CSCD27 Assignment 1

1. **Course Feedback**
2. **PGP/GPG: secure email in practice**
3. **GPG: Revoking a Key**
   For Q1-3, please refer to files attached to the email. :)

4. **Insider Threats**
   a. **determine conclusively whether your GCC had been hacked with a backdoor:**

      My GCC compiles: $\mathbf{GCC_0}$,
      Use Clang compiler to compile GCC: $\mathbf{GCC_1}$,
      (GCC_0 and GCC_1 are compiled by different compilers, so they should have differnt binary representation, and we cannot tell if GCC_0 has been hacked or not)
      Use GCC_0 to compile a new GCC again : $\mathbf{GCC_0'}$,
      and similarly, use GCC_1 to compile we get $\mathbf{GCC_1'}$.
      Conduct a binary compare on GCC_0' and GCC_1' ==>
      If GCC_0 have not been hacked, we can assume GCC_0 and GCC_1 are functionally equivalent, therefore they should produce compiled GCC with same binary representation and functionality, i.e. GCC_0' and GCC_1' are the same in terms of binary representation;
      otherwise, GCC_0' and GCC_1' will be different bit-wise which means my GCC have been hacked with a backdoor.

   b. **how you could safely use that GCC, together with Clang, to rebuild your Linux systems to eliminate the strange behavior.**

      I would use Clang compiler to compile GCC and use the compiled GCC to compile GCC again(so that the executable is bit-wise correct), then use that version(i.e. GCC_1' as mentioned in part a above) to re-compile everything in linux from the untampered source code

5. **Perfect Ciphers**
   a. **If Alice performs encryption as "M ⊕ K" (using the binary representations of M and K), is this as secure as a perfect substitution cipher (like an OTP)?**

      We define cipher to be perfect if :

      $$P(encrypt(K, M0) = C) === P(encrypt(K, M1) = C)$$

      The M ⊕ K encryption is not a perfect cipher, because for some ciphertext, it is possible to learn about some properties of its plaintext. i.e.

M\K	00	01	10	
00	00	01	10	In the table on the left, we can see the ciphertext table by applying M ⊕ K, for M, K in {0, 1, 2}: If C = 11, then its plaintext M must be chosen from {01, 10} and cannot be in {00}, thus P(encrypt(K, 00) = 11) != P(encrypt(K, 01) = 11); the same approach can be applied to C = 01, C = 10. Therefore, given any ciphertext, all possible associated plaintext values are not equally likely, it is not a perfect cipher.
01	01	00	11	
10	10	11	00	

b. **If so, explain why it is as secure, and if not, show how you could change the algorithm, but not the message or key configurations, to transform it into a perfect cipher.**

I will change to encryption algorithm to C = (M + K) mod 3, the ciphertext table in binary representation are listed below:

M\K	00	01	10	
00	00	01	10	P(encrypt(K, 00) = 00) = P(encrypt(K, 01) = 00) = P(encrypt(K, 10) = 00) P(encrypt(K, 00) = 01) = P(encrypt(K, 01) = 01) = P(encrypt(K, 10) = 01) P(encrypt(K, 00) = 10) = P(encrypt(K, 01) = 10) = P(encrypt(K, 10) = 10)
01	01	10	00	
10	10	00	01	Given any ciphertext, all possible associated plaintext values are equally likely, it is a perfect cipher.

6. Feistel Ciphers

a. **Show that Feistel encryption is invertible even when Feistel function F() is a one-way (non-invertible) function like a hash function, by showing algebraically how encryption and decryption are related.**

Let **F()** be the round function and let $K_1, K_2, \dots, K_n$ be the sub-keys for the rounds 1,2,...,n respectively.

For each round i (i = 1, 2, …, n), we split the plaintext block into two equal pieces, (L_i, R_i), and we express L_i and R_i based on previous round as:

$$L_i = R_{i-1} \quad \text{and} \quad R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

Then the ciphertext is (R_n, L_n).

Decryption of a ciphertext (R_n, L_n) is accomplished by computing for i=n,n-1,...,1

$$R_i = L_{i+1} \quad \text{and} \quad L_i = R_{i+1} \oplus F(L_{i+1}, K_i).$$

Then (L_1, R_1) is the plaintext again.

We could see that decryption requires only reversing the order of the subkey $K_n, K_{n-1}, ..., K_1$ using the same process and same function F(), therefore F() can be one-way function and even more complex while the encryption is still invertible.

b. **When designing a specific instance of a Feistel cipher, such as DES, a number of parameters must be set (chosen). List 4 such parameters.**
   i. design of the F() function
   ii. algorithm for deriving round-keys K_i from the encryption key
   iii. number of rounds performed
   iv. block-size

c. **Give two reasons why 3DES was not considered a competitive option, and briefly justify your answers.**
   i. 3DES is slow in software implementations, compared with newer algorithms like AES
   ii. 3-DES use 64-bit blocks, but larger blocks preferable for efficiency and security concerns

7. **Brute Force Attacks and Bit Strength**
   a. **What is the probability that they will find a unique k' that satisfies this equality DES(k',p)=c, and thus will have broken Alice+Bob's key k?**

Let P* be the probability that there are two different keys k_1 and k_2 , such that DES(k_1, p) = c = DES(k_2, p), for a specific pair of (p, c).

It is equally likely to ask, given plaintext p and ciphertext c, when would a random key k' happens to produce a ciphertext c' that collides with c = DES(k, p).
For any random key k', the number of possible ciphertext is 2^{64} (as ciphertext is 64-bit), thus the probability for generated c' = c (for a specific c) is:
   P(c' = c) = 1 / 2^{64}, for any key k in the keyspace
This probability is true for any random key k' other than the real key k, so there are 2^{56} - 1 possible keys in total (as the key is 56-bit). We have:
   P* = \SUM { P(c' = c) for key k'} [for all possible k']
      = (2^{56} - 1) * P(c' = c)
      = (2^{56} - 1) / 2^{64}
The probability that there is a unique k' such that DES(k, p) = c is the complement of P*, therefore
   P = 1 - P* = 1 - (2^{56} - 1) / 2^{64} ≈ 0.99609

b. **What additional information would the adversary require in order to achieve closer to 100% confidence of k' uniqueness (say 99.99+% confidence?).**

Given more known plaintext pairs (p, c) and do the same analysis, and the probability will be very close to 1

c. **Answer parts (i) and (ii) above for the alternate situation where Alice and Bob are encrypting using AES encryption, rather than DES.**

   AES is not vulnerable to known cipher attacks, therefore the probability is 0 for the attacker to find the key k.

d. **For each of the following encryption/hash algorithms, indicate the key-length (or minimal algorithm variant) or hash-size (or minimal algorithm variant) required to achieve the NIST-recommended security strength, and in each case, briefly explain of your reasoning:**

   i. ***DES (* one or none of: DES, 2DES or 3DES)**
      DES and 2DES are not able to achieve the strength according to the DES challenge, only 3DES with 168 bits key size can make it.
   ii. **AES**
      the minimum key-length would be 112 bits, as it requires 2^212 computations to break the key by brute-force.
   iii. **RSA**
      2048-bit RSA keys are able to achieve the goal as we learn from lecture that 1024 bit RSA keys are equivalent to 80-bit symmetric keys
   iv. **SHA* (* one or none of: 1, 256, 512)**
      SHA-256 would be sufficient to achieve the security.

e. **What impact would Moore's law have on the NIST recommended bit strength value, by year 2035, assuming Moore's law continues to hold?**

   Moore’s rule is the observation that the number of transistors in a dense integrated circuit doubles approximately every two years (18 months), which indicates that one additional bit can be broken every 18 months.
   ceil((2035 - 2015) * 12 / 18) = 14
   Therefore, the NIST recommended bit strength value should be at least 126 (= 112 + 14) digits by 2035.

8. Linear Cryptanalysis

Define C_j = 000….0010..00 where the only appearance of 1 is at index *j* for the 128-bit-long ciphertext, with corresponding plaintext P_j

Let CJ be the union of all Cj’s for 1 <= j < = 128, that is, CJ = {C1, C2, ... , C128}

For any given ciphertext C, we can select a subset from CJ such that C can be expressed as the exclusive-or of those selected Cj's.
i.e. for a 4-bit case

$C = 1001$ can be expressed as $C = C_1 \oplus C_4 = 1000 \oplus 0001$

Since both ciphertext and plaintext are of the same size (128-bits), and that non-chained plaintext would have identical ciphertext for 2 identical plaintexts, given Lincipher(k, p1 ⊕ p2) = Lincipher(k, p1) ⊕ Lincipher(k, p2), we have:

$$\text{Decrypt}(C_1 \oplus C_2)=\text{Decrypt}(C_1) \oplus \text{Decrypt}(C_2)$$

Thus for any ciphertext C, we have $\text{Decrypt}(C) = \text{Decrypt}(C_1 \oplus C_2 \oplus \ldots) = \text{Decrypt}(C_1) \oplus \text{Decrypt}(C_2) \oplus \ldots = P_1 \oplus P_2 \oplus \ldots$, therefore, we can conclude that the encrypted plaintext P is $P = P_1 \oplus P_2 \oplus \ldots$ (which is, the XOR's of corresponding P_j for all C_j in the subset that represents C).

9. Block-Mode Integrity

a. increase his bonus payment from $250 to $25000

(based on original encrypted message:
586488ddf4db5e20d1ccc6efa3fea352 8365009e837b89c2fbee95876a0b1548**)**

$C_i = AES(K, (P_i \oplus C_{i-1}))$, where $C_{-1} = IV$
$P_i = D(K, C_i) \oplus C_{i-1}$
Changing last part of P(last 6 bytes, bits 80-127) from 250.00 (00000061A8) to 25000.00(00002625a0), and alter the IV accordingly, so that the rest of the blocks stay the same as before.
Zero bits have no effects in xor operation, so we can just take the last 3 bytes:
fea352 ⊕ 0061A8 = fec2fa
fec2fa ⊕ 2625a0 = d8e75a

The altered hex-encoded AES-CBC encrypted message would be :
586488ddf4db5e20d1ccc6efa3**d8e75a** 8365009e837b89c2fbee95876a0b1548

b. Is it important that the IV be random, or could a fixed value be used?
Yes, the IV needs either to be random, or being used as a fixed value. In encryption, if the data and the key are the same then the encrypted data will be the same, and thus exposes the vulnerability once a pattern is found. CBC resolves it by XORing the encrypted data from previous blocks to generate randomness, but the overall process starts from the IV, so the attacker could change bits of first block and change IV to compensate.

c. Is it safe for the bank to send the IV unencrypted?

