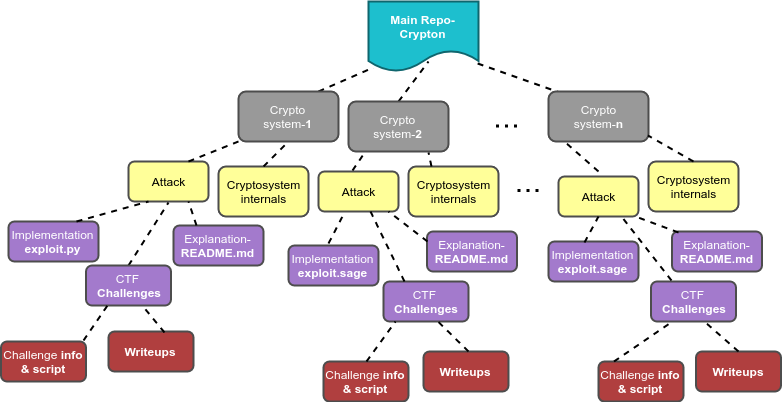
Lab notes

Contents

[Introduction 1](#_Toc137304526)

# Introduction

* library Crypton
* Check out the library on GitHub: <https://github.com/ashutosh1206/Crypton>
* platform to learn and practice Offensive and Defensive cryptography for people interested in this field .
* this lib talks about secure implementation of the crypto systems
* Matasano Crypto Challenges have provided a very nice headstart to learning crypto (not updated).
* by means of
  + analysing how various cryptosystems work
  + crypto system internals
  + math behind the concepts
* The library consists of
  + Explanation
  + implementation
* of all the existing attacks on various
  + Encryption Systems
  + Digital Signatures
  + Hashing Algorithms
* along with example challenges from “Capture The Flag” and their respective writeups.
* As of now the library consists of around 45 attacks



* Block Cipher is a method of encrypting text using an algorithm which takes in a
  + key
  + (iv in some modes, as you will see),
  + one block of data at a time,
* unlike stream ciphers which encrypt one byte of text at a time.
* Identical bytes of plaintext don't get encrypted into identical bytes of ciphertext in the case of block ciphers.
* Also, we must remember that even if one byte in a block of plaintext is changed/flipped, then the entire corresponding block of ciphertext changes in case of block cipher encryption.
* Key and iv are pseudo-random generated strings and their size depends upon the encryption algorithm being used.
* The block-size too, depends on the encryption algorithm being used (For example, block size for AES is fixed and equal to 16 bytes).
* detection of blocksize in a block cipher.
* Suppose a program is running on a server we can nc to.
* The server takes in input from the user,
* pads it to make it a multiple of blocksize,
* encrypts it using a block cipher encryption algorithm
* gives the ciphertext as the output to the user.
* Using this,
* we need to get the size of the block being used in the encryption algorithm.
* Suppose we send plaintext of size 1 byte and we get a ciphertext of size x = 16.
* We then plaintext of size 2 bytes and check if the length of ciphertext is same as the length of previous ciphertext returned that is x here.
* We can keep increasing our input by 1 byte and checking if the length of ciphertext returned is equal to the length of the ciphertext previously returned.
* In case lengths do not match,
* then the blocksize is simply equal to the difference between the lengths of the two ciphertexts(the present and the previous one).

Why does this happen? Remember from the padding section that when plaintext length becomes a multiple of blocksize, then the padding algorithm adds an entire block to the plaintext due reasons already discussed in Padding section. So, we keep adding one byte to our input and noting the ciphertext length for each of our inputs. As soon as our input becomes a multiple of blocksize, an entire new block is added and hence the length of ciphertext changes. We can then compute blocksize = difference between the lengths of two ciphertexts(present and previous). Read about it in detail on my blog here

* This section will enable us to determine if the encryption of each block is dependent on the encryption of block next to it or not.
* Determining this is fairly easy,
* recall that in the case when the encryption of each block is independent, then two blocks having same data will have same data in their respective ciphertext blocks.
* for doing this, first we need to know the blocksize being used for encryption,
* which we can determine by the method discussed in Block size detection.
* Then, we can send two blocks of plaintext having same data and check if their corresponding ciphertext blocks match,
* if yes, then the mode of encryption is definitely ECB mode, otherwise the mode won't be ECB.
* Read about it in detail on my blog here

# Detecting IV in CBC mode

Prerequisites:

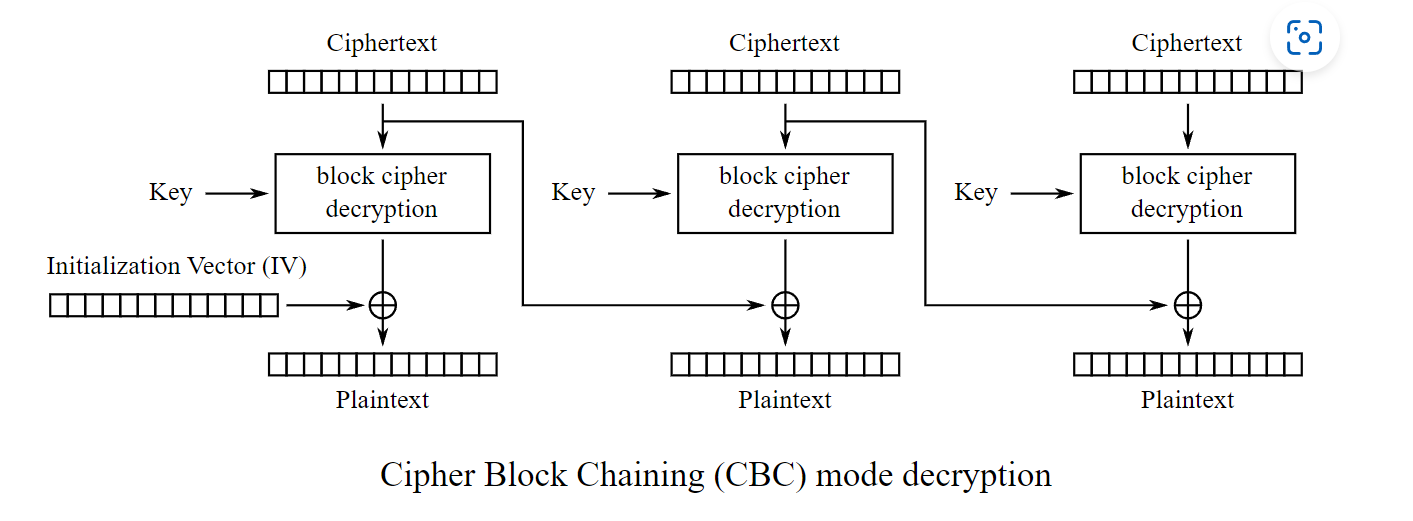
* This section will help us get the value of the Initialisation Vector (IV) in CBC mode when its value is not known.
* This exploit is particularly helpful in attacks such as Padding Oracle Attack, which requires knowledge of IV to get the value of first block of plaintext.
* Consider a scenario where there is a program running on a server which encrypts/decrypts data given to it as an input.
* The encryption/decryption is based on a block cipher in CBC mode.
* We only have access to input and output and we need to know the value of IV using this.

Suppose we send an input of size > 32 bytes. Then after padding the minimum size of our plaintext input will be 48 bytes. We want this as a minimum due to reasons which you will come to know later.

As of now, consider the conditions.

We will get a ciphertext as an output, let it be assigned as ciphertext. For our attack, we will select first three blocks of ciphertext because IV is directly used in the first block.

decryption



We will write down equations for each plaintext block: (First block of plaintext as p1, second as p2 and so on)

p1 = D(c1) xor iv

p2 = D(c2) xor c1

p3 = D(c3) xor c2

When c1 = c3 and c2 is an empty block, i.e. c2 = "\x00"\*blocksize, then

p1' = D(c1) xor iv

p2' = D("\x00"\*blocksize) xor c1

p3' = D(c3 = c1) xor "\x00"\*blocksize = D(c1)

We can now simply calculate iv as p1' xor p3'. So we call the decryption oracle and give the ciphertext input as c1 + "\x00"\*blocksize + c1. We can now XOR the first block of plaintext output and the third block of plaintext output to get IV.

I have written a script in python to illustrate working of the above exploit. You can check it here- example.py.

# CBC Bit-Flipping Attack

* Bit Flipping Attack requires CBC mode
* This attack is usually in scenarios where the encryption function takes
  + Send some input as a payload
  + prepends a random string
  + appends another string
  + encrypting it.
* There are cases where the encryption function escapes some characters or character sequences from the payload supplied, before encrypting it.

|  |
| --- |
| def encrypt(payload):      obj = AES.new(key, AES.MODE\_CBC, iv)      for i in xrange(len(payload)):          if payload[i] == ";" or payload[i] == "=":              payload = payload.replace(payload[i], "?")      str1 = "comment1=cooking%20MCs;userdata=" + payload + ";comment2=%20like%20a%20pound%20of%20bacon"      str1 = padding(str1)      ciphertext = obj.encrypt(str1)      return ciphertext |

* Value at 2. is fixed
* BlockCipherDecryption(B[n]) = A[n] xor P[n] 🡪 2
* We know that we want P[n] in 1 to be of our desired value(Let this be ‘PD’) whereas P[n] in 2.
* is the actual value of the plain text (Let this be ‘PA’) after the decryption of cipher text without flipping any of them. So, A[n] then becomes:
* A[n] = PD xor A[n] xor PA
* A[n] = A[n] xor (PD xor PA)
* PD xor PA —> XOR of the desired plain text byte with the actual byte present in the plain text block.
* So, we simply XOR the result (PD xor PA) with the actual value of the A[n]. The result is the value we should give at that byte in the cipher text block previous to the plain text containing “?admin?true?”. Repeat this for all other blocks.

# References

* Practical crypto challenges and labs

[ashutosh1206/Crypton: Library consisting of explanation and implementation of all the existing attacks on various Encryption Systems, Digital Signatures, Key Exchange, Authentication methods along with example challenges from CTFs (github.com)](https://github.com/ashutosh1206/Crypton)

* Authors blog post

[Announcing Crypton- An educational library to learn offensive and defensive crypto(graphy)! – masterpessimistaa (wordpress.com)](https://masterpessimistaa.wordpress.com/2018/08/12/announcing-crypton-an-educational-library-to-learn-offensive-and-defensive-cryptography/)