Jinali Sheth
Jms1333@nyu.edu
Application Security Class Assignment 1

Description:

This password manager is interactive. First, the user is prompted on whether they would like to access the credentials they have previously stored, or if they would like to store a new set of credentials. If the user chooses to:

- Access credentials :
    - User must enter an existing credential to recover.
    - If the username exists, the stored username and password are returned to the user in plaintext.
    - If the user would like, they can retrieve additional credentials they have stored.
- Add credentials :
    - The user must enter a username that does not exist.
    - Then the user must enter the password they would like to store along with that username. The user is prompted for their password twice in case they fat- finger the first password and mistakenly enter the wrong password.
    - A check is performed for same username.
    - Then the user selects which AES mode to encrypt their credentials with – ECB, CBC, or CTR
    - If the user would like, they can enter additional credentials to be stored.

Master Key:

Each encryption method has an inbuilt master key that helps in encrypting and decryption of the data. The key is 32 bytes long. I created a master key using the file generate_key.py. A master key is randomly generated using os.random(32). The randomly generated value could have been 16 bytes, but I chose 32 bytes since I felt it would be more secure; more characters, more randomization. Later on I added a master key of my own in the program, which is inbuilt.

Working:

Each encryption method has its own storage file where a username and password are encrypted and held. With this scheme of database it allows you to provide a username and password as input along with the selection of any type of encryption. It then stores this data in the user specified encrypted database file of your choosing. Before storing private information the password manager will check for any duplicate entries and will deny write access if such entries exist. It also allows you to decrypt a usernames password if a user decides to retrieve it at a later time. And lastly the password manager has rich help menu that provides guidance for the user selected criteria.

Select different modes ecb, cbc, ctr any of the 3 all are working here. Followed by –encryption or –decryption mode to encrypt or decrypt data followed by –username and a –password. These are separate flags that I have set for the encryption and decryption scheme to decode the data stored using dictionary structure use –decryption flag. If we try to enter the username again it will prompt as duplicate user. We have verified in the database file that there are no duplicate entries in it.

We perform this same action for every mode we have probably the best mode of all is the CTR mode because it converts the block ciphers into stream ciphers as a result of which hacking of the ciphers would be difficult.

Primitives:

AES is a block cipher. That means, it encrypts/decrypts in blocks of 128 bits. The key used for the [en|de]cryption is 128, 192 or 256 bits. So, this means that your data has to be divisible by 128bits and your key must be 128, 192 or 256 bits. Pretty tough to meet these requirements considering we're dealing with binary/text files that will be of some random length? Well, that's where padding comes in.
In order to make the file's length divisible by 128b(16B) we have to pad it. The most used padding scheme is PKC7. Simply append n < 16 bytes to the file with value n to meet the 16B block size. So, the workflow now would be, pad -> encrypt -> decrypt -> unpad.

There's one more thing - IV - Should also be 16B. It's a common practice to have this at random, prepend it to the file on every encryption. We'll assume a static IV now for the sake of simplicity.
Library used is Pycrypto that helps in importing the above primitives.

Brute force Difficulty:
Even if the master key is decoded the attackers still cannot get into the passwords as the combination is of unique characters with symbols and numbers as a result of which the password strength is the strongest. And the randomization of every-time a password is entered in the program will make it safer. Also if you think that you are being targeted by an adversary that will break the encryption, it is not possible as AES 128 breaking will require 2^128 bit cycles to have a collision and to break it.