Total points: 80 (+6 bonus)

Assignment 2

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

Written assignment

1. [10 points] In February 2015, it was found that Lenovo computers came pre-installed with Superfish. Superfish adds itself as a root certificate authority to the computer. Every time a user visits an HTTPS site with a valid certificate, Superfish instead generates a fake certificate for the website to the user, and the browser would automatically trust such certificates. (The website's real certificate is intercepted and never presented to the user.) Using the fake certificate, Superfish identified users' browsing patterns and added or changed advertisements on web pages, creating a scandal for Lenovo. In the same month, Windows Defender started removing Superfish, and Lenovo ended their partnership with Superfish.
   (a) [4 points] Explain why Superfish is able to change the contents of an encrypted web page. Be specific about what Superfish steals, intercepts, and/or changes. (Hint: If the user is talking to the website using the website's encryption key, then Superfish certainly cannot change the contents because Superfish doesn't have the website's decryption key. So...)
   (b) [3 points] Superfish used the same signing key in every laptop computer. If an attacker (not Superfish) wants to impersonate a website, explain how the attacker can generate a valid certificate for that website to present to any computer with Superfish installed.
   (c) [3 points] How can a careful user notice that their computer is infected by Superfish using the browser?

2. [15 points] Download the file vignere.txt from the course website, which contains a ciphertext created using the Vignere cipher. Break it.

   The ciphertext is created as follows:

   1. Only letters in the plaintext are used - whitespace, numbers and punctuation are removed. All letters are converted to upper case.
   2. Each letter is assigned a number, starting with A = 0, B = 1, ..., Z = 25.
   3. The key is a string of upper case letters, with unknown length K (at least 5).
   4. To encrypt a letter with a key letter, add them modular 26.
      - e.g. C (2) + F (5) = H (7)
      - e.g. T (19) + N (13) = G (6)
   5. The n-th letter is encrypted with the ($n \mod K$)-th letter of the ciphertext, where K is the length of the key string, and indices start at 0.

   The encryption code is available as vignere_encrypt.py.

   You need to put in your report:

   - The key and the first word of the plaintext.
   - All steps you took to arrive at the result, including a description of any methods you used and explanation of why they work.
   - Screenshots showing snippets of relevant code. There is no need to submit the code.

   Your submission for this entire problem (Vignere cipher) should be in your pdf file.

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.

Two-Time Pad [25 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 600 bytes each, and they both come from popular English Wikipedia articles. 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.

Padding Oracle Attack [30 points]

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

The following is an adaptation of Vaudenay's "Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS ..." paper. However, **the padding scheme is intentionally different from the one used in that paper, to discourage plagiarism**. A solution derived directly from that paper will not work for this assignment.

AES encrypts plaintexts in blocks of 16 bytes at a time. 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 there are n real bytes, then we pad to 16 bytes with

null characters, except the last byte is set to the total amount of padding in the block. 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 to make x, the padded version of x':

$$x = (\texttt{CA013AB4C5610000000000000000000A})_{16}$$

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 add an additional block of null bytes (but the last byte is set to $(\texttt{10})_{16}$.

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, $D(C(z)) = z$.

We will now break AES in CBC mode with such padding using a *padding oracle*. A padding oracle is some entity that tells the attacker if the padding of some ciphertext $(IV|y_1|\ldots|y_N)$ is correct after decryption. In other words, it decrypts $(IV|y)$ using the correct key, 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 returned "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 $(r_1|r_2|\ldots|r_{k-1}|i|D(y)_{k+1}|D(y)_{k+2}|\ldots|D(y)_{16} \oplus (17-k))$. 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 $D(y_N)_k = i$
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}.

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 OR python3 decrypt.py ciphertext OR java 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. The plaintext is in ASCII.

Your program will be terminated in 10 minutes. The maximum size of the ciphertext is 64 bytes, which takes about 4 minutes on the marking machine.

After you get the plaintext, write it to a file called "plaintext.txt". Do not write any additional text into the file.

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, marks will be given for partially completing each step.

Hint: 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 $D(y_2)$.)

Bonus (6 points)

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 OR python3 encrypt.py plaintext OR java 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. You can assume your decrypt and encrypt code are in the same folder.)

Notes on the padding oracle

The padding oracle should be run with:

```
python3 oracle.py ciphertext
```

You need to install pycrypto to run the oracle. pycrypto is slightly out of date, and you may need to use an older version of Python - it is confirmed to work up to Python 3.9.

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.

You will also have to capture the output and feed it into your own code. The command to do so is `system(<your command>)` in C and C++, `subprocess.check_output(<your command>)` in Python, and `Runtime.getRuntime().exec(<your command>)` in Java. You may have to look up your preferred function to learn how to use it.

Since the oracle is not compiled, the key is hardcoded into the oracle code. **Do not use the key in any way.** When we test your code, we will use an 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.py`. It was encrypted with the same key as the oracle, and you can see the IV and plaintext used to create it; see if you can decrypt it correctly.

Submission instructions

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

- `a2.pdf`, containing all your written answers. Make sure it is not the question file.
- `ptext0` and `ptext1`, for part (a) of the programming assignment.
- `decrypt.{cpp, py, java}`, for part (b) of the programming assignment, as well as any other code necessary to run it. This may include a Makefile. Submit your code; do not submit any compiled files. You may also submit `encrypt.{cpp, py, java}` for the bonus marks. The bonus marks can only be applied to this assignment.

To run decrypt, for example, I will do the following:

C++: I will compile `./g++ decrypt.cpp -o decrypt` and then run `./decrypt`.

Python3: I will call `python3 decrypt.py`.

Java: I will compile `javac decrypt.java` and then call `java decrypt`.

If you are using Python, pleaase make sure it is Python3 instead of Python2.

If there is a Makefile in your folder, the Makefile will override all of the above. I will call `make` to compile the code, and then I will call `make run`.

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. You will receive no marks if there is no submission within 48 hours after the due date.