TP1 Network Security

Building a password manager

18/10/2024

DOAN Alexis - NGUYEN Denis

1) Task 1: Build a CBC Mode Block Cipher and Decipher

1.

```
>> help cipher
'cipher' is a function from the file /home/alexisd/octave/Octave_Workspace/Scripts/AES/cipher.m

CIPHER  Convert 16 bytes of plaintext to 16 bytes of ciphertext.

   CIPHERTEXT = CIPHER (PLAINTEXT, KEY)
   converts PLAINTEXT to CIPHERTEXT using the key KEY

   PLAINTEXT has to be a vector of 16 bytes (0 <= PLAINTEXT(i) <= 255).
   KEY has to be a vector of 16 bytes (0 <= KEY(i) <= 255).


Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
```

```
>> help inv_cipher
'inv_cipher' is a function from the file /home/alexisd/octave/Octave_Workspace/Scripts/AES/inv_cipher.m

INV_CIPHER  Convert 16 bytes of ciphertext to 16 bytes of plaintext.

   PLAINTEXT = INV_CIPHER (CIPHERTEXT, KEY)
   converts CIPHERTEXT (back) to the plaintext PLAINTEXT using the key KEY

   CIPHERTEXT has to be a vector of 16 bytes (0 <= CIPHERTEXT(i) <= 255).
   KEY has to be a vector if 16 bytes of bytes (0 <= KEY(i) <= 255).


Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
```

2.

```
% Define two 16-byte secrets keys in ASCII
key_ascii1 = 'sinjSINJsinjSINJ';
key_ascii2 = 'Die with a smile';

% Their nuerical representation
key_ascii1_conv=double(key_ascii1);
key_ascii2_conv=double(key_ascii2);

% Plaintext in ASCII format
char='this is my Password Manager';

% Creation of a nonce
nonce=num2str(cputime*1e12);
```

```
Command Window
  key_ascii1 =

      'sinjSINJsinjSINJ'


  key_ascii1_conv =

     115   105   110   106    83    73    78    74   115   105   110   106    83    73    78    74
```

```
nonce =

    '71130000000000'
```

Note : We will need the **nonce** to be of length 16, which is not always the case if we use **cputime**. We will implement a padding on **nonce** for that.

```
% Force nonce to have 16 elements
nonce_pad_len = 16-length(nonce_ascii);
nonce_ascii_16 = [nonce_ascii,zeros(1,nonce_pad_len)];
```

3 + 4.

Estimation of the padding length :

```
function pad_len = padEvaluate(plaintext_ascii)

  %%%%%%%%%%%%%%%%%%%%%%%%% initializations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  block_len = 16;                       %length of an AES block in bytes
  padded_plaintext = [];                %initialization

  %%%%%%%%%%%%%%%%%%%%%%%%% add padding %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  len=length(plaintext_ascii);        %length of the plaintext
  pad_len=16-rem(len, 16);            %length of padding in bytes

  %padded plaintext using the TLS convention
  padded_plaintext=double([plaintext_ascii, repmat(pad_len,1, pad_len)]);

  outString = ['For plaintext : ',plaintext_ascii,'\n'];
  printf(outString);
  printf('Padding length : %o\n',pad_len);
endfunction
```

```
For plaintext : this is my Password Manager
Padding length : 5
```

Encryption and Decryption of CBC :

```
function ciphertext = encryptCBC(plaintext_ascii,key,IV,pad_len)
  %%%%%%%%%%%%%%%%%%%%%%%%% initializations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  block_len = 16;                       %length of an AES block in bytes
  padded_plaintext = [];
  ciphertext = [];

  %%%%%%%%%%%%%%%%%%%%%%%%% add padding %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % padded plaintext using the TLS convention
  padded_plaintext = double([plaintext_ascii, repmat(pad_len,1, pad_len)]);

  % length of the CBC chain
  num_chain = length(padded_plaintext)/block_len;

  %%%%%%%%%%%%%%%%%%%%%%%%% Encryption %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Creation of the blocks
  padded_plaintext_res = reshape(padded_plaintext,block_len,num_chain)';

  % Initialisation
  xorVar=IV;

  for row = 1:num_chain
    encrypt = bitxor(padded_plaintext_res(row,:),xorVar);
    xorVar = cipher(encrypt,key);
    ciphertext = [ciphertext,xorVar];
  end

endfunction
```

```
function recovered_plaintext = decryptCBC(ciphertext,key,IV)
  %%%%%%%%%%%%%%%%%%%%%%%%% Initializations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  block_len = 16;                       %length of an AES block in bytes
  recovered_plaintext = [];

  %%%%%%%%%%%%%%%%%%%%%%%%% Decryption %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  cipher_res = reshape(ciphertext,block_len,[])';
  xorVar = IV;
  dim = size(cipher_res);

  for row = 1:dim(1)
    nextXorVar = cipher_res(row,:);
    decryptCipher = inv_cipher(nextXorVar,key);
    recovered_plaintext = [recovered_plaintext,bitxor(decryptCipher,xorVar)];
    xorVar = nextXorVar;
  endfor

  recovered_plaintext = reshape(recovered_plaintext,1,[]);

endfunction
```

