Assignment 3: Block Ciphers

CSCI3381-Cryptography

Due February 21, 2017

You're not expected to do all the problems: 100 points is a 'perfect' score. Additional points will be put into an extra-credit bank. As usual, there are written problems and there are coding problems. One section of written problems is more 'math-y' than the others.

1 Block Cipher Modes of Operation

6 points per part, 36 total. All your answers should contain brief, but careful, explanations. If you have installed a tool like `pycryptodome` that contains implementations of block ciphers, you can test some of your answers, but this is not necessary, and you still need to explain any phenomenon that you observe. Drawing diagrams is a very good idea. *(a)* Suppose a message of 100 plaintext blocks is being encrypted with

CBC mode. Suppose that, prior to encryption, one bit of the tenth plaintext block is inadvertently changed. How many blocks of plaintext, after decryption, are certain to be correct?

(b) Same setup as part (a), but this time, one bit of the tenth *ciphertext* block is changed after encryption but before decryption. How many blocks of plaintext, after decryption, are certain to be correct?

(c) Same set up as (b), but this time the tenth and eleventh ciphertext blocks are swapped before decryption.
(d) Suppose we encrypt two multi-block messages with CBC mode but use the same IV each time. What information about the plaintext is leaked by the ciphertext? (You also might try to imagine a scenario in which this information is useful to an attacker.)

(e) Suppose we encrypt two multi-block messages with CTR mode but use the same IV each time. What information about the plaintext is leaked by the ciphertext? (You also might try to imagine a scenario in which this information is useful to an attacker.)

(f) Below is the transcript of a sequence of encryptions and decryptions in CBC mode, carried out using the library `pycryptodome`. You should know that you cannot use the same cipher object in CBC mode for both encryption and decryption, because the object maintains state, keeping track of the last ciphertext block of the preceding encryption/decryption to use in the next encryption/decryption. Thus

```
     cipherobject.encrypt(a)
     cipherobject.encrypt(b)
```

is equivalent to

```
     cipherobject.encrypt(a+b)
```

and likewise for decryption.

```
>>> message='Super Bowl LI:The New England Patriots beat the Atlanta Falcons.'
>>> iv='0123456789abcdef'
>>> key=' Boston College '
>>> cipherobject=AES.new(key,AES.MODE_CBC,iv)
>>> c1=cipherobject.encrypt(message)
>>> c2=cipherobject.encrypt(message)
>>> c1 == c2
False
>>> cipherobject2=AES.new(key,AES.MODE_CBC,iv)
>>> cipherobject2.decrypt(c2)
'\x043\xfc\x9cg\xa6\x1e\x80\x1bV\xb5\x9bu\xa4m9e New England Patriots beat the Atlanta Falcons.'
>>> cipherobject2.decrypt(c2)
'\x82\xddGc\x18\x84\x8c\xb6\xc8SL\x97\r\x0ed\xbae New England Patriots beat the Atlanta Falcons.'
```

Because of the aforementioned 'statefulness', it is no surprise that when we perform two successive encryptions of the original plaintext, we get two different ciphertexts. But why, when we build the decryption object and use it to decrypt the *second ciphertext*, is the end of the plaintext intact? Explain this behavior.

2 Modifications to Substiution-Permutation Networks.

8 points per part, 24 total This problem is about the internal structure of block ciphers. The model here is the generic substitution-permutation network described in the notes. (While you don't need to explicitly consider AES in this problem, you should be aware that conclusions also apply to AES, where the mixing permutation is replaced by multiplication by an invertible matrix.)

(a) Suppose our encryption algorithm uses only *one round* of the SP network. Explain how we can completely decrypt any block given a single known plaintext-ciphertext pair of blocks. Describe the procedure in detail.

*(b)*Suppose that instead of alternating AddKey, Sub, and Mix in the r rounds of our substitution cipher, we do all r rounds of AddKey, then all r rounds of substitution, then all r rounds of the mixing permutation, in that order. Show that if we have a single known plaintext-ciphertext pair of blocks, we can decrypt any block.

(c) Suppose we got rid of the S-boxes, and just retained the AddKey and Mix phases of each round. Show how, given a single known plaintext-ciphertext pair of blocks, we can decrypt any other block of ciphertext. HINT: Show that this forces

$$E_K(M_1) \oplus E_K(M_2) = \pi(M_1 \oplus M_2),$$

where π is some permutation of the bits, for any two plaintext blocks M_1 and M_2.

3 Block Cipher Statistics.

8 points per part, total 40 An obvious requirement of cryptographic systems is that if you have two different plaintexts $m_1 \neq m_2$ and encrypt them under the same key k, then the ciphertexts have to be different: $E_k(m_1) \neq E_k(m_2)$. However, if you encrypt the same plaintext m under two different keys $k_1 \neq k_2$, nothing in the definition requires $E_{k_1}(m) \neq E_{k_2}(m)$.

