**1 – Complexities of Brute-Force Attacks**

* 1. – 26 (uppercase) + 26 (lowercase) + 10 (numbers) + 10 (symbols) = 72 possibilities for each of 5 characters. 72 x 72 x 72 x 72 x 72 = 1,934,917,632 possible 5-character combinations. So in the worst case it could take 1,934,917,632 attempts for a brute-force attack to work.

**1.2** – Since any letter in the original document could be replaced with any one of the upper or lower case letters in the alphabet, 10 digits, 10 symbols, and the space, the cipher key will contain 73 characters. This means that there will be 73! (73 factorial), or 4.470115e+105 possible keys. Thus, in the worst case, it would take 4.470115e105 trials with a brute-force attack to guess the correct cipher key.

**1.3a** – 72^8 = 7.22204136e14 character combinations for the password. With the server only allowing 5 attempts per minute, it would take (7.22204136e14 / 5) minutes in the worst case for a brute-force attack to work. So it would take approximately 1.444408272e14 minutes to crack this password.

**1.3b** – 72^8 = 7.22204136e14 character combinations for the password. With the server only allowing 15 attempts every 30 minutes, it would take (7.22204136e14 / 15) 30-minute periods in the worst case for a brute-force attack to work. So it would take approximately (4.814694242e13 \* 30) = 1.444408272e15 minutes to crack this password.

**2 – Bail Out Money**

**2.1** – Since the encryption algorithm is deterministic, the same pieces of plaintext will always be encrypted to be the same pieces of ciphertext. For instance, the “How much to \_\_\_?” is always the same in Barack’s messages. So the attacker can infer that the \_\_\_ in all of Barack’s encrypted messages is the name of the bank in that message. The attacker could then use the public keys to encode the names of the ten largest banks to figure out what those names look like when encoded. The attacker can then figure out which messages are referencing which banks from there. Then he could do the same for all 900 possibilities for the amounts given to the banks (could be anything from $1B to $900B) to see what those values look like when encrypted. From there, he could figure out what amounts are being given to which banks.

**2.2** – One modification Barack could implement is to switch his block cipher to the CTR (or counter) mode of operation. This would result in the introduction of random numbers/randomness into the encryption process, which would mean that the same, repeated pieces of plaintext in the messages would *not* be encrypted to the same pieces of ciphertext every time. In other words, it would make the encryption algorithm non-deterministic. Another modification Barack could implement is to change the encryption scheme to use symmetric cryptography (or symmetric keys). Barack could create a key (A) and encrypt his messages with that key. Then he could encrypt key A, itself, with Tim's public key (which we will call key B). Barack would then be able to send his messages encrypted with key A. But in order for Tim to be able to decrypt them, Barack would have to send key B to Tim (at least the first time). Tim, and only Tim, would be able to decipher key B since he is the only one who has access to his private key. Then he could use the deciphered new key (which would be key A at that point) to decipher Barack’s messages from then on.

**3 – RSA Cryptosystem**

**3.1** – Bob’s response to Eve’s message will be *B mod n,* where B = (reC mod n). B is a verified message, signed by Bob’s signature S. Now Eve has what she needs to decipher the ciphertext.

*B = (reC mod n)d*

*= (Cd mod n)(red mod n)*

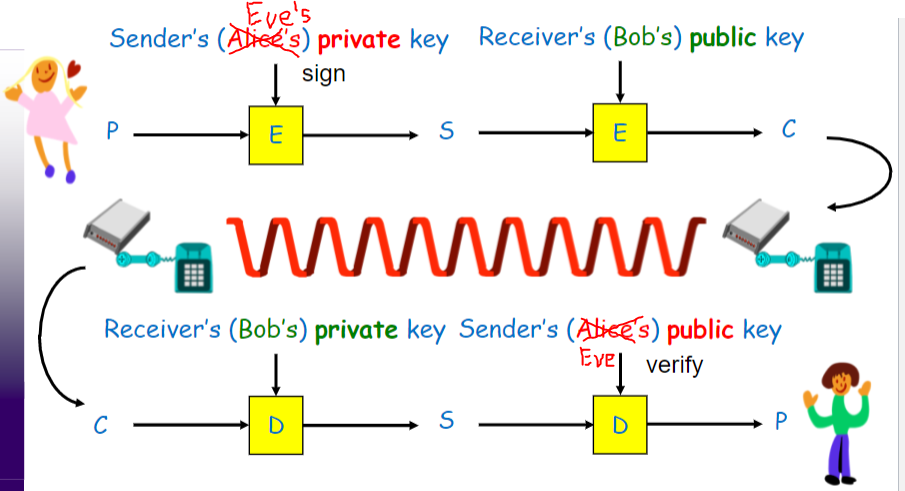
*= (rCd) mod n*

*And because r is a random value:*

*= (P)(r ) mod n || Cd mod n = P* per RSA rules.

Now, to find the plaintext P, Eve can calculate **P = (K)(r-1) mod n** (or, P = (K/r) mod n). This operation will result in Eve decrypting the ciphertext into the original plaintext that Alice tried to send to Bob.

