Total points: 80 (+10 bonus)

Assignment 2

Due: 11:59 PM, 10th July (Wed)

Written assignment

1. [15 points] In this assignment, we will learn about ciphertext indistinguishability, which underpins the theoretical basis for proofs of security of cryptograpic schemes.

   Ciphertext indistinguishability under the chosen plaintext attack (IND-CPA) roughly means that an attacker could not extract any information from a ciphertext, even if they could ask the encrypter to encrypt texts under the same key. IND-CPA is defined with the following game (slightly simplfiied) between an **adversary** (who claims they can break the cipher) and a **challenger** (who wants to verify the adversary's claim) for symmetric key schemes:

   1. The challenger has a secret key K of n bits used for all encryption/decryption in this game. The adversary does not know K.
   2. The adversary may ask the challenger to encrypt any number of texts that is at most polynomial in n, and send the reults back to the adversary. The adversary may also perform any polynomial time computation.
   3. The adversary sends two plaintexts of the same length, P_0 and P_1, to the challenger.
   4. The challenger randomly chooses to encrypt one of them, and sends the ciphertext back to the adversary.
   5. The adversary may further ask for (polynomially bounded) encryptions and perform computations.
   6. The adversary makes a guess. If the adversary guesses correctly with non-trivially greater than 50% chance, the adversary wins and the scheme is not IND-CPA secure.

   The scheme is IND-CPA secure if no such adversary exists. IND-CPA is sometimes called the "master property" because it covers a broad range of potential content leakage from the ciphertexts. We will examine some of them. Note that the definition is slightly altered to discourage the use of AI, so you must use the above definition in answering the following.

   (a) [3 points] Prove that AES under ECB mode is not IND-CPA secure even if messages can only be one block large. Hint: After choosing P_0 and P_1 and receiving the ciphertexts, what can you ask the challenger to encrypt?

   (b) [4 points] Suppose the attacker can always notice ciphertext repetition: given $Enc(M)$ and $Enc(M')$, the attacker always knows if M and M' share a word or not. Prove that such a scheme is not IND-CPA secure.

(c) [4 points] Suppose that an attacker can always correctly guess the first byte of any M given $Enc(M)$. Prove that such a scheme is not IND-CPA secure.

(d) [4 points] Does the CRIME attack in the programming part of this assignment work against IND-CPA secure schemes? Why or why not? Hint: One of the steps of the adversary-challenger game precludes the CRIME attack.

Programming assignment

Breaking Cryptography

In this assignment, we will continue to write programs to automatically break some weak ciphers. Please make sure to read the submission instructions carefully.

(a) Two-Time Pad [25 points]

Open citext.zip from the course website and fetch your two ciphertexts by cross-referencing your computing ID with ttp-shorten.txt. Those two files were encrypted using the same one-time pad. They are exactly 600 bytes each, and they both come from popular English Wikipedia articles. The topics of these articles include popular shows and movies, celebrities, notable political figures, countries, and companies. Find the contents of both files using crib-dragging, and submit them as `ptext0` and `ptext1`. You can flip around ptext0 and ptext1.

You may assume the plaintext to consist only of ASCII characters with the following byte values, all ranges being inclusive of both ends:

- Symbols: 32 to 41, 44 to 59, 63, 91, 93.
- Capital letters: 65 to 90.
- Small letters: 97 to 122.

The ctext file was derived by XOR'ing the plaintext (as ASCII bytes) with the key. Each byte of the ctext file would be the XOR of the corresponding byte of plaintext with the corresponding byte of the key (written as bit strings). If you cannot find the full texts, submit as much of the text as you can find.

(b) CRIME Attack [40 points]

To save bytes on the wire, plaintexts can be compressed before encryption for delivery. In particular, HTTP headers were compressed. One particular HTTP header is the Cookie header, which can be used to deliver authentication cookies; therefore, an attacker who is able to steal this cookie will be able to impersonate the victim.

The compressed packet size can leak information. In particular, if the HTTP header contains a duplicate string to the cookie, then the header will be smaller than expected as duplicate strings are compressed. If the user is visiting the attacker's site, the attacker

can start guessing the cookie by manipulating the header to contain such a string. This allows the attacker to steal the cookie without breaking the encryption scheme.

