Cryptography Basics

Due at Midnight on November 5, 2024

Simple Questions

1. My RSA key is **N = 187**, **e = 107**. You observe a ciphertext **c = 2**. What is the plaintext? Show your work.
2. What is $2^{147} \bmod 21$? Show your work.
3. Find all primitive roots of 37.
4. Find the number of positive integers that is less than 187 and a relative prime to 187. Show your work.
5. In an RSA system, the public key of a given user is **e=65**, **n=2881**. What is the private key of this user?
6. You are given two prime numbers **p=11**, **q=17** to generate a pair of keys. Show your work.
   a) What is the RSA modulus **n** and the totient function of **n**, **Φ(n)**?
   b) What is the private key **d** if public key **e** is chosen to 3?
   c) Decrypt ciphertext **C=3** using the parameters above.
   d) Encrypt message **M=70** using the parameters above.
7. In the Diffie-Hellman (DH) scheme, assume that **p=19** and **q=2**. Also assume that Alice's secret is **a=3** and Bob's secret is **b=5**. Compute the shared secret between Alice and Bob established using DH. Show your work.
8. Assume that Alice (**A**) and Bob (**B**) share a secret K_{AB} and **Hash** is a secure hash function. Consider the following authentication protocol.

```
1. A→B   I am A,R1
2. B→A   R2, Hash{KAB, A, B, R1}
3. A→B   Hash{KAB, A, B, R2}
```

   a) Trudy found that he can impersonate Alice in this authentication protocol. Explain how and why the protocol is susceptible?
   b) Fix the protocol by modifying only the meaning of contents in the hash.

Modular multiplicative inverse

A modular multiplicative inverse of an integer **a** is an integer **x** such that the product **ax** is congruent to **1** with respect to the modulus **m**. In the standard notation of modular arithmetic this congruence is written as

$$ax \equiv 1 \pmod{m}$$

It is equivalent of saying the remainder after dividing **ax** by the integer **m** is **1**. Not every element of **a** complete residue system modulo **m** has a modular multiplicative inverse,

for instance, zero never does. Only those integers relative prime to **m** have a multiplicative inverse in the modulus **m**.

My uneducated code to compute the modular multiplicative inverse of **a_int** is

```
4. for( int i=1; i<modular; i++ ) {
5.     if( ( a_int*i ) % modular == 1 ) {
6.         x = i;
7.     }
8. }
```

9. What is the complexity of my code?

   A modular multiplicative inverse of a modulo m can be found by using **the extended Euclidean algorithm**.

10. Explain the Euclidean algorithm first and the extended Euclidean algorithm.

11. Write a code to find a modular multiplicative inverse, using the extended Euclidean algorithm. You can only use such programming languages such as Java, C and C++. You are not allowed to use libraries except for standard libraries.

    Your code accepts an integer and a modulus as an input and prints a modular multiplicative inverse of a modulo m as an output. If no such inverse exists print out "NONE". Justify your code by including screenshots of the output.

12. What is the complexity of your code?

Random number generation

For generation of secret key in a safe way we want a pseudorandom number with a long period and seed.

More information of the random number generation I suggest reading an article in the below link.
https://blog.cloudflare.com/ensuring-randomness-with-linuxs-random-number-generator/

13. Answer to the following questions.

    a) What is the safest library to generate random numbers in your OS?

    b) What is the theoretical period of the generator?

    c) What is the safest way to select a seed?

    d) Using your choice of the generator verify that random numbers are the same if you initialize the generator with the same seed.

    e) Generate 10,000 random numbers between 0 and 1 and draw a histogram of those numbers.

    f) Discuss if those 10,000 numbers are fairly random.

    You must write a program to questions d) and e).

PKCS#7 Padding

You will learn AES encryption and decryption with two modes of operation, CBC and CTR. We are using AES with a 16-byte block size and a 16-byte key.

Those two modes of operation use a random and unpredictable initialization vector (IV). This random IV ensures distinct ciphertexts are produced even when the same plaintext is encrypted multiple times independently with the same key. The IV never needs to be kept secret. In CTR, the IV is used as an initial counter. In general, the size of IV should be identical to the block size not the key size.

AES requires a size of plaintext to be a multiple of block size. If a plaintext is not a multiple of 16 bytes, then AES adds extra bytes called **padding** to the end of plaintext. AES uses byte padding specified in PKCS#7[1].

