CPSC 4200/6200
November 12, 2020

Homework Assignment #5

Total: 100 points

1. Complexities of Brute-Force Attacks (20 points)

1. The passcode to access a tablet PC has 5 characters. Each character can be any of lower or upper case letters, a digit from 0 through 9, or one of 10 symbols. In a brute-force attack to guess the passcode, how many passcodes does an attacker have to try in the worst case?

2. Alice sends Bob an encrypted document. Characters in the original document include upper or lower case letters, numbers (0-9), 10 unique symbols, and the space. Her encryption scheme is substitution, i.e., replacing each occurrence of a character in the original document with any one of the upper or lower case letters in the alphabet, 10 digits, 10 symbols, and the space. Eve intercepts the ciphertext. In a brute-force attack, how many trials does Even have to perform to decrypt the ciphertext in the worst case?

3. Eve attempts to write a script to guess the password to Alice's email account in a brute-force fashion. Suppose Alice's password has 8 characters, consisting of lower/upper case letters, digits 0-9, and 10 symbols. The total network time for each password submission takes 2 seconds. Under each of the following conditions, how long does it take for Eve to successfully break into Alice's account in the worst case?

   (a) The server allows at most 5 login attempts within one minute.

   (b) The server allows at most 15 login attempts within 30 minutes.

2. Bail-Out Money (20 points)

As soon as Barack took office, he decided to embrace modern technology by communicating with cabinet members over the Internet using a device that supports cryptographic protocols. In a first attempt, Barack exchanges with Tim brief text messages, encrypted with public-key cryptography, to decide the exact amounts of bailout money to give to the largest 10 banks in the country. Let p_B and p_T be the public keys of Barack and Tim, respectively. A message m sent by Barack to Tim is transmitted as $E_{p_T}(m)$ and the reply r from Tim to Barack is transmitted as $E_{p_B}(r)$. The attacker can eavesdrop the communication and knows the following information:

- Public keys p_B and p_T and the encryption algorithm.
- This encryption algorithm has the following property. Given a public key, the same plaintext is always encrypted to the same ciphertext, i.e., the encryption algorithm is deterministic.

- The total amount of bailout money authorized by congress is \$900B.
- The names of the largest 10 banks.
- The amount each bank will get is a multiple of \$1B.
- Messages and replies are terse exchanges of the following form:

  Barack: How much to Citibank?

  Tim: \$144B.

  Barack: How much to Bank of America?

  Tim: \$201B.

  ...

1. Describe how the attacker can learn the bailout amount for each bank even if he cannot derive the private keys.
2. As a result of the above attack, Barack decides to modify the protocol for exchanging messages. Describe two simple modifications of the protocol that will make the above attack much more difficult. The first one should use random numbers and the second one should use symmetric encryption.

3. RSA Cryptosystem (30 points)

Bob has modulus n and exponent e as his RSA public key, (e, n). He has told Eve that she can send him any message $M < n$ and he is willing to sign it using a simple RSA signature method to compute $S = M^d \bmod n$, where d is his private RSA exponent, and he will return the signature S to Eve. Unfortunately for Bob, Eve has captured a ciphertext C that Alice encrypted for Bob from her plaintext P using his RSA public key. (Bob never actually got C.) Eve wants to trick Bob into decrypting C for her and she does not want Bob to see the original plaintext P that goes with C. So Eve asks Bob to sign the message $M = r^e C \bmod n$ using his private RSA exponent, and send her back the signatures S for M, where r is random number that Eve chose to be relatively prime to n.

1. Explain how Eve can use Bob's signature, S, on M, to discover the plaintext, P, for C. Please show your work step-by-step.
2. How can Bob prevent this attack?

4. RSA Public-Key Encryption Lab (30 points)

Background : The RSA algorithm involves computations on large numbers. These computations cannot be directly conducted using simple arithmetic operators in programs, because those operators can only operate on primitive data types, such as 32-bit integer and 64-bit long integer types. The numbers involved in the RSA algorithms are typically more than 512 bits long. For example, to multiple two 32-bit integer numbers a and b, we just need to use a*b in our program. However, if they are big numbers, we cannot do that any more; instead, we need to use an algorithm (i.e., a function) to compute their products.

There are several libraries that can perform arithmetic operations on integers of arbitrary size. In this lab, we will use the Big Number library provided by openssl. To use this library, we will define each big number as a BIGNUM type, and then use the APIs provided by the library for various operations, such as addition, multiplication, exponentiation, modular operations, etc.

BIGNUM APIs: please refer to "SEED Labs – RSA Public-Key Encryption and Signature Lab" for more details. We also provide a sample code in Canvas. **Compilation**: We can use the following command to compile the code (the character after - is the letter l, not the number 1; it tells the compiler to use the crypto library).

$ gcc -o yourcode yourcode.c -lcrypto

Submission requirement: you should describe your steps and running results (including screenshots of your code and outputs) in the report.

Task 1: Deriving the Private Key

Let p, q, and e be three prime numbers. Let $n = p*q$. We will use (e, n) as the public key. Please calculate the private key d. The hexadecimal values of p, q, and e are listed in the following. It should be noted that although p and q used in this task are quite large numbers, they are not large enough to be secure. We intentionally make them small for the sake of simplicity. In practice, these numbers should be at least 512 bits long (the one used here are only 128 bits).

p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3

Task 2: Encrypting a Message

Let (e, n) be the public key. Please encrypt the message "A top secret!" (the quotations are not included). We need to convert this ASCII string to a hex string, and then convert

the hex string to a BIGNUM using the hex-to-bn API `BN_hex2bn()`. The following python command can be used to convert a plain ASCII string to a hex string.

$ python -c 'print("A top secret!".encode("hex"))'
4120746f702073656372657421

The public keys are listed in the followings (hexadecimal). We also provide the private key d to help you verify your encryption result.

- n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
- e = 010001 (this hex value equals to decimal 65537)
- M = A top secret!
- d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

Task 3: Decrypting a Message

The public/private keys used in this task are the same as the ones used in Task 2. Please decrypt the following ciphertext C, and convert it back to a plain ASCII string.

C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

You can use the following python command to convert a hex string back to a plain ASCII string.

$ python -c 'print("4120746f702073656372657421".decode("hex"))'
A top secret!