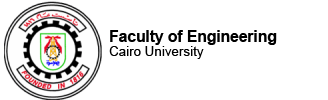
****

**Security Project #4**

**Submitted to: Dr. Samir Shaheen**

**Name: Ehab Rabie Amin Sec: 1 BN: 14**

**Project Overview:**

My implementation for this project is divided into 3 main constructs:

1. GUI module:

* Using python open source tool “**Pygubu**”, I have designed a simple UI which require two inputs from the user: his password, the desired input file (original, encrypted).
* The user should enter a password and input file path.
* There are some alerts to interact with the user for example:
  + Error message when there is no entered password or input file
  + Error message when there is incorrect password (in case of decryption scenario)
  + Success message when the encryption module is completed
* The UI contains a label at the bottom to inform user which phase is in progress.

1. Encryption module:

* When the user enters a password and a valid input file path, then click on **Encryption** button
* First, it’s required to have a DES symmetric key from the password so the solution is to translate the input password using an encryption method which is **pbkdf2**
* Second, by creating a counter of 64 bits length then using DES in **CTR** mode to encrypt the input file.
* Finally, the encrypted ciphertext is stored with the key of DES and the computed HMAC within output file with extension “**. enc**”.

1. Decryption module:

* When the user enters a password and a valid input file path, then click on **Open** button
* To ensure confidentiality the user must enter the same password he has entered in the encryption mode
* Then I extract the key and hmac value from the encrypted file
* To ensure authenticity, I calculate the hmac value of the ciphertext and compare it with the stored one. If they are the same then complete decryption process and open the decrypted file to the user automatically, otherwise as alert is prompted to the user to inform him that the content of the encrypted file has been changed and unsafe.

**HMAC algorithm:**

The working of **HMAC** starts with taking a message **M** containing blocks of length **b** bits. An input signature is padded to the left of the message and the whole is given as input to a hash function which gives us a temporary message digest. the temporary message digest again is appended to an output signature and the whole is applied a hash function again, the result is our final message digest **MD** which is the **HMAC** value of the input message. The applied hash function is **SHA1**.

In details, there is a key to the hmac algorithm which has to be no more than the block size in length. There is two different signatures Si, So the first one is produced with the padded key in **xor** operation with some constant called **ipad**. This signature is used to encrypt the message and the output is re-encrypted again to produce **HMAC** value of that message.

**Result analysis and conclusion:**

The encrypted file is vulnerable and cannot be decrypted back if the **key** or **hmac** field is changed even if the user entered the correct password