בס"ד



***Project 8 – SGX Based File Encryption Program “My-Safe”***

Created by – Ezra Block and Shy Tennenbaum

Instructor – Arie Haenel

Year – 5777 (2016 – 2017)

Semester - First Semester

Faculty – Computer Science

Course – Fundamentals For Software Security

Institution – Jerusalem College of Technology (JCT)

Programming Languages Used – C++, C#

Table of Contents

Abstract ---------------------------------------------------------- 4

Introduction ----------------------------------------------------- 5

* The Issue --------------------------------------------------------------------- 5
* The Solution ----------------------------------------------------------------- 6

Solution in Depth ----------------------------------------------- 7

* Solution Layout ------------------------------------------------------------- 7
* Creation of a Vault --------------------------------------------------------- 9
* Loading an Existing Vault ------------------------------------------------- 11
* Encryption Process --------------------------------------------------------- 14
* Decryption Process --------------------------------------------------------- 17

Operating Instructions ----------------------------------------- 20

## Abstract

The issue of security is very paramount in any organization. Remote attacks have become very common. Therefore, we created a program to deal with this exact problem, and aid to the never-ending war against hackers and various attackers. This code lock system is not just the normal single – user entered password, rather this program uses Intel SGX technology to seal file chosen from remote access and also encrypts them with a user entered password. The system works on virtual vaults each independent, and gives the option to lock files with it. This program deals with both authenticity and integrity problems. During the creation of a vault a master password is also created to add to the security of the vault. This way we hope to create another alternative solution to many attacks.

## Introduction

## 

## The Issue:

In the modern world today, a very important issue arose as software and computers in general become more abundant. This issue is security for software.   
With all the new threats that keep coming up, we can only block a certain amount. Many of them manage to pass all the security, and get complete access to one’s information and data on his computer.

However, we want to ensure that certain very important files won’t be able to be accessed from an attacker with a different computer, even if the attacker has all the information from the computer.

## The Solution:

## Our project comes to deal with this issue. With the use of the Intel SGX technology, the program we created insures that no one will be able to access files chosen from a different computer, and also has extra security so even if he does get access to the computer with the files on them, the attacker will still have to pass a security wall.

“Intel® Software Guard Extensions (Intel® SGX) is an Intel technology for application developers who are seeking to protect select code and data from disclosure or modification. Intel SGX makes such protections possible through the use of enclaves, which are protected areas of execution. “

As described above, the Intel SGX technology offers the option to create secure enclaves to run code during runtime.

However, this encryption program isn’t perfect and it has its limitation.  
First of all, it doesn’t support computers that don’t support Intel SGX (which quite a lot don’t yet). This is because it is dependent on the Intel SGX enclave.

There isn’t much we can do to make it support computers without Intel SGX, and it’s only a matter of time, until they are more common as is already starting to happen. Many of the newer computers do support Intel SGX, and it’s becoming more of a thing nowadays. So basically, our project probably going to much more useful in the near future.

In addition, the passwords the user chooses in the GUI, aren’t sent to the enclave in a very secure way. This is actually something that can be improved significantly by send them in a more secure way, and not as is done currently.

### 

### Solution in Depth

**Solution Layout:**

The program gives the user the option to create security vaults. These vaults are virtual vaults that can secure files via them. In other words, after creating a vault, the user can use the vault to encrypt the files in a safe way so that it won’t be able to be accessed even with all the information stored on the computer, from a different machine.

The main processes the program implements are creating a new vault. loading an existing vault, encrypting a file with the chosen vault, and decrypting a file in the chosen vault (Each vault can only decrypt the files it encrypted).

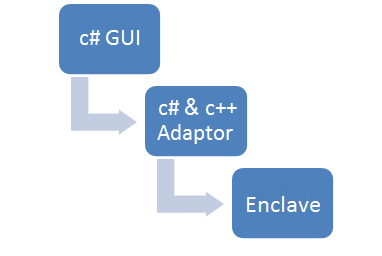
The idea is that if the user creates a vault, he or she can use to encrypt files he chooses. Those files are then considered as if they’re “in the vault”. So, as expected, the files that are in a certain vault (encrypted by a vault), can only be decrypted by it.

