Total points: 80 (+6 bonus)

Assignment 2

Due: 11:59 PM, 22nd July (Fri)

Written assignment

1. There is an error in each of the following uses of cryptography that degrades its security. Identify the mistake and explain how it should be done instead.

   (a) [4 points] Alice wants to send Bob an encrypted e-mail. She generates two ECDH key pairs: one for encryption and one for signing. She uses the public encryption key to encrypt the e-mail, and attaches a signature created by using the private signing key to sign a SHA-256 hash of the e-mail. The ~~e-mails are~~ e-mail is sent through SMTP. She publicizes both public keys.

   (b) [4 points] Bob manages a website. To store Alice's password securely, Bob first asks Alice to encrypt it using 128-bit AES with a secret key (the key is shared with Bob). Then, Bob stores a hash of the encrypted version of the password using SHA-512 and also stores the secret key.

   (c) [4 points] Alice and Bob use the Diffie-Hellman protocol to establish a shared 256-bit secret key. After doing so, Alice wants to send Bob her bank account number so Bob can transfer money to her. Alice encrypts her bank number using the shared 256-bit secret key under AES in counter mode.

   (d) [4 points] In a private end-to-end encrypted messaging app, Alice adds Bob as a friend, which involves downloading his public 512-bit RSA key from a trusted server manging the app. To ensure that the public key is correct, the server uses its private 256-bit ECC key to sign it, which is verified by the corresponding public key that comes with the app. Having Bob's public key allows Alice to start creating secure connections with him.

2. [14 points] Cryptography relies on a series of assumptions regarding computational difficulty and key use. In each of the following cases, a commonly held assumption is broken. Discuss the impact of breaking that assumption, focusing on current cryptographic tools, how people interact with them, and how we (including all relevant parties such as cryptosuite developers) should respond to these discoveries.

   (a) [3 points] A quick polynomial algorithm for integer factorization has been found.

   (b) [4 points] The private signing key used by Facebook to sign HTTPS certificates was stolen by an unknown group two months ago.

   (c) [3 points] An easy way to determine the input corresponding to a given SHA-2 hash has been found.

   (d) [4 points] Large, practical quantum computers have been constructed. (Hint: Start with Shor's algorithm.)

Programming assignment

Breaking Cryptography

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

Two-Time Pad [20 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~~ 600 bytes each, and they both come from popular English Wikipedia articles downloaded on 27th June (around 4 PM). 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 ~~400-byte~~ 600-byte 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). 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 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 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 the bytes to add is exactly $16 - n$ 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 $(\texttt{06})_{16}$ ten times to make x, the padded version of x':

$$x = (\texttt{CA013AB4C5610606060606060606060606})_{16}$$

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 $(\texttt{00})_{16}$ (byte 0). There will be a whole block of padding at the end. The maximum value for padding is one byte of ~~$(\texttt{0E})_{16} = 15$~~$(\texttt{0F})_{16} = 15$.

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 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 $D(y_N)_{16} = i \oplus 15$.
6. If the oracle returned "no" when we replaced r_k in Step 4, set $D(y_N)_{16} = i \oplus (k-1)$.
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 $(r_1|r_2|\ldots|r_{k-1}|i|D(y)_{k+1} \oplus (k-1)|D(y)_{k+2} \oplus (k-1)|\ldots|D(y)_{16} \oplus (k-1))$. 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 $D(y_N)_k = i \oplus (k-1)$
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.

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

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.)

Notes on the Padding Oracle

The padding oracle should be run with:

```
python3 oracle.py 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.

The padding oracle was written in Python. It is not compiled, and it can be directly run on Unix-like systems such as Ubuntu and macOS. If you want to run it on Windows, you will have to install Python and then type:

```
python3 oracle.py ciphertext
```

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 documentation for the relevant command.

Since the oracle is not compiled, the key is hardcoded into the oracle code. **Do not use this key in any way.** When we test your code, we will set the oracle with a different key. Your code should work independent of what the actual key 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` ciphertext.

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

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

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.