Rizzo and Duong who developed the CRIME (Compression Ratio Information-Leakage Made Easy) Attack to expose this leakage. We will recreate it here.

Attack scenario

1. The attacker needs two capabilities to steal the cookie for `example.com`:
   - The victim is tricked into accessing the attacker's web page. (The victim does not need to do anything.)
   - Eavesdropping capability on the victim's connection to `example.com`.
2. On the attacker's site, the attacker (using Javascript) asks the victim to obtain the resouce `example.com/<Guess>`. This triggers the corresponding `GET <Guess>` request with the Cookie sent to `example.com`
3. The attacker can observe this packet's size on the wire with eavesdropping.
4. Based on the size of the encrypted packet, the attacker determines if the guess is (partially) correct.

Compression

The main program for HTTP header compression is gzip. gzip uses the Deflate algorithm, which proceeds in two steps:

1. If there are duplicate strings, Deflate uses LZ77 to replace them with two values: a displacement and a length. The displacement tells us where to find the string and the length tells us how long it is. For example, the string

   ```
   Never forget what you are, for surely the world will not.
   ```

   can be replaced with:

   ```
   Never forget what you are, (-21, 3) surely the world will not.
   ```

   Depending on the compression level, LZ77 scans for duplicate strings within a limited range.
2. For the rest of the text, use Huffman encoding. We are not concerned with Huffman encoding in this assignment; we want to use duplicate strings to trigger LZ77.

Assignment

For this assignment, we will use an encryption oracle rather than a web server. The encryption oracle is provided in oracle.py; it encrypts texts and provides you with the answer. This simulates both of the attacker's capabilities at once.

- The cookie will always be preceded by the "Cookie:" string. Note the lack of a space.
- The cookie is written in Base64 encoding. The number of bytes can vary.

- Your goal in this assignment is to obtain the Cookie by repeatedly calling the oracle. Write the cookie to a file called "cookie.txt". Do not include the "Cookie:" header and do not include any other text in cookie.txt.
- oracle.py will be changed during grading to include a different key and IV as well as to defeat attempts to read its contents directly. The encrypt() function's signature will not change. oracle.py will always be in the same directory as your code, which must be named guess-cookie.py.

Guessing techniques

Here are some pointers to help you write your solution.

- You should guess the cookie byte by byte. If `"abc"` are indeed the first characters of the correct cookie, then your next guess should be `"abc<x>"`, replacing `x` with a possible Base64 character.
- The fixed format of the Cookie can help you in your attack. You can ensure that your first guess is long enough to trigger the duplicate string detection of LZ77.
- The limited range of LZ77 can be manipulated for your attack. If your guess is outside the range, then placing it inside the range will create a different compressed size if it is correct.
- The only possible characters of the cookie are valid Base64 characters.

You can read Rizzo and Duong's original presentation for more information.

Grading

Your code will be tested on ten different cookies. Your grade will be:

```
round(Correct outputs/2) * 8
```

The number of correct outputs is rounded up so that one mistake would still result in full marks.

Coding errors that cause the output to be malformed or incorrect for all of your answers can still be graded at a penalty, but you will need to contact me for remarking.

An additional bonus grade is based on the number of times you call the oracle. Fewer times is better. Only submissions with at least 70% correct will be eligible:

Top 50% of submissions	5 points
Top 20% of submissions	7 points
Top 5% of submissions	10 points

Submission instructions

All submissions should be done through CourSys. Submit the following files:

- `a2.pdf`, containing all your written answers. Make sure not to accidentally submit this file.
- `ptext0` and `ptext1`, for part (a) of the programming assignment.
- `guess-cookie.py`, for part (b) of the programming assignment.

Please put them into one ZIP file, and do not include any extra folders.

To run guess-cookie.py, I will call `python3 guess-cookie.py`. The output should be in `cookie.txt`.

You may use another language. However, you will have to write your own oracle. Please submit the oracle to me at least 3 days before the assignment deadline so I can verify its correctness.

Keep in mind that plagiarism is a serious academic offense; you may discuss the assignment, but write your assignment alone and do not show anyone your answers and code.