In order to insure this, each vault has a unique master key created inside its Intel SGX enclave which is used in the process of locking and unlocking of files that are in the vault during the encryption and decryption and the sealing and unsealing phase.

To keep the secret master key of a given vault secure, each vault has a master password chosen by the user during the creation which is used to lock the master key from use.

Finally, there is another password given to each file during the encryption, to give it an additional level of security.

The architecture of the program is as described below in the step diagram:



There are four main processes the program is in charge of doing, as described below:

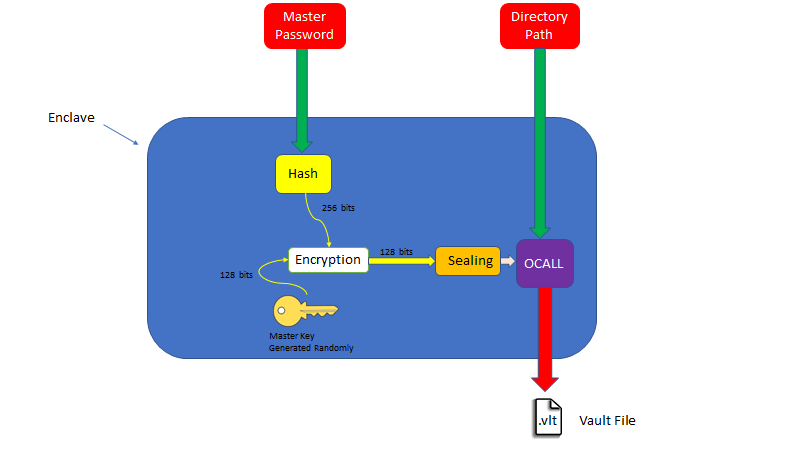
**Creation of A Vault:**

When creating a vault, the user must enter the following things:  
 a. A directory in which to store the vault.

b. A master password which is used to encrypt all the file keys generated by the enclave in the vault.

sgx\_status\_t create\_valut\_file(char \* path, char\* password, size\_t len)

The creation process is as presented below:



The function gets the master password and the directory path for the vault .vlt file. Firstly, it calculates the hash of the master password using the hash function sha-256, the master key which is generated randomly (128 bits long) in the enclave with AES-GCM 128 bit key cryptography function as shown below.

sgx\_status\_t res = sgx\_rijndael128GCM\_encrypt((sgx\_aes\_gcm\_128bit\_key\_t\*)hash, master\_key, 16, encrypted, iv, 12, NULL, 0, &out\_mac);

if (res)

return res;//Error

Then, it takes the encrypted master key and seals it in the enclave using MRENCALVE.

uint8\_t result = seal\_ex(encrypted, 44, sealed\_data, sealed\_size, &sealed\_size);

Finaly, it takes the encrypted master key and injects it into the newly created .vlt file called in an OCALL from the enclave.

encalve\_write\_file(&result, path, (char \*)sealed\_data, sealed\_size);

if (result)

{

return SGX\_ERROR\_FILE\_BAD\_STATUS;

}

return SGX\_SUCCESS;

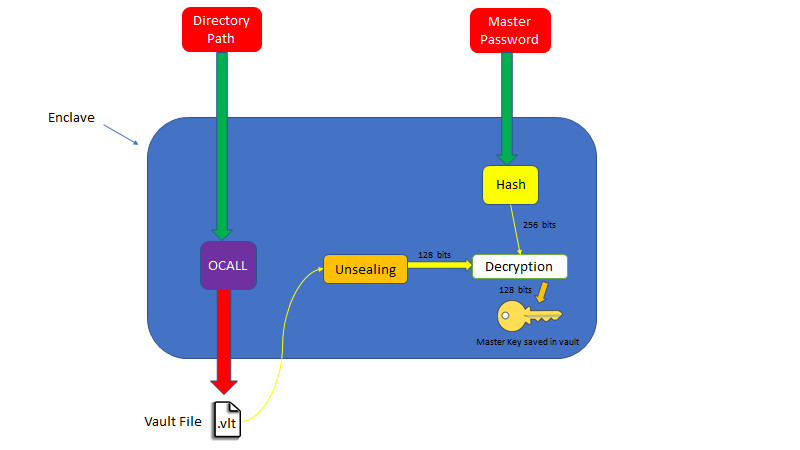
**Loading an Existing Vault:**

The user also has the option to load an already existing vault. In this case, the user must enter the directory where it is being saved, in order to load it.  
  