**3.2** – There are many ways Bob could prevent this attack from occurring as it is a very simple attack from the attacker’s perspective. One way is that he can require that Eve practice “Confidentiality with Authentication” as shown below from slide deck 19:

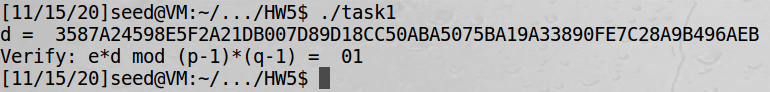


**4 – RSA Public Key Encryption Lab**

**4.1 – Deriving the Private Key**

For this task, I just used the DerivePrivateKey.c file provided by Dr. Cheng and plugged the following values into the program:

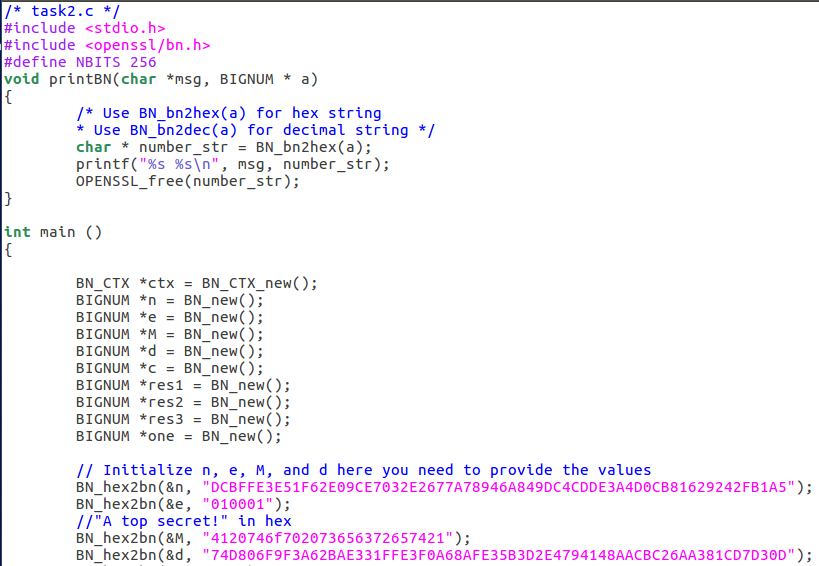


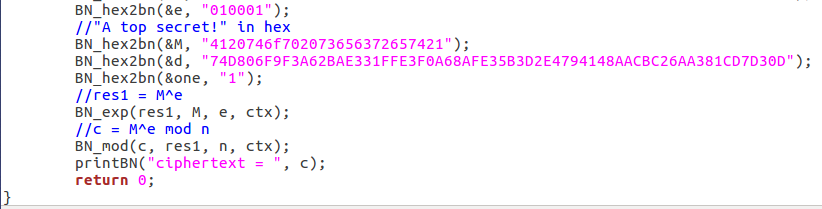


**4.2 – Encrypting a Message**

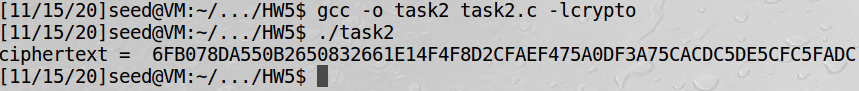
In this program, I have plugged in values for n, e, M, and d. The lines that perform the encryption start 5th from the bottom: BN\_exp(res1, M, e, ctx); This line performs the mathematical function of res1 = me. Then the line 3rd from the bottom: BN\_mod(c, res1, n, ctx); This line performs the mathematical function of c = Me mod n. Then we print out c which is the encrypted ciphertext.

Below are screenshots of my encryption program:





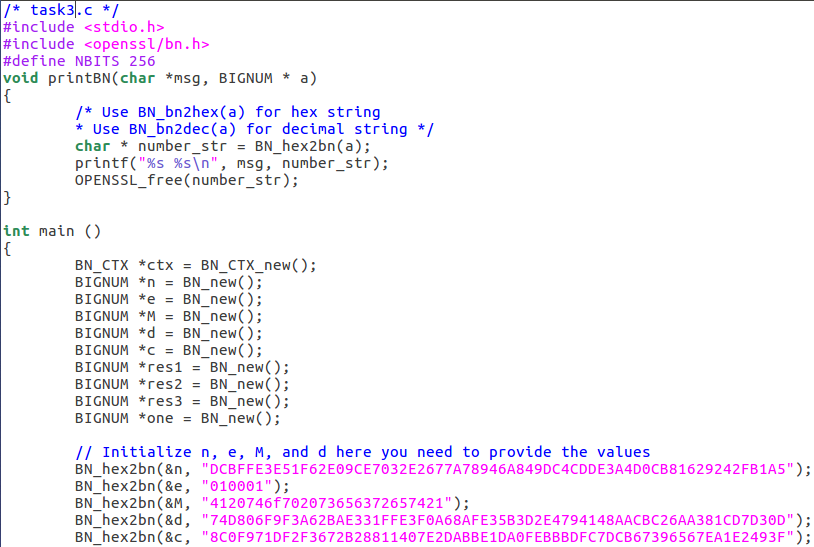
And my results:



**4.3 – Decrypting a Message**

In this program, I have plugged in values for n, e, M, d, and c. The line that performs the decryption is 3rd from the bottom: BN\_mod\_exp(M, c, d, n, ctx); This line performs the mathematical function of m = cd mod n. I can then convert the result of the program back to an ASCII string with the python command shown below.

Below are screenshots of my decryption program:





And my results:

