CS370 Computer Security

Homework 2

Jingyuan Xu

3.1 [5 pts] Observation Task: One-way Hash and their randomness.

1. Create a text file with the following text -- 'The cow jumps over the moon'

2. Generate the hash value H_1 for this file using SHA512 (SHA256).

Answer:

By using SHA512, we can get hash value

H1=3f03cd845ee8f5405d64055c08be02ca1c386b153e8f10ef19fdca85ffe406eb1c038fa4b922
0ad414cc2cb88a1ef0e81b6c91339321d415808d2e0d0b7dd02d

By using SHA256,

H1=9ed7b218f3402376f0d2f77b89093704515f68f82857f17c9ba8dc4348ecf81f

3. Flip one bit of the input file. You can achieve this modification using hex editors like ghex or Bless.

Answer:

By using hex editor, we can change the original input file: The cow jumps over the moon (54686520 636F7720 6A756D70 73206F76 65722074 6865206D 6F6F6E0A) into a new one with changing the second bit to "5": Uhe cow jumps over the moon (55686520 636F7720 6A756D70 73206F76 65722074 6865206D 6F6F6E0A)

4. Generate the hash value H_2 for the modified file.

Answer:

By using SHA512:

H2=
a8ec5654b7a4b899465ff45a6b26eef75da9331756c39e91525ca22ffc782389b550250e12bda5
d46d482dd5aab44ba2cb7dafe33864617a7aea8d1954b9a9a1

By using SHA256:

H2= 9086d3ab996a83a0ee5de080c6d9efcebd7e398a508e8bd07a3c8eeb3b816994

5. Please observe whether H_1 and H_2 are similar or not. How many bits are the same between H_1 and H_2.

Answer:

Not samiliar. with SHA256: the same bits are 130, with SHA512, the same bits are 265

6. Flip bits 1, 49, 73, and 113 and record the number of bits that are different between H_1 and H_2 when using SHA512 and SHA256 in a table. What trend do you see in table? Does this trend change if you flip different bits in the file or flip multiple bits?

different	1	49	73	113
SHA256	124	128	119	138
SHA512	250	258	272	255

The number of different is nearly half of the total bits. It won't change if I flip different bits in the file or I flip multiple bits

3.2 [5 pts] Observation Task: Keyed Hash and HMAC

Please generate a keyed hash using HMAC-SHA256, and HMAC-SHA512 for the text file that you created in the task above. In each case try keys of length 128, 160 and 256 bits and record your output. Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

128 bits key:

cmd: openssl dgst -sha256 -hmac "0123456789abcdefg" text.file
openssl dgst –sha512 -hmac "0123456789abcdefg" text.file
result: HMAC-SHA256(text.file)=
440fdeca3cf5207311350b4f1e9dfae49a9cf227e026bf34c7516dc972f1a330
HMAC-SHA512(text.file)=
5f4a51337956828f9a10ff4402bd46dedcf05f151c347223c9c5fd631d2e5361570ea06e6d534f3
32bda16300970901dd84d238c3759b7c1578e928058e18d46

160 bits key:

cmd: openssl dgst -sha256 -hmac "0123456789abcdefghij" text.file
openssl dgst –sha512 -hmac "0123456789abcdefghij" text.file
result: HMAC-SHA256(text.file)=
144c160724a6385abf3b300c2f998bd891ef997a8b667ddb287ddbd8f5648485
HMAC-SHA512(text.file)=
2a9e8cebc44b961fc88b94917d82f9a08ead4d37a878a78abfa9e626dea7b6c820ca4a0f47cf110
205c4a190ceb53ed9602a2df10eb272e94ec6f7155b5b8de1

256 bits key:

cmd: openssl dgst -sha256 -hmac "0123456789abcdefghijklmnopqrstuv" text.file
openssl dgst –sha512 -hmac "0123456789abcdefghijklmnopqrstuv" text.file
result: HMAC-SHA256(text.file)=
af6f012c269f46fe250d53c560f9a954d5c1a26faecfac54ecc24d278e45f496
HMAC-SHA512(text.file)=
6eb038208ed6153518de73ceb0caadcb1e3a680d179c0ae7f06b91b3de2afcdbf81f754ab010c5
8c1697f1781441ecbbfd204e35dbddaa465cbbaa3be91b52f0

We don't need the fixed key size. If the length of result of hash function SHA256, SHA512 is L, then if the number of bits of key is larger than L, then the algorithm will hash the key first, and get a new key with length of L. If the bit number of key is smaller than L, Then use this key directly