After loading or creating a vault, the user can use it to encrypt a file. In this case, he enters the following things:

1. The directory of the file, where it’s stored in the disk.
2. The master password for the vault.

sgx\_status\_t load\_valut\_from\_file(char \* path, char\* password, size\_t len)

As presented below, the loading of the .vlt file, works as following:



The method is given the full directory path and the master password. It calculates the hash of the master password using the hash function sha-256 (256 bits long).

sgx\_sha256\_hash\_t hash;

get\_sha256\_hash(password, len, &hash);

In addition, it also calls the .vlt file from the path given with an OCALL from the enclave. It then takes out the content and unseals it with MRENCLAVE to restore the encrypted master key.

uint8\_t sealed\_data[DEFAULT\_SEALED\_SIZE];

uint8\_t result;

encalve\_read\_file(&result, path, (char \*)sealed\_data, DEFAULT\_SEALED\_SIZE, NULL);

if (result)

{

return SGX\_ERROR\_FILE\_BAD\_STATUS;

}

And afterwards -

result = unseal(sealed\_data, DEFAULT\_SEALED\_SIZE, decrypted, DEFAULT\_DECRYPTED\_KEY\_SIZE);

if (result == -1)

return SGX\_ERROR\_MAC\_MISMATCH;

else if (result >0)

return SGX\_ERROR\_OUT\_OF\_MEMORY;

else if (result > 0)

return (sgx\_status\_t)result;

Finally, using the hash of the master password calculated earlier, it decrypts the master key, to be used later when using the loaded vault.

sgx\_status\_t res = sgx\_rijndael128GCM\_decrypt((sgx\_aes\_gcm\_128bit\_key\_t\*)hash, decrypted, 16, master\_key, iv, 12, NULL, 0, &old\_mac);

if (res)

return res;

memcpy(main\_key, master\_key, 16);

return SGX\_SUCCESS;

**Encryption of a File:**

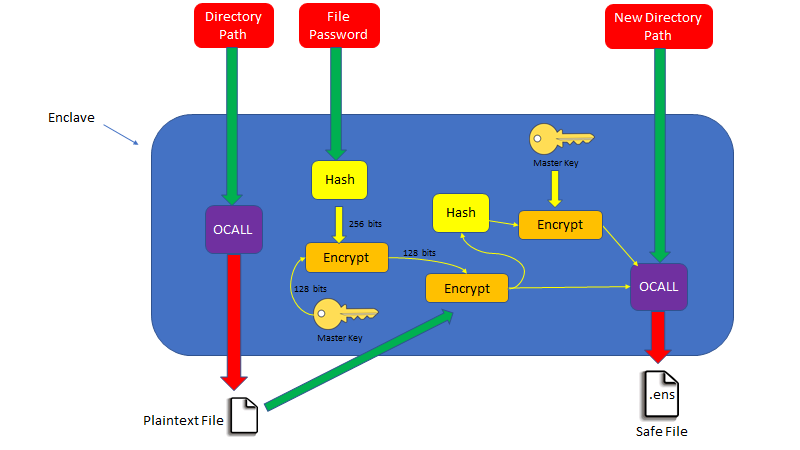
The encryption process is the main part of the program. It is used to encrypt and seal files in order to make them be inaccessible by any attacker from a different machine or that doesn’t have the file password.

When encrypting a file the user must enter the following things:

1. The full path of the file that the user wants encrypted
2. A new encrypted .ens file
3. A file password used to encrypt the file in addition to the sealing.

sgx\_status\_t enclave\_encrypt\_file(char \* path, char \* file\_password, size\_t len)

The functionality of the encryption process is as presented below:



As shown above, the method is given the path of the file being encrypted. The program calls the file being encrypted (Plaintext file) with an OCALL.

encalve\_read\_part\_open\_file(&result, path, NULL, 0, &size\_input, 0);

do

{

encalve\_read\_part\_open\_file(&result, path, (char \*)input, SIZE\_AES\_BLOCK\_BYTE, &size\_input,1);

sgx\_sha256\_update((uint8\_t\*)input, size\_input, hash\_handle);

if (size\_input != 0)

