Lab 1
Building a secure password manager

Network security and cryptographic principles

November 2018

Dr. Arsenia Chorti

Outline: The aim of this lab is to use message authentication codes (MAC) and authenticated encryption (A.E.) schemes to build a basic password manager scheme. The central idea is to use a MAC and an A.E. to store safely in your PC's hard drive a database containing your passwords. By encrypting the password database with a single "strong" key, a user would be able to store a large number of passwords without having to compromise their "strength" (e.g., to ease their memorization). At the first part of this lab, you will use a baseline MATLAB implementation of the AES-128 block cipher to build a cipher-block-chaining (CBC) mode encryption, as wells as the corresponding decryption. At the second part of the lab you will build a MAC in CBC mode and the corresponding verification algorithm. At the third part of the lab you will build a password manager using the MAC and the A.E. schemes to protect your password database. At the end of the lab you will be able:

- Understand how to build a semantically secure block cipher in CBC mode, using the AES-128 block cipher
- Understand how to build a semantically secure MAC in CBC mode, using the AES-128 block cipher
- How to combine the two to build an A.E. scheme
- How to use a MAC and an A.E. to store confidential data in your hard drive

Scope

One of the weak points of online security software lies in the potential use of “weak” user passwords. Recent studies have highlighted that it is not uncommon for users to choose passwords that are susceptible to dictionary attacks; some alarming examples of such practices include passwords such as “123456” or “password1”. On one hand, in terms of online security, it is widely accepted that it is very important that users choose “strong” passwords to secure their access and login details. On the other hand, it is also undeniable that there's a maximum complexity to the passwords people are willing to memorize. On top of that, an average user of online services and apps must typically remember a large number of different passwords.

A simple approach to reconcile these contradictory requirements is to use a password manager; for example, Chrome offers such a service under the name LastPass, while KeePass is an open source alternative. Password managers rely on the idea that users can remember a single, “strong” password to securely store a database containing the actual passwords they wish to use in different applications. Therefore, a central task in building a good password manager is to have a provably secure scheme for storing confidential data, as well as offering guarantees against tampering attacks in case an adversary obtains access to your storage memory (in this TP the hard disc of the PC you will be working on). In this lab session, you will build a secure password manager using a message authentication code and an authenticated encryption scheme. You will employ a MATLAB® implementation of AES-128 primitive as your building block.

Note that the implementation of the AES-128 primitive you are provided with is intended for educational purposes **only.** In general, it is a bad idea to build your own implementation(s) of cryptographic primitives; you are rather advised to use well established libraries. Furthermore, in this lab session we will not engage in securing the password manager against many other possible attacks, including timing, side channel attacks, etc.

Task 1: Build a CBC Mode Block Cipher and Decipher

1. Open the Module Moodle. Under the folder "Lab1" you will find a zip file named AES.zip. Download it **to your local MATLAB folder** (the folder you will be working on) and unzip it. Among the various Matlab files (with extension ".m") you will find the function

   ciphertext = cipher (plaintext, key)

   which implements the AES-128 block cipher primitive and returns the ciphertext of the plaintext using the provided key. Furthermore, the function

   plaintext = inv_cipher (ciphertext, key)

   implements the decryption of the AES-128 block cipher.

   ⇨ The AES-128 block cipher is based on the MATLAB® implementation of the Advanced Encryption Standard by J.J. Buchholz (copyright 2000-2005).

   On your command window type
   >> help cipher
   and inspect the help file of the function cipher.
   On your command window type
   >> help inv_cipher
   and inspect the help file of the function inv_cipher.

2. Open a new script file and save it under the name "*yourname*_encipher.m". Define **two secret** 16-byte keys of your choice in ASCII format (i.e., as texts, each of 16 characters long).
   In this TP, do not use special characters (only alphanumeric).
   As an example, you could define the following two keys. Note that the spaces count as ASCII symbols and that each ASCII symbol corresponds to 1 byte in numerical representation.
   >> key_ascii1=This is just fun';
   >> key_ascii2='Isoloveblueskies';
   To convert the keys from ASCII format to numerical format you can use the function "double".
   Choose a plaintext of your choice in ASCII format, i.e., as text. As an example, you could use the following plaintext
   >>m_char='this is my Password Manager';
   Finally, define a 16-byte nonce of your choice. This nonce should be updated every time you call the encryption routine, so a good idea is to implement it as a timestamp, e.g., using the cpu time.

3. In the first task of this lab you are asked to implement a block cipher in CBC mode as shown in Fig. 1, below, and the corresponding decipher shown in Fig. 2, for a 2-block long message. An important step in your implementation is to correctly break the message into 16-byte blocks and to perform padding in the last block as necessary. For padding, use the approach employed in TLS.