Not safe unless IV is a fixed value. If the IV is known, then the attacker can easily modify it to make the resulting first block of the plaintext be anything he wants, without any other changes anywhere in the resulting plaintext, thus makes the encryption unsafe. We should consider the security of the IV be the same as the encrypted data blocks.

d. **encrypted using AES-CTR**
   Yes, this change close the security hole exploited by Mallory above because he can no longer modify the IV with ease, thus unable to tamper the message undetected.

   However, using the same secret key to encrypt IV as is used to generate the CTR keystream would bring new problems. This is because CTR encrypts the successive values of a counter, with IV being the initial value. Security is achieved as long as no counter value is ever used twice with the same key, otherwise the attacker could easily exploit the internal structure of the data due to the properties of XOR operation.

10. Hash Extensions and Collisions

a. **Hash Extension Concepts** [done in tutorial]
b. **Hash Extension Attack**
   see mathlab submission: extension.py

c. **MD5 Hash Collisions** [read]
d. **Generating an MD5 Hash Collision**
   i. the time to generate hash collision

```
zhangmao@mathlab:~/cscd27f15_space$ time ./hashclash -o f1 f2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'f1' and 'f2'
Using initial value: 0123456789abcdeffedcba9876543210

Generating first block: ..
Generating second block: S00..........
Running time: 1.37 s

real    0m1.394s
user    0m1.374s
sys     0m0.004s
```

   ii. the hex content of files f1 and f2

```
zhangmao@mathlab:~/cscd27f15_space$ xxd f1
0000000: 94b5 5f60 f27b 1db0 b3cf eb6d 0dc8 a390  .._`.{.....m....
0000010: ecf5 266b 4927 f590 6332 38e6 9886 a1d6  ..&kI'..c28.....
0000020: edf1 c5cd 6284 338a 4f6a bc19 f201 261d  ....b.3.Oj....&.
0000030: 7b98 b61e 132d 7c0e eb6b f9a3 ac1e f068  {....-|..k.....h
0000040: c595 25c7 2473 162d 2106 0d16 a926 613c  ..%.$s.-!....&a<
0000050: ae6d 0543 f83d 4161 cd3b f6c0 595a 7d32  .m.C.=Aa.;..YZ}2
0000060: 492f 5fbe 2f1c af77 7d6c bccf bc79 5b44  I/_./..w}l...y[D
0000070: c85b 34ee f307 0247 351a 3a60 3f39 4a3f  .[4....G5.:`?9J?
zhangmao@mathlab:~/cscd27f15_space$ xxd f2
0000000: 94b5 5f60 f27b 1db0 b3cf eb6d 0dc8 a390  .._`.{.....m....
0000010: ecf5 26eb 4927 f590 6332 38e6 9886 a1d6  ..&.I'..c28.....
0000020: edf1 c5cd 6284 338a 4f6a bc19 f281 261d  ....b.3.Oj....&.
0000030: 7b98 b61e 132d 7c0e eb6b f923 ac1e f068  {....-|..k.#...h
0000040: c595 25c7 2473 162d 2106 0d16 a926 613c  ..%.$s.-!....&a<
0000050: ae6d 05c3 f83d 4161 cd3b f6c0 595a 7d32  .m...=Aa.;..YZ}2
0000060: 492f 5fbe 2f1c af77 7d6c bccf bcf9 5a44  I/_./..w}l....ZD
0000070: c85b 34ee f307 0247 351a 3ae0 3f39 4a3f  .[4....G5.:.?9J?
```

iii. the sha-256 hashes of files f1 and f2

```
zhangmao@mathlab:~/cscd27f15_space$ sha256sum f1 f2
acd479d9c19f8eb649819bbe392334034484316a3fd6e85351e15471dd5c2d5f  f1
7c8bdd2e0d5d1f3f59aecd49accd4ec85d3e98d1a3973962fb2a3c117b5a249c  f2
```

iv. the md5 hashes of files f1 and f2

```
zhangmao@mathlab:~/cscd27f15_space$ md5sum f1 f2
3ca717b6336210fd332a2a66ba79448d  f1
3ca717b6336210fd332a2a66ba79448d  f2
```

e. **Don't be Evil**

see mathlab submission: nice.py and nasty.py
both programs will will print the sha-256 bits first, then:
nice.py prints “This is a nice program... :)”,
and nasty will print “rm -rf / in progress.... Kidding! :p”

11. **RC4 Implementation**

see mathlab submission: rc4.py and rc4_nose.py (with test file neil.wav)