{

if (size\_input < SIZE\_AES\_BLOCK\_BYTE)

finsihed = true;

sgx\_aes\_ctr\_encrypt((sgx\_aes\_gcm\_128bit\_key\_t\*)file\_key, input, size\_input, nounce\_counter, 16, cipher\_text);

encalve\_write\_end\_of\_open\_file(&result, new\_path, (char \*)cipher\_text, size\_input,1);

}

else

finsihed = true;

} while (!finsihed);

encalve\_read\_part\_open\_file(&result, path, NULL, 0, &size\_input, 2);

It then uses the file password given by the user, and calculates the hash of it with the hash function sha-256, and uses it as a key for the encryption of the master key using AES-GCM 128 (returns a 128 bits long bit stream).

sgx\_sha256\_hash\_t hash;

get\_sha256\_hash(file\_password, len, &hash);

This process is done in order to create new key which is strongly dependent on the user’s file password given, and on the master key.

With the result, a new key is created (128 bits long) which is used to encrypt the file the user wants encrypted.

sgx\_status\_t res = sgx\_rijndael128GCM\_encrypt((sgx\_aes\_gcm\_128bit\_key\_t\*)hash, main\_key, 16, file\_key, iv, 12, NULL, 0, &out\_mac);

if (res)

return res;//Error

During the encryption, it’s hash is calculated using the hash function sha – 256 with the master key. This is used in order to insure the integrity of the encrypted file. This is used later, in the decryption process.

sgx\_sha256\_update((uint8\_t\*)part\_from\_file\_input, size\_input, file\_hash\_handle);

A new .ens file is created in the path given by the user, with an OCALL, and the header containing the two IV vectors and nounces for the encryptions, and the hash of the encrypted file are injected into it. Finally, the encrypted file is also put in the encrypted cipher data of the file.

**Decryption of a File:**

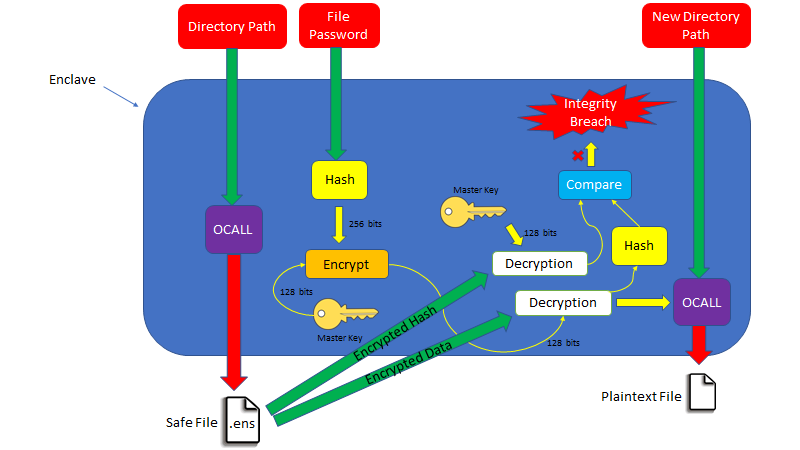
The decryption process is used to unlock files from a given vault.  
When the decryption process is activated the user must enter the following list of things:

1. The full path of the .ens file which is being decrypted.
2. The file password which was given during the encryption of the original file while locking it in the vault (besides the sealing the Intel SGX provides).
3. The directory path where the user would like the decrypted file to be.

As explained above, the decryption of a file must be done through the same vault used to encrypt that file.

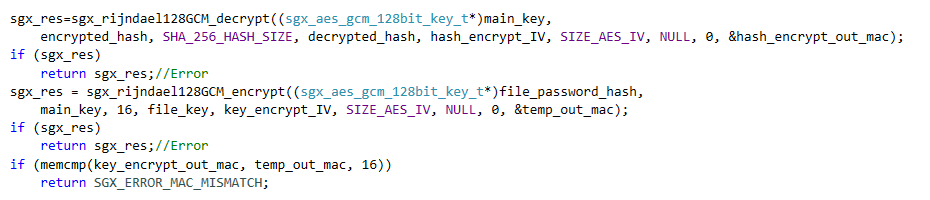
The extra password is there to solve the issue where if an attacker had access to your computer, but doesn’t have all the so this password is a security wall, in order to block the attacker from getting to the file.

