TP 7 : HttpD

Submission instructions

At the end of the practical, your Git repository must follow this architecture:

```
csharp-tp07-firstname.lastname/
|-- README
|-- .gitignore
|-- HttpD/
    |-- HttpD.sln
    |-- HttpD/
        |-- Bonus.cs
        |-- Encryption.cs
        |-- HttpClient.cs
        |-- MiniHttps.cs
        |-- Program.cs
        |-- curl.cmd
        |-- req01
        |-- req02
        |-- req03
        |-- Everything except bin/ and obj/
```

Do not forget to check the following requirements before submitting your work:

- You shall obviously replace `firstname.lastname` with your login.
- The `README` file is mandatory.
- There must be no `bin` or `obj` folder in the repository.
- You must respect the prototypes of the given and asked functions.
- Remove all personal tests from your code.
- **The code MUST compile !**

README

In this file, you can write any and all comments you might have about the practical, your work, or more generally about your strengths and weaknesses. You must list and explain all the bonuses you have implemented. An empty `README` file will be considered as an invalid archive (malus).

1 Introduction

1.1 Objectives

The goal of this tutorial is to help you understand and manage HTTP requests. You will also have to manage a new encryption algorithm [1], which we will use in this pratical the AES. Once you have mastered these notions, you will be able to understand and implement a highly simplified version of HTTPS.

2 Course

2.1 HTTP

2.1.1 A little context

HTTP is a client-server communication protocol. It is a protocol which is located on the applicative layer of the OSI model[2]. HTTP stands for *Hyper Text Transfer Protocol.* The version we will use in this tutorial is the HTTP 1.1 which is described in an exhaustive way by the RFC 2616[3]

2.2 A simple HTTP request

During this tutorial, we will only deal with the client part of an HTTP exchange.

```
GET / HTTP/1.1
Host: www.example.com
Accept: text/html
User-Agent: Votai-Test
```

What you have above is an HTTP request sent via the command **curl**[4]. We will deconstruct the request line by line

line 1 : **GET** is the HTTP method used[5]. In our case GET means that we ask to retrieve a resource from the server. The / represents the path to this resource. **HTTP/1.1** represents the HTTP version used

line 2 : **Host** is a mandatory parameter which indicates the server address.

line 3 : **Accept** is a parameter that specifies the type of content accepted, here we expect an html code.

line 4 : **User-agent** This field contains the "identity" of the sender (which browser for example)

The HTTP request we sent contains the 4 mandatory elements of a valid request (Method, Path, Version, Host). However you can add many other headers[6].

In an HTTP request you can also send data via the **POST** method for example. In order to send data you just have to leave an empty line between all your headers and the data.

[1]Remembering the AFIT
[2]https://en.wikipedia.org/wiki/OSI_model
[3]https://datatracker.ietf.org/doc/html/rfc2616/
[4]You can access the documentation of the curl command by typing man curl
[5]https://datatracker.ietf.org/doc/html/rfc2616/#section-5.1.1
[6]https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

```
POST / HTTP/1.1
Host: all-students-of-epita.com
Content-Type: text/html
Content-Length: 20

<h1>Votai Test.</h1>
```

In this example we send the HTML code ”<h1>Votai Test.</h1>” to the server. Notice the line **Content-Length** which specifies the size of the data sent in bytes.

2.3 An HTTP response

```
HTTP/1.1 200 OK
```

A response is formed in the same way as a reply; there are just different headers. The main difference is that instead of the **method** there is the **response code**[7].

The body of the answer contains the data like the HTML code of a page or a picture.

2.3.1 HTTP Parameter

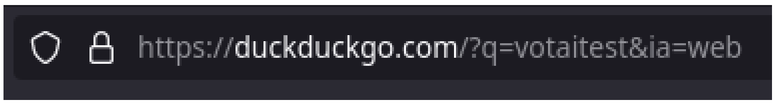
https://duckduckgo.com/?q=votaitest&ia=web

Figure 1: A URL with two parameters

It is also possible to pass parameters via the GET method. They take the form $var = value$ and are placed after the **?** character. In this example, we pass the parameters **q** and **ia** with the respective values ”votaitest” and ”web”.

2.3.2 URL encoding

Some ASCII characters cannot be directly written in HTTP parameters because they are already used in the URL format, for example the /. To overcome this problem there is a URL encoding which allows to pass ”forbidden” characters. Let’s take the example of / again, how to use the URL encoding to pass this character in a parameter.

1. The ASCII hexadecimal value of / is 2F
2. A % is added before the value
3. / => **%2F**

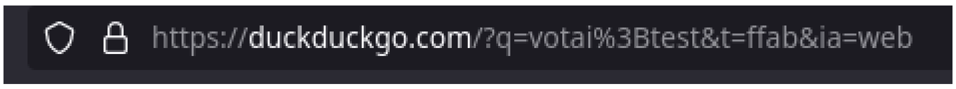
https://duckduckgo.com/?q=votai%3Btest&t=ffab&ia=web

Figure 2: A URL with a parameter that contains a text in the q parameter

[7]the famous 404 is an HTTP response code

Important

For this tutorial you will probably need to encode your parameters. To do this you can use this online encoder `https://www.urlencoder.io/`

2.4 Advanced Encryption Standard[8]

The Advanced Encryption Standard is a symmetrical encryption algorithm based, among other things, on a private key of predefined length before the start of the communication between the peers. It is, to date, considered the most secure encryption algorithm. It is derived from the algorithm of *Rijndael*. The AES keys are represented in **Base64**[9]. Each message you want to encrypt is accompanied by an initialization vector (IV). This IV is used to add a random aspect to the chunking and encryption of the blocks. It must be generated in a random/semi-random manner.

The algorithm takes into account a 4x4 matrix:

$$A = \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

To this matrix, a key of 128, 192 or 256 bits is added. It will then be a question of executing, during a number of stages defined by the size of the key, a given algorithm. At each iteration, a new key is derived from the initial key to accomplish the tasks.

The algorithm proceeds in this way:

1. At the first iteration, we apply to each element of the matrix (which is in practice a byte) a `xor` with another byte of the key associated to the iteration.
2. If the current iteration is neither the first nor the last:
   (a) The substitution step is applied to the matrix, this simply consists in applying a transformation to each element, we will then have a matrix B with $b_i = S(a_i)$ where S represents the transformation in question.
   (b) Left rotation step is then applied sequentially to each row. It consists in making a left rotation of $n-1$ element at line n. Thus, the first line will not undergo any rotation. The last line (the fourth) will be rotated by 3 elements.
   (c) A matrix multiplication[10] is then applied to the matrix B, it is simply multiplied by

$$M = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

   (d) The famous `xor` is performed on the matrix as described in the first step.
3. For the last iteration, only the steps *a*, *b* and *d* of the preceding case will be carried out.

[8] In French we say chiffrer not crypter
[9] urlhttps://en.wikipedia.org/wiki/Base64
[10] In keeping with the idea of staying on bytes, modulo 256 will be applied to this multiplication

Note that for decryption, it will be enough to do the inverse operations (inverse order, inversion of M, ...).

2.4.1 Binary operators

For AES you will need to master the basic binary operators.

We'll start by looking at the bit shift operators » and «. The » operator shifts all value's bits to the right while « shifts to the left. The bits that are inserted are set to 0 and the bits that go beyond are lost. You can rely on the following examples. You can check out the following examples.

```
ushort chetor = 65; // 0000 0000 0100 0001
chetor = chetor << 2; // 0000 0001 0000 0100
chetor = chetor >> 3; // 0000 0000 0010 0000
```

Tip

You can read the MSDN documentation `https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/operators/right-shift-operator`

There are two other very used binary operators: the binary AND *'&'* and the binary OR '||'.

```
ushort chetor = 65; // 0000 0000 0100 0001
chetor = chetor & 64;
// 0000 0000 0100 0001 & 0000 0000 0100 0000
// chetor = 0000 0000 0100 0000
chetor = chetor | 65;
// 0000 0000 0100 0000 | 0000 0000 0100 0001
// chetor = 0000 0000 0100 0001
ushort PR = 17; // 0000 0000 0001 0001
PR = PR & 15; // 0000 0000 0000 0001

/*
* The last example is a filter that keep
* the last four bits of the variable PR
*/
```

The & and the | operators respectively perform a binary AND or a binary OR between the bits of two values.

Tip

You can read the MSDN documentation `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#logical-and-operator-`

To continue with the binary operators you will (re)-discover the XOR.

```
ushort votaitest = 17; // 0000 0000 0100 0001
votaitest = votaitest ^ 1;
// 0000 0000 0100 0000
```

Tip

You can read the MSDN documentation ***`https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#logical-and-operator-`***

3 Exercises

3.1 Exercise 1

*The aim of the exercise is to use the **curl** command to retrieve a resource at the following address : `https://photos.cri.epita.fr/vlad.argatu`. The retrieved content is an image.*

Important

You will have to explain in your README how you found the curl command. Moreover you will have to copy in a file named curl.cmd which command you used.

Tip

The resource you are going to retrieve is an image you can add "> ACDC.jpg" at the end of your command to then display it with feh.

3.2 Constructor

Let's get down to business. Here you'll have to fill in the constructor of the HttpClientStream class.

Note that everything will depend on the variable _httpClient that you have to assign and give it a basic Uri. Don't forget to clear all the default headers of the variable.

```
public HttpClientStream(HttpMethod method, string url="http://127.0.0.1:8000");
```

3.3 SendMessage

```
public StreamReader SendMessage(string message);
```

Your role here is to send a request Method by including the message in the body. You must return a StreamReader

Tip

*Use the **HttpRequestMessage** class for the request and the **ReadAsStream** function to return a response to the request as a StreamReader.*

3.4 ResponseFromStream

```
public string ResponseFromStream(StreamReader stream);
```

The objective here is to read the content of the stream and return the corresponding string.

Important

Don't forget to close the stream before exiting the function.

4 Encryption

Before we can send our messages, it is important to secure their content so that that not anyone can access them. We are therefore going to develop all the functions allowing us to encrypt and decrypt messages using a private key which will be used with the AES algorithm.
As seen in the course, AES is based on XOR operations on the given key by performing successive rotations. You will find in the provided code, all the byte-array used to perform the rotations of the key.
Although this part may seem daunting, you will be guided all along in order to implement AES in the best way. Also note that we will only cover AES-128 bit encryption (with 11 key rotations).

Tip

Throughout this part, we will divide the tasks to be done into small functions. You won't necessarily understand what you are doing without reading the documentation.
It is therefore highly recommended that you read the course at least once, as well as the Wikipedia page dedicated to this subject:
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

4.1 SubBytes

```
private static void SubBytes(byte[] a);
```

```
private static void SubBytesInv(byte[] a);
```

Here you must replace each element of 'a' with the value corresponding to that element in the byte-arrays LookupSbox *and* LookupSboxInv *for the SubBytes and SubBytesInv functions respectively.*

4.2 KeySchedule

```
private static void KeySchedule(byte[] a, int i);
```

We are going to implement here the heart of the Key Schedule algorithm : [11] *Here you must make a left rotation of the 4 bytes contained in a, then apply SubBytes and make an XOR between the byte and the LookupRcon table at index i.*

4.3 Xor

```
private static void Xor(byte[] a, List<byte> b, int offset);
```

This function will be used for Xor operations between 2 arrays. You must therefore execute for each byte of a an XOR with the byte of b taking into account the offset in the index of b.

4.4 Expand

```
private static byte[] Expand(byte[] key);
```

[11] https://en.wikipedia.org/wiki/AES_key_schedule

***Using the last two functions**, you must implement the entire AES Key Schedule algorithm [12]. The goal is to extend the 128-bit AES key (in byte array format) by applying the 11 rotations shown. The function must return a byte array of size 176 (11 * 16 bytes) containing the 11 rotations of the key in succession.*

Tip

In the algorithm that you will find on Wikipedia:

bullet N corresponds to 4 in our case (the 128 bits keys make 4 words of 4 bytes).

bullet K_i corresponds to the 16 bytes of the key passed in parameter.

bullet R corresponds to 11 in our case (11 rotations for the 128 bits keys).

*bullet W_i corresponds to the temporary list of 11 * 16 bytes mentioned above.*

4.5 XorCipherWithRoundKey

```
private static void XorCipherWithRoundKey(byte[] cipher, byte[] keys,
                                          int round);
```

Here you must apply an XOR operation on each byte of the cipher with the corresponding byte of the key at the round*-th rotation.*

4.6 ShiftRows

```
private static void ShiftRows(byte[] cipher);
```

```
private static void ShiftRowsInv(byte[] cipher);
```

*Here you must replace each byte with the byte at the index given by the array **ShiftRowsTable**. (Do the same thing with the array **ShiftRowsTableInv** for the second function.*

4.7 MixCol

```
private static void MixCol(byte[] cipher, int offset);
```

*Here you must mix the columns (the 4 bytes composing a 32 bits word starting from the offset). You must therefore for each byte of the word successively apply XORs between the table **LookupG2** on the byte that we modify, the table **LookupG3** on the next byte, and the 2 following bytes.*

Tip

If you are already on the last byte of the word, the "next" byte indicated above corresponds to the first byte of the word.

[12] `https://en.wikipedia.org/wiki/AES_key_schedule#The_key_schedule`

4.8 MixColInv

```
private static void MixColInv(byte[] cipher, int offset);
```

The principle is the same as the previous function but this time you must apply the XOR between the successive tables ***LookupG14*** *on the byte that we modify and then with the next bytes in reverse order on the tables* ***LookupG9***, ***LookupG13*** *and* ***LookupG11***

4.9 MixCols

```
private static void MixCols(byte[] cipher);
```

```
private static void MixColsInv(byte[] cipher);
```

You must apply the previous function for each word of 4 bytes contained in the 16 bytes cipher. Set the offset at each call of ***MixCol***. *Also reproduce this function with* ***MixColInv***

4.10 Encrypt

```
public static byte[] Encrypt(byte[] message, byte[] key);
```

You've almost finished! All you have to do now is to put together everything you've done so far to encrypt your message with the indicated key.
If you have read the documentation properly, you will easily know how to put each function end to end in the right order : [13] *!*

You have to encrypt then a 16 bytes message with a 16 bytes key and send back the encrypted message (without modifying the original message). The encrypted message should also be 16 bytes long.

4.11 Decrypt

```
public static byte[] Decrypt(byte[] cipher, byte[] key);
```

Now that you know how to encrypt, you should be able to decrypt your own messages. And this is a good thing, with AES, decrypting a message is just as easy as encrypting it. You have noticed that since the beginning, we have copied several functions by modifying only the tables for mixer and shifter. These tables allow you to reverse the AES encryption, as long as you have the same key as the one used to encrypt the original message.

You just have to reproduce the Encrypt function by using the Inv functions instead of the original ones and reversing the execution order of the Encrypt function.

4.12 ToBytes

The purpose of this function is to transform a string of hexadecimal characters into a byte array. The string must contain an even number of characters (it is a sequence of valid bytes in hexadecimal format).

```
static byte[] ToBytes(string str);
```

Example with the string "BABE"

```
[0xBA, 0xBE]
```

[13] `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#High-level_description_of_the_algorithm`

4.13 Pretty

You have to implment the function Pretty that prints a byte array.

```
static void Pretty(string label, byte[] a);
```

Example: label = "Test." et a = [0x01, 0x05, 0x1F]

```
Test.:
01-05-1F
```

5 MiniHttps

You have almost completed this tutorial. Now you will have to use your encryption and http messaging features to send messages via http securely and receive a secure response.

5.1 Constructor

You must initialize the attributes _client and _key of the MiniHttps class.

```
public MniHttps(string url, string key);
```

5.2 EncryptFullCipher

You know how to encrypt 128-bit messages. But what if your message is longer? You have to encrypt each 128-bit part of the message separately and then concatenate them to form the entire encrypted message.

```
public byte[] EncryptFullCipher(string message);
```

5.3 DecryptFullMessage

```
public string DecryptFullMessage(byte[] cipher);
```

Now do the same thing but decrypt a cipher to return the original message string.

5.4 EncryptAndSend

```
public string EncryptAndSend(string message);
```

For this last function, you must encrypt a message, send it to the remote server, and then send back the decrypted answer.

6 Pawn me daddy !

6.1 Objectives

The purpose of this section is to use your HTTP client to resolve three small CTFs. The application that you are going to "attack" is an application developed by OWASP[14]. If you're interested in the rest of this tutorial, you'll find a lot of resources on the organization's website to go further.

[14] `https://owasp.org/`

Important

You are asked to put in "reqXX" files the URLs that allowed you to succeed in the exercise. For example, if the URL ***https://duckduckgo.com/?q=votaitest&ia=web*** *allows you to succeed in an exercise, it is the latter that you must put in the FIRST LINE of the associated req file.*

6.2 File injection

The goal of this exercise is to write an HTTP request to read the contents of the file ***/etc/passwd****. To do this, you must send a request with the* ***GET*** *method to the following address:* **`http://httpdtest.ml/vulnerabilities/fi/?page=include.php`**[15]*. You will copy your request to the file* ***req01*** *and explain how you find it.*

6.3 Command injection

This time you will have to execute a command on the server via a request with the ***GET*** *method. The address is* **`http://httpdtest.ml/vulnerabilities/exec?ip=0.0.0.0&Submit=Submit`**[16]*. You will need in the file* ***req02*** *your request as well as the contents of the file flag_2.*

6.4 SQL Injection

For the last exercise you will have to recover the password of the account ***Test.****. To do this, you will have to carry out an SQL injection via the* ***id*** *parameter,* [17]*. Here is the URL you must use :* *`http://httpdtest.ml/vulnerabilities/sqli/?id=2&Submit=Submit`*.

Important

The passwords are stored in the form of a hash, but you are expected to retrieve the password in clear text. The hash method used is an old and easily breakable method (Hint: 5)

Just like the other two exercises, you will give the query in the file ***req03****.*

7 Bonus

AES keys are used in binary format to perform encryption and decryption operations. A simple representation of the binary is Base 64.[18] *Indeed, it makes it possible to encode a binary number of 24 bits (3 bytes) in 4 characters contained in the alphabet of the base 64.*

One finds thus the upper case letters, lower case letters, the numbers as well as the 2 characters + and / in this order. (A corresponding to 0 and / corresponding to 63. Finally the character = which allows to complete the string in base 64 if the conversion does not result in a length multiple of 4. This operation is called padding. [19]

[15]`https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion`

[16]`https://owasp.org/www-community/attacks/Command_Injection`

[17]`https://owasp.org/www-community/attacks/SQL_Injection`

[18]urlhttps://en.wikipedia.org/wiki/Base64

[19]urlhttps://en.wikipedia.org/wiki/Base64Output$_{p}$*adding*

7.1 StringToBase64

```csharp
public static string StringToBase64(string s);
```

Here you must encode a string in its base 64 equivalent.

Tip

You must use the shift binary operators to convert the characters of the string to base 64.

7.2 Base64ToString

```csharp
public static string Base64ToString(string s);
```

Here you have to decode a string in base 64 to its original string.

Tip

You must throw an exception if the string is not in a valid base 64 format.

8 Erratum

We could not see the server part in this exercise, so you are only able to send encrypted messages and receive a reply that you can decrypt. However, you can't yet spontaneously receive messages.

There is nothing more deceptive,
than an obvious fact.