Now we apply this with our inputs :

```
doublePlaintext =
 Columns 1 through 13:
   116   104   105   115    32   105   115    32   109   121    32    80    97
 Columns 14 through 26:
   115   115   119   111   114   100    32    77    97   110    97   103   101
 Column 27:
   114
```

```
ciphertext =
 Columns 1 through 13:
   162     3   225    30   115    53   157    30   241    64   105   146   202
 Columns 14 through 26:
    55    28    57   171   247    56   120   172    44   143   111   192   139
 Columns 27 through 32:
   204   214   200    54    22   243
```

```
recovered =
 Columns 1 through 13:
   116   104   105   115    32   105   115    32   109   121    32    80    97
 Columns 14 through 26:
   115   115   119   111   114   100    32    77    97   110    97   103   101
 Columns 27 through 32:
   114     5     5     5     5     5
```

```
recovered_char = this is my Password Manager
>>
```

The initial plaintext is correctly recovered (with spaces at the end due to the padding).

5.

a. What are the main blocks inside the AES-128 block cipher?

- SubBytes : we use a S-box (non linearity) to substitute each bytes of the block
- ShiftRows : the rows of the blocks are shifted cyclically
- MixCollumns : in each blocks, we combine the four bits of each columns
- AddRoundKey : each block is xored with a rounded key derived of an encryption key

What is the role of an S-box?

A S-box introduces non linearity into the cipher. It maps each byte to a corresponding substitute byte based on a non linear transformation.

b. What is the role of the nonce in CBC mode?

Its role is to ensure that the same plaintext, when encrypted multiple times with the same key, will not give the same ciphertext.

c. Why does the pair (key, nonce) have to be different in every encryption? What happens if this requirement is not met?

It is to prevent patterns from emerging. If this requirement is not met, the encryption scheme becomes vulnerable to several cryptographic attacks, compromising confidentiality.

d. Is the CBC mode semantically secure?

It can be if the nonce is renewed at each utilisation.

2) Task 2: Building a CBC-MAC

3. CBC-MAC building :

```
function mac = generateMAC(plaintext,key1, key2)

% Definition of constants
num_chain= 4 ;
block_len = 16;

key1_double=double(key1);
key2_double=double(key2);

% add padding
len=length(plaintext); %length of the plaintext
pad_len=4*16-len; %length of padding in bytes
padded_plaintext=double([plaintext, repmat(pad_len,1, pad_len)]);

% Creation of the blocks
padded_plaintext_res = reshape(padded_plaintext,block_len,num_chain)';

% calculate mac
xorVar=padded_plaintext_res(1,:);
for row = 1:num_chain
    encrypt=bitxor(xorVar,padded_plaintext_res(row,:));
  xorVar = cipher(encrypt,key1_double);
end
ciphertext = [xorVar];

mac = cipher(ciphertext,key2_double);

end
```

4+5. MAC authentication on ciphertext from task 1 :

```
cipherMAC =

    23    76   123   177   180    97    43    32    91   135   122   184   113    82   232    51
```

6. MACing the original plaintext :

```
plainMAC =

    41   139   223   191    15    13    85   250    94   237   163   191    49   125   120   236
```

Authentication tests with same and different key :

```
function response = isAuthenticated(ciphertext,MAC,keyMAC1,keyMAC2)
  MAC2 = generateMAC(ciphertext,keyMAC1,keyMAC2);

  if MAC == MAC2
    printf('Athentification : Yes\n');
    outString = [keyMAC2,' is the right key\n'];
    printf(outString);
  else
    printf('Athentification : No\n');
    outString = [keyMAC2,' is the wrong key\n'];
    printf(outString);
  endif
endfunction
```

```
% tests with the same and different keys
isAuthenticated(plaintext,MAC,keyMAC1, keyMAC2);
isAuthenticated(plaintext,MAC,keyMAC1, keyMAC3);
```

```
% Define two 16-byte secrets keys in ASCII
keyMAC1 = 'cla vier mecanic';
keyMAC2 = 'la bete est laaa';
keyMAC3 = 'hle < blg : wow!';
```

```
Athentification : Yes
la bete est laaa is the right key
```

```
Athentification : No
hle < blg : wow! is the wrong key
```

7.

a. The more secure mode is **[ciphertext || MAC(ciphertext)]**
   - Only the ciphertext and MAC are transmitted, the plaintext is not exposed (Confidentiality).
   - If the ciphertext is modified, the MAC will not match during verification, detecting any tampering (Integrity).
   - The MAC ensures that only parties who know the secret key can generate the correct MAC (Authentification).

b. To maintain semantic security, **AES-128 in CBC-MAC mode** can securely authenticate $\mathbf{2^{64}}$ **blocks** of data before requiring a key change (birthday bound : collisions probability become non-negligible after 2^{64} blocks).