The functionality of the decryption process is as below:



As presented above, the method is given the path for the .ens file. An OCALL is called to retrieve the .ens file with the path given.

With the retrieved data from the .ens file, which was retrieved earlier, the encrypted header is decrypted using the master key, with the decryption method for AES – GCM 128 bit.



In addition the file password is supplied. Its hash is calculated (with the hash function sha-256). This is used to encrypt the master key of the vault with the encryption method – AES – GCM 128 bit, which returns a new key as its result (128 bits long).

The new key created is used to decrypt the rest of the data from the .ens file which is the actual data from the original file, with the decryption method for AES – CTR.   
While decrypting the file, the hash is calculated (with the hash function sha-256), and when it is complete, it is compared to the hash decrypted earlier from the header in order to check the integrity – if there were any changes in the file in which case the hashes are very unlikely to come out the same, which would trigger an error.

If the integrity test passes successfully, an OCALL is called with the decrypted data, and the path for the new file given by the user, and the decrypted file is saved there, thus having finished the decryption process.

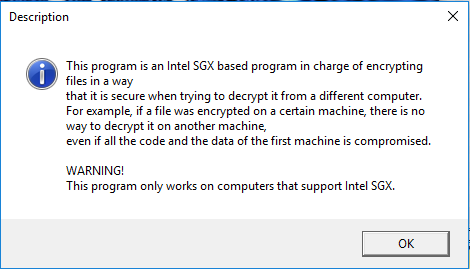
### Operating Instruction for the Program’s User Interface:

To begin with, when the program is first opened, the following window comes up:



As shown above there are the following options:

First of all, There is the information tab on the top. When opened, the following message box comes up



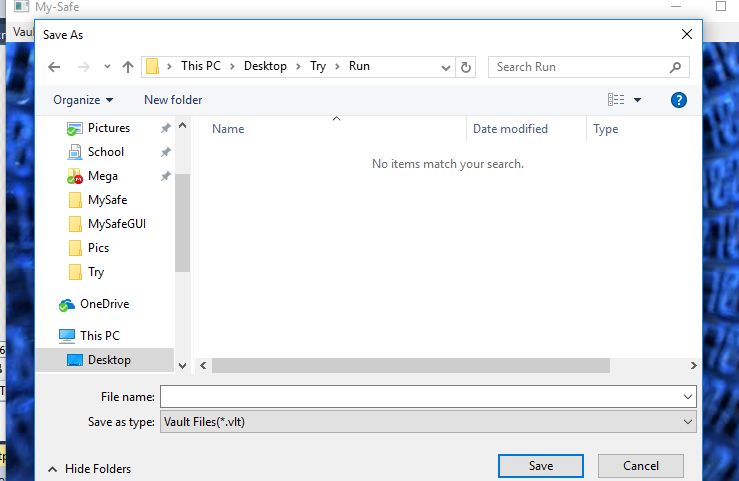
As shown, this gives a quick summary of the program’s abilities.

Another one of the options, is the first tab named “vault”. When clicked on the following options come up:

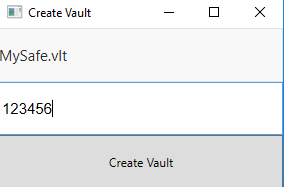


As shown above, there are the options to create a new vault, to load an existing vault and to close a vault.

When creating a new vault, first, a file chooser window is opened in order to choose a place and name for the new .vlt vault file as shown bellow



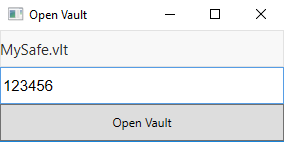
After choosing a place and name for the .vlt file, the following window comes up:



This window is for choosing a master password for the vault (and there’s an option to choose a new path for the .vlt file), and then you press the “Create Vault” button to finish.

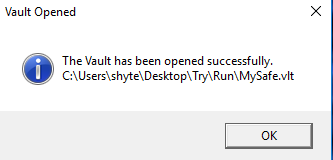
The second option is to load an existing vault (.vlt file) used before. To do this you choose the second option on the vault tab as shown above.

