Total points: 75 (+7 bonus)

Assignment 2

Due: 11:59 PM, 10th November, 2017 (Fri)

Written assignment

1. There are mistakes in the following uses of cryptography. Point out what is wrong, suggest a correct use, and explain.
   (a) [4 points] Alice wants to communicate with Bob, her personal friend, securely. She generates a public/private key pair using RSA (128 bits), sends Bob the public key in person, and then Bob uses the public key to encrypt secret keys for AES (128 bits) in CTR mode and sends them to Alice.
   (b) [4 points] To store passwords securely, a website administrator uses AES encryption with a secret key and a 64-bit IV to encrypt all users' passwords. A CRC32 checksum is used to ensure correctness against random bit flip errors.
   (c) [4 points] A website has a TLS certificate, which is a CA's RSA signature of the website's private ECC key. After the visiting web user verifies the ECC key, all further communication between the client and the server is encrypted and decrypted with this key.

2. [9 points] For each of the following network-based attacks in the left column, find the most fitting network defense in the right column. Explain why.

Attack	Defense
IP spoofing	Proxies
Eavesdropping	Deep Packet Inspection
Teardrop attack	Ingress/egress filtering

3. [15 points] Two files, `ctext0` and `ctext1`, have been sent to you by e-mail. Those two files were encrypted using the same one-time pad. They are exactly 400 bytes each, and they both come from English Wikipedia articles. Neither file ends with a newline, and all characters are ASCII characters with byte values between 32 and 126. Find the contents of both files using crib-dragging, and submit them as `ptext0` and `ptext1`. (The order does not matter.) You may write your own code to implement crib-dragging or use someone else's code with permission; remember to credit them.

4. [13 points] When a client C accesses server S through Tor, she usually builds a circuit of three nodes: N_1, N_2, and N_3. A connection is established as follows:

$$C \to N_1 \to N_2 \to N_3 \to S$$

   N_1 is also known as the entry node or the guard node, and N_3 is also known as the exit node. Visit `metrics.torproject.org` to answer the following questions:

(a) [3 points] Give the total amount of advertised bandwidth of relays with the "Guard" flag but not the "Exit" flag and relays with the "Exit" flag but not the "Guard" flag on 2017-01-01. Which is more? Give one reason to explain this phenomenon.

(b) [4 points] Give the median download rate of a file (in bits per second) for a 50 KiB file and a 5 MiB file to the `op-hk` onion server on 2017-06-01. Can you explain the difference?

(c) [3 points] What is a disadvantage of using three nodes in a Tor circuit instead of one node? What is an advantage of doing so?

(d) [3 points] For all countries with more than 1,000 daily Tor users, which is the country with the greatest ratio of bridge users compared to relay users on 2017-06-01? Give the ratio, and suggest why this is the case.

(e) [3 points (bonus)] Which country has the most number of users by percentage of population? Can you explain why?

Programming assignment

Padding Oracle Attack [26 points]

AES — the standard block cipher in use today — had a padding algorithm that introduced vulnerabilities when combined with CBC (Ciphertext Block Chaining). In this assignment, we will investigate why it was insecure. In fact, the attacker can arbitrarily decrypt and encrypt in AES without knowledge of the key, and even without any understanding of the operations of AES.

The following is an adaptation of the explanation in Vaudenay's "Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS ..." paper, which has been shared with you. If you choose, you may skip the explanation here and read the first four pages of this paper to answer the questions directly. Note that AES operates on blocks of 16 bytes instead of 8 in the paper.