In this problem you will study the statistics of such 'collisions'. Some pointers to relevant probability tools that you may have forgotten are given in each part. *In general,* use K to represent the number of bits in the key, M to denote the number bits in a block, and then use your general answers to obtain specific numbers for DES and AES.

(a). Fix a plaintext block m and a candidate ciphertext block m'. What is the probability that there is *no* key k such that $E_k(m) = m'$? HINT: Choosing a random key k to obtain E_k should be like choosing a random permutation of the set of blocks. Thus, choosing one random key and encrypting m under it to see if you get m' is , or should be, the same as choosing a random block and seeing if you get m'. Then think of multiple independent trials of this experiment.

(b) Use the result of *(a)* to estimate, for each block m, the number of ciphertext blocks that cannot be gotten by encrypting m. Do this for each of our three example block ciphers. What does this tell you about the likelihood of the existence of colliding keys for a given plaintext m, at least in the case of AES? (HINT: It is very helpful to know that $(1 - 1/n)^n$ is close to e^{-1} for large n, and that more generally $1 - x$ is well approximated by e^{-x} for x close to 0. This problem might give a rather inconclusive result for DES unless you are really clever about it.)

(c) This is an extension of parts (a) and (b) above. Estimate the number of ciphertext blocks m' such that *exactly one* key encrypts m to m', exactly two keys, *etc.* (HINT: This uses the Poisson distribution.)

(d) Parts (a)-(c) should tell you that if you pick a plaintext block at random, there are likely to be many pairs of distinct keys that encrypt it to the same ciphertext block. Suppose I go looking for such a collision, testing keys at random until I discover a match. About how many keys will I have to test to have better than even odds of finding a colliding pair? (HINT: Do you know the problem about the chances of two people in a room having the same birthday?)

(e) What we've been showing is that although the function

$$m \mapsto E_k(m)$$

is one-to-one for each fixed key k, the function

$$k \mapsto E_k(m)$$

for a fixed is block m is almost surely not, at least not if we want our keys to behave like random permutations of the set of plaintext blocks. This may come as a bit of a surprise after the one-time pad. Estimate (do this just for AES) the probability that this map is one-to-one. (HINT: Stirling's formula for $n!$). The answer, as you might expect after all this, is 'very small'. But is it very small like one in a million, or very small like one in a googol, or like one in a googolplex?

4 Computer Problems: Padding Oracle Attack on CBC

(8 points per part, 40 total)The padding oracle attack on CBC is described in the notes, and in many, many other documents. I have set up two web pages that use CBC encryption and which, because of the way that are implemented, can be used as padding oracles. One is a 'captcha', an environment in which such padding oracles have been observed in real life. the second, less elaborate one, simply presents you with a ciphertext, which is the encryption of a long word chosen at random from the dictionary.

```
http://cscicrypto.bc.edu:8080/captcha
http://cscicrypto.bc.edu:8080/dictionary
```

Access to these websites is restricted: You can only reach them from a BC IP address, so if you are working from off-campus, you need to use the VPN tool provided at the BC software page.

Here's what you need to know about the ciphertexts displayed in the text field on the dictionary website: Each one is the encryption, under CBC mode, of a randomly chosen word from the online dictionary. The same key, and in fact the same IV, was used for each encryption. The choice was restricted to long words, those having at least 12 letters. The IV was prepended to the ciphertext before posting to the page. Thus the length of the byte sequence you see is at least 12 bytes for the plaintext length, plus the length of the IV, plus the padding. The encryption was done using either DES or AES: You have to figure out (part (a)) which is which.

(a) By repeatedly refreshing the page, you can examine multiple ciphertexts, and by doing so, determine whether DES or AES was used.

By clicking the submit button, you can see how the website responds to an incorrect guess for the plaintext. By altering a digit of the ciphertext, you can also see how it responds to an incorrectly padded ciphertext. Of course, you do not want to be sitting there entering thousands of candidate ciphertexts in the form and noting the response, so I have provided two simple functions to help you automate the process: the first grabs the ciphertext from a the site and returns it. The second takes a ciphertext and a guess of the plaintext and returns the resulting HTML string. Looking at this return value will tell you whether there was a padding error.

The remaining parts of the problem ask you to write generic functions for implementing stages of the padding oracle attack. They should work with any correctly-formed ciphertext, but in your writeup you should choose a particular ciphertext that you find on the site and describe the results you get using that ciphertext.

(b) Write a function that determines the length of the padding on a correctly-padded ciphertext.

(c) Write a function that determines the last byte of the plaintext that is not part of the padding, assuming that the plaintext length is not an exact multiple of the block length.

(d) Write a function that determines the entire last block of the plaintext, including the padding, given the ciphertext as input.

(e) Finally, write a function that decrypts any given ciphertext.

5 What to Hand In

Most of this is written homework. For the computer problem, you should include functions

```
pad_length(ciphertext)
last_nonpad_byte(ciphertext)
last_block(ciphertext)
padding_oracle_decrypt(ciphertext)
```

Your writeup should explain your solution to part (a) of the last problem, and the results you were able to obtain with the sample ciphertext you chose.