When chosen, the following file choosing window comes up to choose the vault you want to load. After choosing a file the following window comes up:



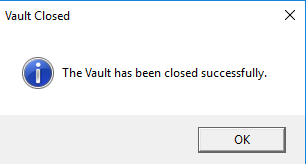
This window is for entering the master password which was given to the vault during the creation process. To finish the loading of the vault, you press the “Open Vault” button.

If it is loaded successfully, then the following window comes up:



The third option is to close a vault that is currently opened in the program. To do so you choose the third option in the vault tab shown above and choose the option to “Close Vault”.

It automatically closes the currently open vault, and the following message box comes up:



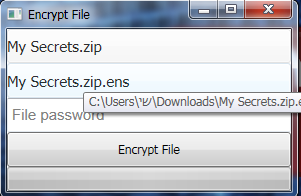
Besides the vault tab, you can also choose the file tab. When clicked, the following list is shown:



As shown above, there is an option to encrypt or decrypt a file with the currently opened vault.

To encrypt, you can choose the encrypt file tab, or just click the big green button in the main window.

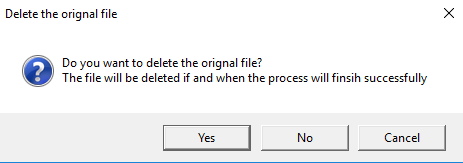
When encrypting a file, the user must choose a file to encrypt with a file choosing window. When chosen, the encryption window comes up:



Here you can change the file you chose if you want by clicking it, or choose a different place & name for the encrypted file (the default is the name of the original file being encrypted with the additional suffix .ens as shown in the example above).

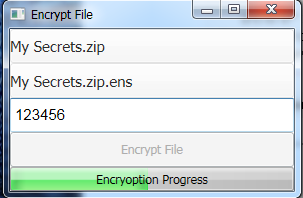
In addition, you choose the file password and to finish you press encrypt.

To encrypt it after choosing, you press the button, and the following window comes up –

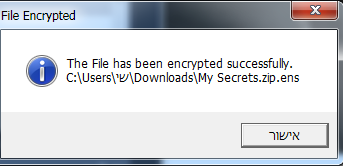


This is for choosing whether to delete the original after encrypting it or not.

During the encryption process, there is a progress bar which shows the percentage of it encrypted so far as shown below:



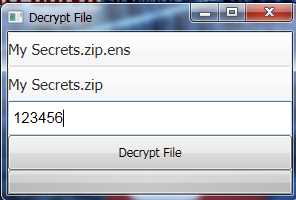
Finally, when the encryption is done the following message comes up:



And the encrypted file comes to be as shown below - C:\Users\Shalom\AppData\Local\Microsoft\Windows\INetCacheContent.Word\Screenshot_14.png

The second option is to decrypt a file. This can be done, through the file tab and then decrypt file choice in the list as shown above, or just pressing the big red button in the main window.

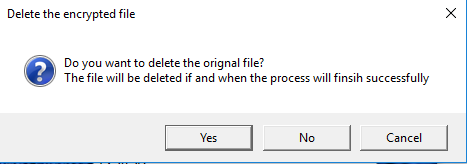
When decrypting a file, a file choosing window comes up to choose the file and then the main decryption window comes up



To change the file being decrypted press it and choose a different one, and the same goes for the decrypted file.

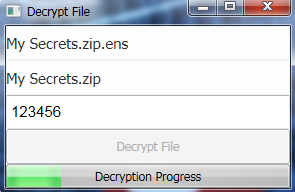
In addition, the user must enter the secret file password given to the file during the encryption process.

Finally, to finish you press the decrypt button, and then the following window comes up:

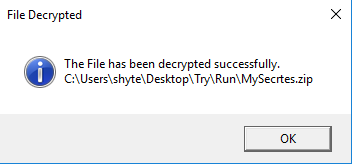


This window is for choosing whether or not to save the encrypted version of the file.

During the decryption process, there is a progress bar which shows the percentage of the file decrypted so far.



Finally, when it finishes, the following window comes up -



And the new decrypted file appears - C:\Users\Shalom\AppData\Local\Microsoft\Windows\INetCacheContent.Word\Screenshot_20.png

When the program is closed it automatically closes the currently open vault.