AES encrypts plaintexts 16 bytes at a time (i.e. the block size is 16 bytes). If there are fewer than 16 bytes of plaintext data, AES adds **padding** bytes to the end of the plaintext until there are 16 bytes exactly. (During decryption, those padding bytes will be discarded.) If there are more than 16 bytes of data, AES operates on each block one by one in order, and pads the final block to 16 bytes. If we need to add n bytes of padding, then the bytes to add is exactly n copies of n. For example, suppose the plaintext we want to encrypt is:

$$x' = (\texttt{CA013AB4C561})_{16}$$

In the above, x' is written in hexadecimal notation, and it has 6 bytes. We want to add 10 bytes to make 16 bytes, so we will add the byte $(\texttt{0A})_{16} = 10$ ten times to make x, the padded version of x':

$$x = (\texttt{CA013AB4C5610A0A0A0A0A0A0A0A0A0A})_{16}$$

Note that the minimum amount of padding is 1 byte: that is to say, if the original plaintext has a multiple of 16 bytes, then we will need to add 16 bytes of padding of $(\texttt{10})_{16} = 16$. There will be a whole block of padding at the end.

After padding x' to x, we can perform AES encryption (denote the operation as C) on x to get the ciphertext $C(x)$. C is dependent on the secret key K and the initialization vector IV; the attacker knows IV because it is sent in the clear.

Suppose x contains N blocks of data (in other words, the size of x is $16N$ bytes), denoted as $(x_1|x_2|\ldots|x_N)$. $|$ is the concatenation operation, meaning that the bytes of x_1 are followed by that of x_2, and then by x_3, and so on. After encryption, the resulting ciphertext is $(IV|y_1|y_2|\ldots|y_N)$. In CBC mode, we have:

$$\begin{aligned} y_1 &= C(IV \oplus x_1) \\ y_i &= C(y_{i-1} \oplus x_i) \text{ for } i = 2, 3, \ldots, N \end{aligned}$$

The inverse of C, the AES block encryption function, is denoted as D, the block decryption function. Note that both C and D do not perform any padding on their own; they both input and output 16 bytes of data. For any 16-byte block z, $D(C(z)) = z$.

(a) [2 points] Consider the following plaintext x', which contains 5 repetitions of the byte AB_{16}:

$$x' = (\texttt{ABABABABAB})_{16}$$

x' is therefore 5 bytes long. Write down x, the padded version of x'.

(b) [2 points] Suppose you are given the ciphertext $(IV|y_1|y_2)$. Write down the plaintext $(x_1|x_2)$ using D, IV, y_1, and y_2. (It is not simply $D(y_1)$ and $D(y_2)$.)

(c) [22 points] We will now break AES in CBC mode using a *padding oracle*. A padding oracle is some entity that tells the attacker if the padding of some ciphertext $(IV|y = IV|y_1|\ldots|y_N)$ is correct after decryption. In other words, it decrypts $(IV|y)$ using the correct key and IV, gets the plaintext x, and checks if x uses the correct padding scheme described above. The padding oracle has been shared with you. (See "Notes on the Padding Oracle" later for more details on how to run the padding oracle.)

Suppose we are deciphering some ciphertext $(IV|y_1|\ldots|y_N)$. There will be three steps. First, we will learn how to find the last byte of x_N ("Decrypt byte"). Then, we will find the whole x_N ("Decrypt block"). Finally, we will find all of $(x_1|x_2|\ldots|x_N)$ ("Decrypt").

— *Decrypt byte* —

Extract y_N from the ciphertext by taking the last 16 bytes, and y_{N-1} as the last 32 to 16 bytes. Denote the ith byte of y_N as $y_{N,i}$. Here, we want to find $x_{N,16}$.

1. First, generate a random block $r = (r_1|r_2|\ldots|r_{15}|i)$ with 15 random bytes, followed by a byte i. Initially $i = 0$.
2. Ask the padding oracle if $(r|y_N)$ is valid. $(r|y_N)$ contains the 16 bytes of r, followed by the 16 bytes of y.
3. If the padding oracle returns "no", increment i by 1, and then ask the padding oracle again. Keep incrementing i until the padding oracle returns "yes".
4. Replace r_1 with any other byte and ask the oracle if the new $(r|y_N)$ has valid padding. If the padding oracle returns "yes", similarly replace r_2. Repeat until either we have finished replacing r_{15} and the oracle always returned "yes", or the oracle has returns "no" while we were replacing some r_k.
5. If the oracle always returned "yes" in Step 4, set $D(y_N)_{16} = i \oplus 1$.
6. If the oracle returned "no" when we replaced r_k in Step 4, set $D(y_N)_{16} = i \oplus (17 - k)$.
7. The final byte of x_N is $x_{N,16} = D(y_N)_{16} \oplus y_{N-1,16}$.