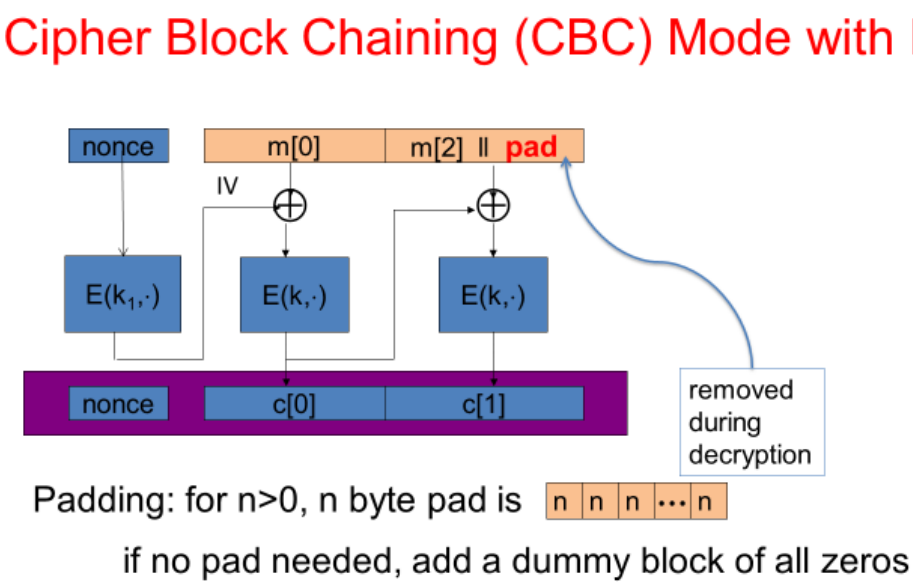

Figure 1: CBC mode with nonce

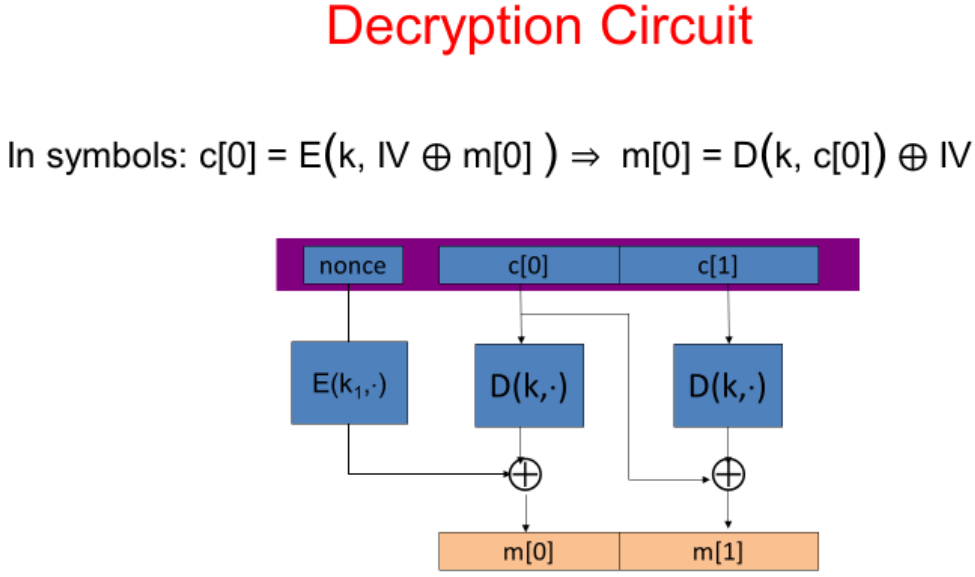

Figure 2: Decipher circuit of CBC mode with nonce

Some useful MATLAB functions that you might choose to use are listed below:

- repmat
- rem
- zeros
- char, double
- bitxor
- cputime

To get you started, use the following MATLAB® script to evaluate the length of the padding

```
clear all
clc

%%%%%%%%%%%%%%%%%%%%   initializations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
block_len=16;                        %length of an AES block in bytes
key_ascii='Good weather now';        %a secret key for encryption
key=double(key_ascii);               %convert from ASCII format to double
key_nonce_ascii='Have you secrets';%key to encrypt the nonce
key_nonce=double(key_nonce_ascii); %convert from ASCII format to double
nonce_ascii=num2str(cputime*1e12); %create a nonce as a counter
nonce=double(nonce_ascii);           %convert nonce to double
IV = cipher(nonce, key_nonce);       %generate the IV
padded_plaintext=[];                 %initialization
ciphertext=[];                       %initialization
recovered_plaintext=[];              %initialization
plaintext_ascii='embezzled pizzalike buzzwords';

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

4. Encrypt your chosen plaintext using your chosen keys and nonce in the CBC mode cipher. Then decrypt it using the corresponding decryption. Validate your results. Report the values for all encryption parameters you used.

5. Answer the following questions:
    a. What are the main blocks inside the AES-128 block cipher? What is the role of an S-box?
    b. What is the role of the nonce in CBC mode?
    c. Why does the pair (key, nonce) have to be different in every encryption? What happens if this requirement is not met?
    d. Is the CBC mode semantically secure?

Task 2: Building a CBC-MAC

1. In this task, you are expected to build a CBC-MAC and the corresponding verification algorithm using the baseline AES-128 function. The CBC-MAC is depicted in Fig. 3 for a four-block message. In the following assume that the pseudorandom function F(k,·) is the AES-128 block cipher.

2. Open a new script file and save it under the name "*yourname*_MAC.m". Choose two **new secret keys** of 16 bytes each.

3. Build the CBC-MAC depicted below.

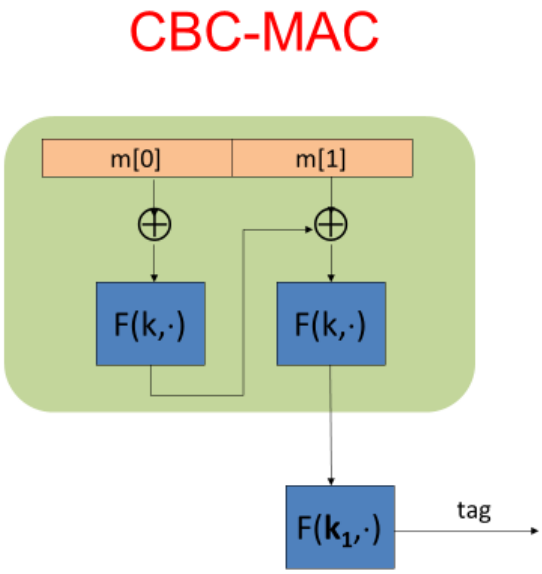

Figure 3: CBC-MAC

4. Use this MAC to authenticate the ciphertext you generated in Task 1 and write down the generated MAC. To pad a message to be MACed in CBC mode, you can use the same approach as that in Task 1.

5. Use the MAC to authenticate the ciphertext you generated in Task 1.

6. Repeat the procedure by MACing the original plaintext instead.

7. Answer the following questions:
   a. Which mode do you believe is more secure?
   [ciphertext || MAC (ciphertext)] or [plaintext || MAC(ciphertext)]?
   Explain your answer.
   b. How many blocks can be MACed with an AES-128 in CBC mode MAC without changing the key(s), before breaking semantic security?

Task 3: Building a basic password manager

1. In general, a password manager will store a passwords database on some storage (e.g., your hard disk), encrypted and MACed with a strong master password.[1] For example, a sample password manager instance might store the following information:

Table 1

Web service	Password
www.amazon.com	thisisagreatsunnydayinthewinter
www.spotify.com	I_ like_to_chill_out_with_music
www.ebay.com	SellingEverythingForJustAPenny

Naturally, saving in the clear this information in an EXCEL file in your PC's hard disk is not secure. In this Task, you will need to preserve
a) the integrity of the "Web service" list
b) the confidentiality and the integrity of "Password" list.
You will achieve these goals by using the schemes you built in Tasks 1 and 2.

2. We want to maintain the ability to search for the data corresponding to a specific entry in the column "Web service". In this implementation, the "Web service" list will be MACed only. Then, to look up the data corresponding to "Web service" X, you first will need to compute MAC(k,X), (where k is part of the master secret key) and check whether the result exists as an entry in the MACed list "Web service". Write the relevant MATLAB® code for the entries in Table 1.

3. The passwords themselves are encrypted using A.E. Write the relevant MATLAB® code.

> To accommodate a potentially large number of entries in the password manager, you should encrypt and store each record individually. In other words, it is not advisable to encrypt the entire list as a single "blob". This way, you will not have to decrypt every entry in the password manager when fetching a single record.
>
> We also do not want to leak any information about the domains the user has stored in the password manager. For this reason, assume that all entries are by default 32 bytes long.

4. Answer the following questions:
   a. When designing any system with security goals, it is important to specify a threat model: i.e., we must precisely define the power of the adversary, as well as the condition the adversary must satisfy, to be considered to have "broken" the system. What is the threat model your password manager can withstand?
   b. A particular type of attack for password managers is the so-called "swap" attack, in which an adversary gains access to your hard disk and swaps the contents of

[1] Instead of implementing a full standalone password manager application, for this lab you will only be responsible for the core library. Thus, you will not need to implement the GUI for interacting with the password manager, nor will you need to write the contents to disk.

entries in the “Web service” list. As an example, an adversary might swap www.amazon.com for www.spotify.com, so that when you believe you retrieve the password for amazon you are in reality retrieving the password for spotify. Propose a simple approach to overcome such an attack.

c. Another possible attack is the so-called “roll-back” attack in which the adversary can replace an entry with a previous version of the respective entry (e.g., assume you changed your password for www.ebay.com , but an adversary has kept a record of an older entry and used it to roll it back). Propose an approach to secure your password manager against this type of attack.