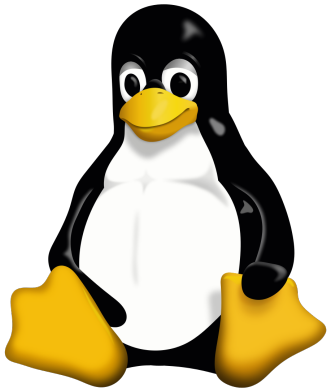
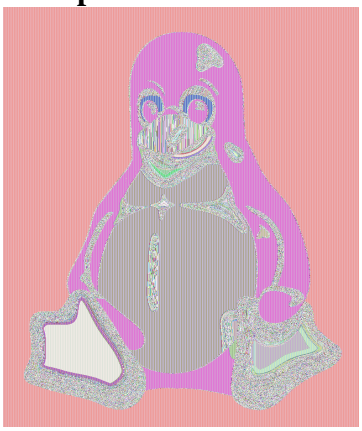
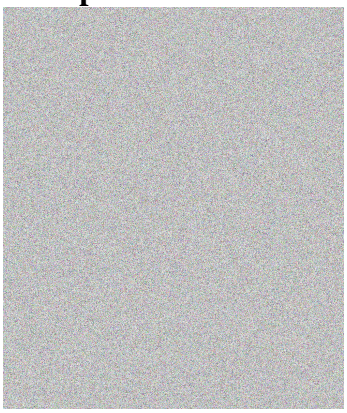
3.3 [10 pts] Encryption using different ciphers and modes

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, for the .bmp file, the first 54 bytes contain the header information about the picture, you have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. Replace the header of the encrypted picture with that of the original picture. You can use a hex editor tool (e.g. ghex or Bless) to directly modify binary files

For ECB mode, we can use the command:
openssl enc -bf-ecb -e -in Tux.bmp -out Tux.ecb.bmp -K
00112233445566778889aabbccddeeff -iv 0102030405060708

For CBC mode, we can use the command:
openssl enc -bf-cbc -e -in Tux.bmp -out Tux.cbc.bmp -K
00112233445566778889aabbccddeeff -iv 0102030405060708

2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Record both encrypted pictures and the original picture for your report for both CBC and ECB modes**.**

Original picture	**ECB picture**	**CBC picture**
		

From the three pictures list above, we can find out, the picture, which encrypted by ECB still left the border area, we still can recognize it's a penguin. However, the picture encrypted by CBC is filled by noisy points, these are because of different enciphrement mechanisms with two modes.

3.4 [10 pts] Observation Task: CBC Encryption using different IVs

Following by the steps, then use "cat" command to show the three contents in the command line, we can get:

openssl enc -aes-128-cbc -e -in plain.txt -out encrypt1.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

Encrypt1.bin: `?C???L??´???b?@??<R? ?`

openssl enc -aes-128-cbc -e -in plain.txt -out encrypt2.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

Encrypt2.bin: `?C???L??´???b?@??<R? ?`

openssl enc -aes-128-cbc -e -in plain.txt -out encrypt3.bin -K 00112233445566778889aabbccddeeff -iv 0807060504030201

Encrypt3.bin: `n?r[Ą????<??^???'?????D?`

1. Does the contents of 'encrypt1.bin' match the contents of 'encrypt2.bin'? Explain why or why not.

The contents of 'encrypt1.bin' match the contents of 'encrypt2.bin'. The reason is that CBS encryption uses initialization vectors to initialize the program which actually performs encryption, same IVs create same encryption program, which create same encrypted data.

2. Does the contents of 'encrypt1.bin' match the contents of 'encrypt3.bin'? Explain why or why not.

The contents of 'encrypt1.bin' does not match the contents of 'encrypt3.bin'. The reason is same as above, different IVs create different encryption program, which create different encrypted data

3.5 [20 pts] These set of questions are based on a real world vulnerability in an open source File Storage application called OwnCloud . Please refer to this blog post for the details about the vulnerability and answer the following questions in brief

1. Please summarize the attack mentioned in the above blog post. Less than 150 words.

When an end-user uploads a file to the Owncloud, the application will perform encryption to the file using AES-256 in CFB-mode. The problem is that the filename is not protected and the way it encrypts the file is in the very beginning of the encryped file. As AES is symmetric, the attacker can flip any bits in the ciphertext, then the same bit will be flipped in the decrypted data. Which means the attacker may be able to control some continuous bits and perform an attack by these bits.

2. What encryption mechanism is being used in the above mentioned cryptosystem?

AES-256 in CFB-mode

3. What shortcoming in the above mentioned cryptosystem and encryption scheme leads to the above mentioned vulnerability?

a. AES is symmetric.
b. Cipher Feedback (CFB) mode is a self-correcting mode.

4. Give an example of an cryptosystem or encryption scheme that could have been used to avoid this vulnerability.

RSA-2048