— *Decrypt block* —

After finding $x_{N,16}$, the attacker can proceed to find all other bytes of x_N, starting from the 15th byte $x_{N,15}$, then $x_{N,14}$, and proceeding backwards to $x_{N,1}$. In this process, the attacker will also find $D(y_N)_{16}, D(y_N)_{15}, \ldots, D(y_N)_1$ as above. The following describes how the attacker can find $x_{N,k}$ for any k; the attacker has already found $D(y_N)_{k+1}, D(y_N)_{k+2}, \ldots, D(y_N)_{16}$.

1. Set r as $(r_1|r_2|\ldots|r_{k-1}|i|D(y)_{k+1} \oplus (17-k)|D(y)_{k+2} \oplus (17-k)|\ldots|D(y)_{16} \oplus (17-k))$. Initially $i = 0$.
2. Ask the oracle if $r|y_N$ is valid.
3. If the padding oracle returns "no", increment i and ask the padding oracle again. Keep incrementing i until the padding oracle returns "yes".
4. When the padding oracle returns "yes", set $D(y_N)_k = i \oplus (17-k)$
5. The k-th byte of x_N is $x_{N,k} = D(y_N)_k \oplus y_{N-1,k}$.

— *Decrypt* —

The above shows how the attacker can decrypt the last block y_N to obtain X_N. To decrypt the k-th block y_k, the attacker simply replaces all of the above y_N with y_k and y_{N-1} with y_{k-1}.

Your task is to write a program, `decrypt`, which finds the plaintext x for any ciphertext y and outputs it to standard output. It is run with:

```
./decrypt ciphertext
```

`ciphertext` is a file that contains an amount of data that is a multiple of 16 bytes, and at least 32 bytes. It is formatted as $IV|y_1|\ldots|y_N$, where the IV is the first 16 bytes, y_1 are bytes 17 to 32, and so on.

After you get the plaintext, output it to standard output. Do not add a newline.

This is a difficult task. You should tackle the assignment step by step: do the "Decrypt byte" step, then the "Decrypt block" step, then the "Decrypt" step. In case you cannot finish the assignment, I will give marks for partially completing each step: 8 points if the code decrypts the last byte correctly, 16 points if the code decrypts the last block correctly, and 22 points if the code decrypts the entire ciphertext correctly.

(d) [4 points (bonus)] Write a program, `encrypt`, which takes in some plaintext x and encrypts x using the same encryption algorithm and key that is behind the padding oracle provided. It is run with:

```
./encrypt plaintext
```

`plaintext` contains an amount of data that is a multiple of 16 bytes, and at least 16 bytes. It is formatted as $x_1|x_2|\ldots|x_N$. Output the ciphertext and the IV to standard output as $IV|y_1|\ldots|y_N$.

(Hint: `encrypt` should call `decrypt` as a subroutine in order to guess the right ciphertext. You only need to call `decrypt` once for each block. Note that you can choose your own IV.)

Notes on the padding oracle

The padding oracle should be run with:

```
./oracle ciphertext
```

It will decrypt the ciphertext with the secret AES key, check the padding of the plaintext, and output "1" if the padding is correct and "0" if the padding is incorrect. If the ciphertext cannot be read correctly, it will return with a code of 1; otherwise it will return with a code of 0 after giving standard output.

We have provided two oracles for you, one written in Python 2, and one written in C. The C oracle is compiled and can be run directly:

```
./oracle ciphertext
```

The Python oracle can also be run using the above command on Unix systems like Ubuntu and OSX. For Windows, you will have to install Python and then type:

```
python oracle ciphertext
```

The next page has instructions on how to correctly capture the output of the oracle in your code.