In this scheme, the value of each added byte is the number of bytes that are added, i.e. N bytes, each of value N are added. The number of bytes added will depend on the block boundary to which the plaintext needs to be extended. The padding will be one of

```
01
02 02
03 03 03
04 04 04 04
05 05 05 05 05
06 06 06 06 06 06
```

and is well defined if and only if N is less than 256.

In the following example the block size is 8 bytes, and the 4-byte padding is required.

```
| DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |
```

14. According to the PKCS#7 specification, if the original data is a multiple of N bytes, then an extra block of bytes with value N is added. This implies that padding will always be added to a plaintext. Why are rationales behind adding padding to the plaintext which size is multiple of N bytes?

15. What might be a good way to remember an IV for ciphertexts?

Encryption and Decryption with AES

Use following a secret key and an IV. Use the PKCS#7 scheme for padding.

```
Key:  0x140b41b22a29beb4061bda66b6747e14
IV:   0xffe0c5da05d9476be028ad7c1d81468a
Plaintext: “I am taking the introduction to security in this Fall. Hyoung-
   Kee Choi is a name of a professor.”
```

You may use any cryptography libraries at your preference. You may use C, C++ and Java languages. Justify your code by capturing a screenshot of outputs.

16. Write a program to encrypt the plaintext and decrypt the ciphertext using AES in the CBC mode.

[1] An excerpt from https://en.wikipedia.org/wiki/Padding_(cryptography)#Byte_padding

a) What is the ciphertext in a hexadecimal form?

b) What is the size of plaintext?

c) What is the size of ciphertext?

d) What is the IV (Initialization Vector)?

e) Is the size of ciphertext the same as the size of plaintext? If so or not, why are they different?

Rainbow Table

This homework was originally developed by Jonathan Katz at the University of Maryland.

Assume the following scheme is being used to hash passwords: An **n**-bit password **P** is padded to the left with **128 - n** zeros and used as an AES-128 key to encrypt the 32-byte all **0** plaintext; that is "00000000000000000000000000000000". The result is the **hashed password**.

$$H(P) \overset{def}{\Longrightarrow} \mathrm{AES}_{[0^{128-|P|}\|P]}(0^{128}).$$

So for the 12-bit password $P = 0\mathrm{xABC}$, the result should be

$$H(P) = 970\mathrm{fc16e71b75463abafb3f8be939d1C}.$$

Test online at http://aes.online-domain-tools.com/. You may assume **n** is a multiple of 4. Your attack should use a rainbow table with $2^{n/2}$ chains of $2^{n/2}$ length each.

The scenario is that you are given $H(P)$ and n and need to recover P. This can always be done by a brute-force attack using about 2^n AES evaluations. Alternately, one can pre-compute all 2^n possible hashes and then find P in essentially constant time; this requires O(2^n) space. The goal of using a rainbow table is to do better and faster.

Write two programs, called **GenTable** and **doCrack**. Use any programming language at your preference. The first of these corresponds to the pre-processing phase in which you generate a rainbow table, while the second corresponds to the on-line phase in which you are given $H(P)$ and need to recover P.

GenTable should take a single command-line argument and generates output to a file **rainbow**. The command-line argument will be **n**, the password length in bits. The bound on the size of **rainbow** must be no larger than $3 \times 128 \times 2^{n/2}$ bits (why?). Failure to meet this space bound will result in 0 points. You can use **ls -l** to check the size of your **rainbow** file. You must design your reduction function.

doCrack should take two command-line arguments and generate output to STDOUT. The first command-line argument is the same as above. The second argument is $H(P)$ in hex. The output of **doCrack** should include two items: the password **P** or "failure", and the number of times AES was evaluated.

Justify your code by including screenshots of the output and size of the rainbow file.

17. What are the 20-bit password represented in hex for

```
19. 8de0bcffe587f63ed5c823dcf9bf5131
```

```
20. f7ef413cc51df04abf6872db315e694b
```

18. What are the 24-bit password represented in hex for

```
21. ed078d9b527a81fe4725228d88b664ae
22. ae955b027a3d0cb5401b63b4d26a10ba
```

[Extra point]

19. What are the 28-bit password represented in hex for

```
23. 86527077e1cb39b6b2e6f414b1a758f6
```

Deliverables

- Using MS-Word or other your favorite word-processors answer to the question.
- Name your file to your_school_id.[doc|hwp].
- Do not submit codes. Instead copy them in the report.
- Do not compress the file.
- Failure to fulfill the instruction will result your grade zero.