how the Protected File System (FS) works inside an Intel® SGX application:



Intel Protected File System Layout

Untrusted Application Code:

- It contains the File Description information like filename, mode, etc., which is used to create the file in the enclave. The Application Code also contains the Ecalls to the enclave.

Trusted Application Code:

- It contains the Intel Protected File System function calls such as sgx_fopen(), sgx_fclose(), sgx_fwrite(), sgx_fread(), etc., which have already been provided by Intel SGX

The control flow from the Application to the Enclave takes place in the following manner:

1. The file which needs to be created is called from the application and the details are sent to the enclave via Ecall.
2. After that, the control reaches the enclave the sgx_fopen() or sgx_fopen_auto_key() function is called by enclave, to create the required file from inside the enclave.
3. The control then flows to the SGX protected FS library which has the implementation of the below mentioned API. And there exist an Ocall to the SGX Protected FS untrusted Library. Then the control returns to the enclave.
4. After the file is created the file handle pointing to the file is returned to the application.
5. The application can read, write, delete and perform different operation on the file by generating an Ecall to the enclave which calls the corresponding API's specified below from the enclave.

Basic Implementation of Protected File System Application:

The developer needs to follow the below steps to create an application for implementing the SGX protected File system Library

1. Create an application and Enclave and the necessary libraries need to be added.
2. The enclave must be linked with the already provided sgx_tprotected_fs.lib. This can be done by adding the Library file to the Linker→Input→Additional Dependencies→Edit and then add (sgx_tprotected_fs.lib).
3. The Application must be linked with the provided sgx_uprotected_fs.lib. This can be done by adding the Library file to the Linker→Input→Additional Dependencies→Edit and then add (sgx_uprotected_fs.lib).
4. The enclave's edl must "import" all the functions from sgx_tprotected_fs.edl. The edl can be imported by using the command:
   from "filepath/sgx_tprotected_fs.edl" import *;
5. The source files should include "sgx_tprotected_fs.h". This can be done by adding to the source files:
   #include "sgx_tprotected_fs.h".

After following these steps we can use the functionalities provided by Intel PFS Library in our application. The application developed can be used to create files using SGX FPS functions and can be stored on the cache of the enclave and when the enclave cache is full then it is flushed to disk. User can call this function from the enclave by passing the control from the application (via an Ecall).

Here we are going to implement the file create, open, read, write and close operation using SGX and Protected File System Library.

Please find the below information about how to create a file, file open, file write, file read and file close operation process.

Process of Creating a File:

1. User needs to send the information of the file which is the filename and mode(read/write/both) to the enclave through an Ecall.
   E.g.: ecall_file_open(const char filename, const char mode)
2. Then the enclave calls the API sgx_fopen_auto_key(filename, mode). Now the file is created by the enclave. And the file handle obtained as a return value from the sgx_fopen_auto_key() is returned to the application using SGX_FILE pointer.

Process of Writing to a File:

1. User needs to send the information that needs to be written to the file which is data, size, count and file handle to the enclave through an Ecall.
   E.g.: ecall_file_write(void* data, size_t size, size_t count, SGX_FILE* filehandle).
2. Then the enclave calls the API sgx_fwrite(data, size, count, fileHandle). This function returns the size of the written data and sends it back to the Application.

Process of Reading from a File:

1. User get the information from file that needs to be read from the file which is data, size, count and file handle to the enclave through an Ecall.
   E.g.: ecall_file_read(void* data, size_t size, size_t count, SGX_FILE* filehandle).
2. Then the enclave calls the API sgx_fread(data, size, count, fileHandle). This function returns the size of the read data and sends it back to the Application.

Process of Closing a File:

1. User needs to send the information of the file handle to the enclave through an Ecall.
   E.g.: ecall_file_close(SGX_FILE *fp)
2. Then the enclave calls the API sgx_fclose(fp). Now the file is closed by the enclave.

Code for File Protection System Library using SGX:

Please find in the attachment for basic functionalities of Intel Protected File System Library implementation code.

Protected FS Security Non-Objectives

1. In order to moderate file swapping attacks (with two valid files), file names are checked during a file open operation (verifies that the current name is equal to the file name the file was created with).
2. However, if two files are created with the same file name, there is no way to protect against such a swapping attack.
    a. Since the files are saved in the regular FS, there is no protection against malicious file deletion or modification:
        i. This will only be detected when trying to read or write the modified section from the file (decryption will fail).
    b. There are several things that are not protected when using the Protected FS API, and anyone who can access the OS can see them:
        i. File name
        ii. File size (up to 4KB granularity)
        iii. File modification date
        iv. Key type (user or auto)
        v. Usage patterns
        vi. Read/Write offsets

3. If any of those items might expose sensitive information, and help a potential attacker, the enclave developer should add defense mechanisms on their own to protect against this.
    a. E.g.: An implementation of a “secure browser” should not save the ‘cookies’ with the names of their related websites, because an attacker can learn from that the user’s browsing history.

Limitation of Protected File System

1. Protected Files have meta-data embedded in them, only one file handle can be opened for writing at a time, or many file handles for reading.
2. Operating System protection mechanism is used for protecting against accidentally opening more than one ‘write’ file handle. If this protection is bypassed, the file will get corrupted.
3. An open file handle can be used by many threads inside the same enclave, the APIs include internal locks for handling this and the operations will be executed by one.