The key is hardcoded into the oracle code, and you can find it there. **Do not use the key in any way.** When we test your code, we will use a different oracle with a different key. Your code should work independent of what the actual key value is.

You are also provided with a ciphertext called `ciphertext` for reference, with its generator `ciphertext_gen`. It was encrypted with the same key as the oracle, with an IV = `COMP3632 test iv`, and the plaintext message is `Message block1<two spaces>Message block2`. Since the plaintext is 30 bytes long, it will be padded with 2 bytes, each with a byte value of 2. If you find that the byte value of the last byte of the plaintext is 2, you are on the right track!

Notes on return codes and output

Note that there are largely two types of output any code can give:

1. Return code. This generally indicates if the code ran successfully or not. For `oracle`, it returns 0 if successful and 1 if not successful (because the input ciphertext could not be read correctly). The return code is usually not shown in the terminal when the code is run.

2. Standard output. This may include any relevant output of the program, and it is usually shown in the terminal when the code is run. For `oracle`, it outputs 0 if the ciphertext is wrongly padded and 1 if the ciphertext is correctly padded.

For example, in C++, `printf` and `cout` write to standard output, whereas `return` gives a return code.

The way to capture standard output and return code is different for various languages. We will use `./oracle ciphertext` as an example:

(Python) a = `subprocess.check_output(["./oracle", "ciphertext"])` will store the standard output of the oracle into variable `a`, but only if the input command has a return code of 0. `a` should be interpreted as an integer. If `oracle` does not have a return code of 0, the program will crash.

(C/C++)
```
FILE * fp = popen("./oracle ciphertext", "r");
char output[100];
fgets(output, 99, fp);
```
will store the standard output of the oracle into `output[0]`. Since it is a character, `(int)output[0]` is actually 48 (ASCII of 0) if it failed and 49 (ASCII of 1) if it succeeded. (Of course, if oracle is not trustworthy input then this code has a buffer overflow.)

The Java code is quite involved and you should look up how to run `Runtime.getRuntime()` and capture standard input if you want to write in Java.

Submission instructions

All submissions should be done through the CASS system. For this assignment, there is **no** Milestone deadline. Submit the following programs:

- `a2.pdf`, containing all your written answers, including the answers for parts (a), (b) of the programming assignment.
- The programming assignment, detailed below:
- `ptext0` and `ptext1`, for question 3 in the written assignment.

For the programming assignment, submit your code; do not submit any compiled files.

C++: Submit decrypt.cpp and encrypt.cpp. I will compile them into decrypt and encrypt and call `./decrypt <inputfile>`.

Py: Submit decrypt.py and encrypt.py. I will call `python decrypt.py <inputfile>`.

Java: Submit decrypt.java and encrypt.java. I will call `javac decrypt.java` and then `java decrypt <inputfile>`.

If there is a Makefile in your folder, the Makefile will override all of the above. I will use `make`, which should compile `decrypt` and `encrypt`, and I will call `./decrypt <inputfile>`. This implies if you are not writing in C++, Python, or Java, you must include a Makefile.

Note that your code for the programming assignment may rely on each other. For example, `encrypt` can call `decrypt`.

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.

The submission system will be closed exactly 48 hours after the due date of the assignment. Submissions after then will not be accepted unless you have requested an extension before the due date of the assignment. You will receive no marks if there is no submission within 48 hours after the due date.

Makefile

A Makefile is a set of instructions about how to compile code. When you type `make` in the Terminal of a Unix-based OS, it will search for the Makefile automatically, and run the instructions inside to create compiled code. It is also capable of detecting changes and missing prerequisite files or programs, to ensure that code can be compiled correctly.

For this assignment, if you are not using C++ or Python, you are asked to write your own Makefile so that we may compile your code. A link to a helpful guide about Makefiles has been added to the course materials. At the bottom, there is a sample Makefile for Java.