Alan Chen

ECE 297 Design Project

AES Cryptoware

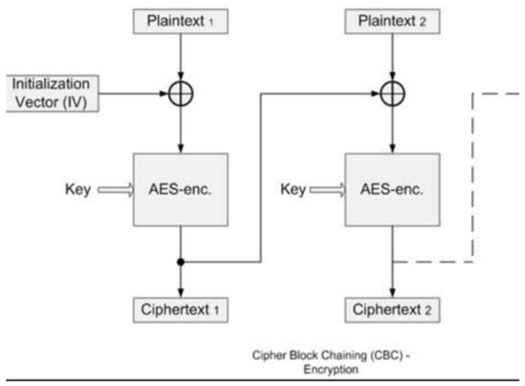
12/10/2018

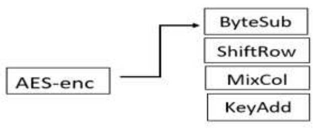
Overview

Advance Encryption Standard is one of the most used symmetrical encryption algorithms. To learn more about the systematic process of AES, I implemented AES-Cipher Block Chaining on an Adafruit M4 Metro board.

About AES-CBC

After Data Encryption Standard (DES) started to some vulnerabilities, the National Institute of Standards and Technology called for a new and more secure algorithm in 1997. 15 submissions were made and 5 proceeded to the final round for testing. In 2000, Rijndael was selected as the new AES. Unlike its predecessor, AES is an open and transparent process and its ciphertext is depended only on the key, plaintext, and initialization vector (IV). AES encrypts data by structuring it into a 4 by 4-byte block (128 bits). The process starts off with getting XOR-ed with the IV which is also 16 bytes. Next, it goes through ByteSub which is a lookup table of S-boxes. Then, it goes through ShiftRow, MixColumn, and lastly, it gets XOR-ed with the key. That is considered 1 round and the next round, it will have the ciphertext of round 1 in place of the IV. The number of rounds and the complexity of the ciphertext depends of the length of the key.





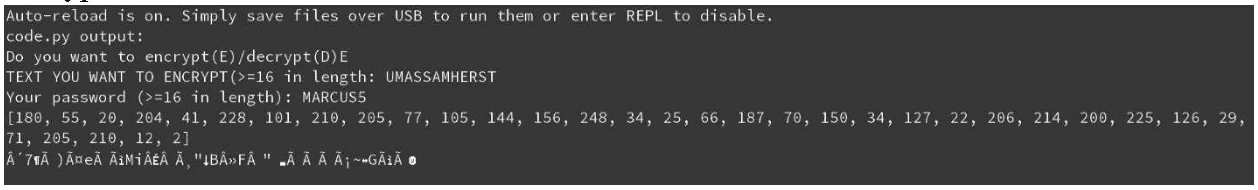
Key Sizes (bits)	Rounds
128	10
192	12
256	14

My Project

My plan was to use PyCrytodome, an encryption library with AES methods, and transfer that into the M4 board. Although I was able to successfully run it on my computer, I was not able to transfer it. First, I thought it was due to its size, so I started to eliminate unnecessary files, by tracking down all the imports lines. After that, I still could not get it to work due to the nature of how PyCrytodome was written in. For example, the library depends on the binascii conversion tools which is nonexistent on the M4 board. Finally, I decided to source a pure python code that depends only on the simplest key words which will be optimal for my purpose. After some changes, the code was able to function properly.

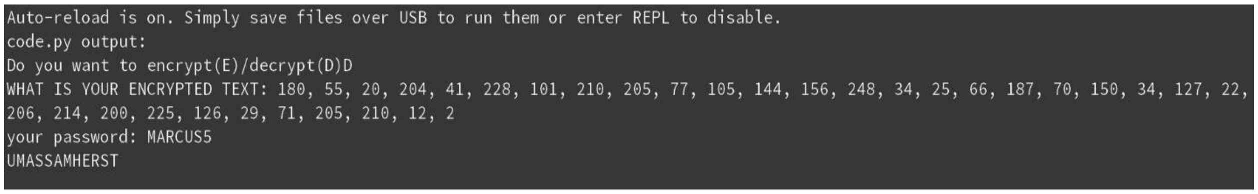
For a normal AES-CBC it can take in many key lengths however for simplification purposes, my project only takes maximum of 128 bits and anything lesser will be resolved with padding. An alternative/better way would be to hash the key. If the plaintext is not correct length it will be padded. I used a random generator for the IV however it is also possible to integrate it within the key. The IV is essential for decryption so I put that added that with the ciphertext. One benefit is that the hacker does not know where to look within the ciphertext. Lastly, Mu cannot read the full range of UTF-8 so instead I used an array of numbers that represent the ciphertext.

Encryption:



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Do you want to encrypt(E)/decrypt(D)E
TEXT YOU WANT TO ENCRYPT(>=16 in length: UMASSAMHERST
Your password (>=16 in length): MARCUS5
[180, 55, 20, 204, 41, 228, 101, 210, 205, 77, 105, 144, 156, 248, 34, 25, 66, 187, 70, 150, 34, 127, 22, 206, 214, 200, 225, 126, 29,
71, 205, 210, 12, 2]
Â´7¶Ã )Ã¤eÃ Ã¡MiÂ£Â Ã¸"↓BÂ»FÂ " ▁Ã Ã Ã Ã¡~↔GÃ¡Ã ☻
```

Decryption:



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Do you want to encrypt(E)/decrypt(D)D
WHAT IS YOUR ENCRYPTED TEXT: 180, 55, 20, 204, 41, 228, 101, 210, 205, 77, 105, 144, 156, 248, 34, 25, 66, 187, 70, 150, 34, 127, 22,
206, 214, 200, 225, 126, 29, 71, 205, 210, 12, 2
your password: MARCUS5
UMASSAMHERST
```

Sources:

https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard