TP 7 : HttpD

Consignes de rendu

À la fin de ce TP, vous devrez rendre un dépôt Git respectant l'architecture suivante :

```
csharp-tp07-prenom.nom/
|-- .gitignore
|-- README
|-- HttpD/
    |-- HttpD.sln
    |-- HttpD/
        |-- Bonus.cs
        |-- Encryption.cs
        |-- HttpClient.cs
        |-- MiniHttps.cs
        |-- Program.cs
        |-- curl.cmd
        |-- req01
        |-- req02
        |-- req03
        |-- Tout sauf bin/ et obj/
```

N'oubliez pas de vérifier les points suivants avant de rendre :

— Remplacez `prenom.nom` par votre propre login.
— Le fichier `README` est obligatoire.
— Pas de dossiers `bin` ou `obj` dans le projet.
— Respectez scrupuleusement les prototypes demandés.
— Retirez tous les tests de votre code.
— **Le code doit compiler !**

README

Vous devez écrire dans ce fichier tout commentaire sur le TP, votre travail, ou plus généralement vos forces / faiblesses, vous devez lister et expliquer tous les boni que vous aurez implémentés. Un `README` vide sera considéré comme une archive invalide (malus).

1 Introduction

1.1 Objectifs

Le but de ce TP est de vous faire comprendre et manipuler des requêtes HTTP. Vous aurez aussi à manipuler un nouvel algorithme de chiffrement [1]. Nous utiliserons dans ce TP de **l'AES**. Une fois ces notions maîtrisées vous serez capable de comprendre et d'implémenter une version fortement simplifiée de l'HTTPS.

2 Cours

2.1 l'HTTP

2.1.1 Un peu de contexte

HTTP est un protocole de communication client-serveur. C'est un protocole qui se situe à la couche applicative du modèle OSI [2]. HTTP signifie *Hyper Text Transfer Protocol*. La version que nous utiliserons dans ce TP est la version HTTP 1.1 qui est décrite de manière exhaustive par la RFC 2616 [3].

2.1.2 Une simple requête HTTP

Durant le TP nous nous contenterons de gérer la partie client d'un échange HTTP. Nous allons décortiquer une requête HTTP simple.

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

Ce que vous avez ci-dessus est une requête HTTP envoyée via la commande **curl** [4]. Nous allons décortiquer la requête ligne par ligne

ligne 1 : **GET** est la méthode HTTP utilisé [5]. Dans notre cas GET signifie que nous demandons à récupérer une ressource sur le serveur. Le / représente le chemin d'accès à la ressource demandée. **HTTP/1.1** représente la version HTTP utilisée.

ligne 2 : **Host** est un paramètre obligatoire qui indique l'adresse du serveur.

ligne 3 : **Accept** est un paramètre qui précise le type de contenu accepté, ici on attend du code html.

ligne 4 : **User-agent** Ce champ contient "l'identité" de l'envoyeur (quel navigateur par exemple)

La requête HTTP que nous avons envoyée contient les 4 éléments obligatoires d'une requête valide [6](Méthode, Chemin, Version, Host). Cependant vous pouvez rajouter bien d'autres headers [7].

Dans une requête HTTP vous pouvez aussi envoyer des données, via la méthode **POST** par exemple. Afin d'envoyer des données, il suffit de laisser une ligne vide entre tous vos headers et les données.

1. En souvenir de l'AFIT
2. `https://en.wikipedia.org/wiki/OSI_model`
3. `https://datatracker.ietf.org/doc/html/rfc2616/`
4. Vous avez accès à la documentation de la commande curl en tapant *man curl*
5. `https://datatracker.ietf.org/doc/html/rfc2616/#section-5.1.1`
6. Pour la version HTTP V1.1
7. `https://en.wikipedia.org/wiki/List_of_HTTP_header_fields`

```
1 POST / HTTP/1.1
2 Host: all-students-of-epita.com
3 Content-Type: text/html
4 Content-Length: 20
5
6 <h1>Votai Test.</h1>
```

Dans cette exemple on envoie le code HTML "<h1>Votai Test.</h1>" au serveur. On remarque la ligne **Content-Length** qui précise la taille des données envoyées en octets.

2.1.3 Une réponse HTTP

```
1 HTTP/1.1 200 OK
```

Une réponse est formée de la même manière qu'une requête. Seuls les headers sont différents. La principale différence est qu'à la place de la **méthode** il y a le **code réponse**[8].

le corps de la réponse contient lui les données demandées par la requête, comme le code HTML d'une page internet ou une image.

2.1.4 Paramètres URL

FIGURE 1 – Une URL avec deux paramètres

Il est également possible de passer des paramètres via la méthode GET. Ils prennent la forme *var* = *value* et sont placés après le caractère **?**. Dans cet exemple, on passe les paramètres **q** et **ia** avec les valeurs respectives "votaitest" et "web".

2.1.5 encodage URL

Certains caractères ASCII doivent ne peuvent être directement écrit dans des paramètres HTTP car ils sont déjà utilisés dans le format URL. C'est par exemple le cas du slash (/). Pour pallier ce problème, il existe un encodage URL qui permet tout de même de passer des caractères "interdits". Il suffit pour cela de remplacer le caractère par **'%xx'**, où 'xx' est la valeur ASCII du caractère dans la base hexadécimale.

1. La valeur ASCII hexadécimale de / est 2F
2. On rajoute un % devant la valeur
3. / => **%2F**

FIGURE 2 – Une URL avec un paramètre qui contient un **;** dans le paramètre q

8. le fameux 404 est un code de réponse HTTP

Important

Pour ce TP vous aurez sans doute besoin d'encoder vos paramètres. Pour ce faire, vous pouvez utiliser ce site `https://www.urlencoder.io/`

2.2 Advanced Encryption Standard [9]

L'*Advanced Encryption Standard* est un algorithme de chiffrement symétrique se basant entre autres sur une clé privée de longueur prédéfinie avant le début de la communication entre les pairs. Il est, à ce jour considéré comme l'algorithme de chiffrement le plus sûr. Il est dérivé de l'algorithme de *Rijndael*. Les clés AES sont représentées en **Base64** [10]. Chaque message que vous voulez chiffrer est accompagné d'un vecteur d'initialisation (IV). Cet IV est utilisé pour ajouter un aspect aléatoire dans les découpages et chiffrements des blocs. Il doit être généré de manière aléatoire/semi-aléatoire.

L'algorithme prend en entrée une matrice 4x4 :

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

À cette matrice, s'ajoute une clé de 128, 192 ou 256 bits. Il s'agira ensuite d'exécuter, pendant un nombre d'étapes définies par la taille de la clé, un algorithme donné. À chaque itération, une nouvelle clé est dérivée de la clé initiale pour accomplir les tâches [11].

L'algorithme se déroule de cette manière :

1. À la première itération, on applique à chaque élément de la matrice (qui est en pratique un octet) un `xor` avec un autre octet de la clé associée à l'itération.
2. Si l'itération actuelle n'est ni la première, ni la dernière :
   (a) Une étape de substitution est appliquée à la matrice, elle consiste simplement en l'application d'une transformation à chaque élément, on aura alors une matrices B avec $b_i = S(a_i)$ où S représente la transformation en question.
   (b) Une étape de rotation gauche est ensuite appliquée séquentiellement à chaque ligne. Il s'agit de faire une rotation gauche de $n-1$ élément à la ligne n. Ainsi, la première ligne ne subira aucune rotation. La dernière ligne (la quatrième donc), quant à elle, subira une rotation de 3 éléments.
   (c) Une multiplication matricielle [12] est ensuite appliquée de manière à la matrice B, elle est simplement multipliée par

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

9. En français on dit chiffrer pas crypter
10. `https://en.wikipedia.org/wiki/Base64`
11. https ://en.wikipedia.org/wiki/AES_key_schedule
12. Moyennant l'idée de rester sur des octets, cette multiplication sera faite modulo 256

(d) On effectue le fameux `xor` sur la matrice tel que décrit dans la première étape.

3. Pour la dernière itération, on exécutera seulement les étapes *a*, *b* et *d* du cas précédent.

À noter que pour le déchiffrement, il suffira de faire les opérations inverses (ordre inverse, inversion de M, ...).

2.2.1 Opérateur binaire

Pour l'AES vous aurez besoin de maîtriser les opérateurs binaires de base.

Nous allons commencer par voir les opérateurs de décalage de bits » et «. L'opérateur » décale tous les bits d'une valeur vers la droite et vers la gauche pour «. Les bits qui sont insérés sont à 0 et les bits qui dépassent sont perdus. Vous pouvez vous fier aux exemples suivants.

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

Les opérateurs de décalage de bits effectuent un décalage de tous les bits dans le sens précisé par la valeur directement à droite.

Il existe deux autres opérateurs binaires très utilisés : le ET binaire **'&'** et le OU binaire **'|'**.

```
ushort vim = 65; // 0000 0000 0100 0001
vim = vim & 64;
// 0000 0000 0100 0001 & 0000 0000 0100 0000
// vim = 0000 0000 0100 0000
vim = vim | 65;
// 0000 0000 0100 0000 | 0000 0000 0100 0001
// vim = 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
*/
```

Les opérateurs & et | effectuent respectivement un ET binaire ou un OU binaire entre les bits de deux valeurs.

Pour continuer avec les opérateurs logiques vous allez (re)-découvrir le XOR.

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

Conseil

Vous pouvez lire la documentation MSDN `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#logical-and-operator-`

3 HttpClient

3.1 Envie de curler

Le but de l'exercice est d'utiliser la commande **curl** pour récupérer une ressource à l'adresse suivante `https://photos.cri.epita.fr/vlad.argatu`. Le contenu récupéré est une image.

Attention

Vous devrez expliquer dans votre README comment vous avez trouvé la commande curl. De plus, il vous faudra rendre dans un fichier nommé *curl.cmd*, la commande que vous avez utilisée.

Conseil

La ressource que vous allez récupérer est une image vous pouvez rajouter "> ACDC.jpg" à la fin de votre commande pour enregistrer le résultat de la requête dans **ACDC.jpg**. Vous pouvez ensuite utiliser la commande 'feh' [a] pour l'afficher.

a. Utilisez **man feh** pour plus d'information sur l'utilisation de la commande

3.2 Constructeur

Passons enfin aux choses sérieuses. vous allez ici devoir remplir le constructeur de la class HttpClientStream.

A noter que tout reposera sur la variable _httpClient que vous allez devoir assigner et initialiser avec une Uri de base. N'oubliez pas de clear tous les headers par défaut de la variable.

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

3.3 SendMessage

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

Votre rôle ici est d'envoyer une requête avec le méthode *Method* en incluant le message dans le body. Vous devez renvoyer un *StreamReader*

Conseil

Aidez vous de la classe **HttpRequestMessage** pour la requête et la fonction **ReadAsStream** pour renvoyer la réponse à la requête en tant que StreamReader.

3.4 ResponseFromStream

```
1 public string ResposeFromStream(StreamReader stream);
```

L'objectif ici est de lire le contenu du stream et renvoyer la string correspondante.

Attention

N'oubliez pas de fermer le stream avant de sortir de la fonction.

4 Encryption

Avant de pouvoir envoyer nos messages, il est important de sécuriser leur contenu pour éviter que n'importe qui puisse y accéder. Nous allons donc développer les fonctions permettant de chiffrer et déchiffrer des messages à l'aide d'une clé privée qui sera utilisée avec l'algorithme AES.

Comme vu dans le cours, l'AES se base sur des opérations XOR sur la clé donnée en effectuant des rotations successives. Vous retrouverez dans le code fourni, toutes les byte-array utilisées pour effectuer les rotations de la clé.

Bien que cette partie puisse sembler impressionnante, vous serez guidés tout du long afin d'implémenter au mieux AES. Notons également que nous n'aborderons que le chiffrement AES-128 bits (avec 11 rotations de la clé).

Conseil

Tout au long de cette partie, nous allons diviser les tâches à faire en petites fonctions. Vous ne comprendrez pas forcément ce que vous faites sans avoir lu la documentation.
Il est donc vivement recommandé que vous lisiez au moins une fois le cours ainsi que la page Wikipédia consacrée à ce sujet :
`https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`

4.1 SubBytes

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

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

Vous devez ici remplacer chaque élément de a par la valeur correspondante à cet élément dans les byte-arrays *LookupSbox* et *LookupSboxInv* pour les fonctions SubBytes et SubBytesInv respectivement.

4.2 KeySchedule

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

Nous allons ici implémenter le coeur de l'algorithme de Key Schedule[13] Vous devez ici faire une rotation gauche des 4 bytes contenus dans a, puis appliquer SubBytes et faire un XOR entre le byte le plus à gauche et la table LookupRcon à l'index i.

4.3 Xor

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

Cette fonction sera utilisée pour les opérations Xor entre 2 tableaux. Vous devez donc effectuer pour chacun des bytes de a un XOR avec le byte de b en prenant en compte l'offset dans l'index de b.

13. `https://en.wikipedia.org/wiki/AES_key_schedule`

4.4 Expand

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

A l'aide des deux dernières fonctions, vous devez implémenter tout l'algorithme de Key Schedule d'AES [14]. L'objectif est d'étendre la clé AES de 128 bits (au format byte array) en appliquant les 11 rotations indiquées.

La fonction doit renvoyer une byte array de taille 176 (11 * 16 bytes) contenant les 11 rotations de la clé à la suite les unes des autres.

Conseil

Dans l'algorithme que vous trouverez sur Wikipédia :

- N correspond à 4 dans notre cas (les clés 128 bits font 4 mots de 4 bytes).
- K_i correspond aux 16 bytes de la clé passée en paramètre.
- R correspond à 11 dans notre cas (11 rotations pour les clés 128 bits).
- W_i correspond à la liste temporaire de 11 * 16 bytes mentionnée précédemment.

4.5 XorCipherWithRoundKey

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

Vous devez ici appliquer une opération XOR sur chaque byte du cipher avec le byte correspondant de la clé à la *round*-ème rotation.

4.6 ShiftRows

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

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

Vous devez ici remplacer chaque byte par le byte à l'index donné par l'array **ShiftRowsTable**. (Faites la même chose avec l'array **ShiftRowsTableInv** pour la deuxième fonction.

4.7 MixCol

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

Vous devez ici mixer les colonnes (les 4 bytes composant un mot de 32 bits en partant de l'offset). Vous devez donc pour chaque byte du mot lui appliquer successivement des XOR entre la table **LookupG2** sur le byte que l'on modifie, la table **LookupG3** sur le byte suivant et les 2 bytes suivants.

Conseil

Si vous êtes déjà sur le dernier byte du mot, le "prochain" byte indiqué au dessus correspond au premier byte du mot.

14. https://en.wikipedia.org/wiki/AES_key_schedule#The_key_schedule

4.8 MixColInv

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

Le principe est le même que la fonction précédente mais vous devez cette fois appliquer le XOR entre les tables successives **LookupG14** sur le byte que l'on modifie, puis sur les bytes en sens inverse pour les tables **LookupG9**, **LookupG13** et **LookupG11**

4.9 MixCols

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

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

Vous devez ici appliquer la fonction précédente pour chaque mot de 4 byte contenus dans le cipher de 16 bytes. Réglez bien l'offset à chaque appel de **MixCol**. Reproduisez également cette fonction avec **MixColInv**

4.10 Encrypt

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

Vous avez presque fini ! il ne vous reste plus qu'à mettre bout à bout tout ce que vous avez fait jusqu'ici pour chiffrer votre message avec la key indiquée.

Si vous avez lu comme il faut la documentation, vous saurez facilement comment mettre bout à bout chaque fonction dans le bon ordre [15] !

Vous devez donc chiffrer un message de 16 bytes à l'aide d'une key de 16 bytes et renvoyer le message chiffré (sans modifier le message d'origine). Le message chiffré devrait aussi faire 16 bytes.

4.11 Decrypt

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

Maintenant que vous savez chiffrer, il convient de pouvoir déchiffrer vos propres messages. Et ça tombe bien, avec AES, déchiffrer un message est tout aussi facile que le chiffrer. Vous avez remarqué que depuis le début, on a recopié plusieurs fonctions en modifiant uniquement les tableaux pour mixer et shifter. Ces tableaux permettent d'inverser le chiffrement AES, à partir du moment où vous disposez de la même clé que celle qui a permis de chiffrer le message d'origine.

Il vous suffit donc simplement d'inverser l'ordre d'exécution de la fonction Encrypt en utilisant les fonctions Inv à la place des originales.

15. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#High-level_description_of_the_algorithm

4.12 ToBytes

Le but de cette fonction est de transformer une string de caractères hexadécimaux en un tableau de byte. La string comprend forcément un nombre pair de caractères (C'est une suite de bytes valides au format hexadécimal).

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

Exemple avec la string "BABE"

```
1  [0xBA, 0xBE]
```

4.13 Pretty

Vous devez implémenter la fonction Pretty qui affiche un tableau de byte.

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

Exemple avec label = "Test." et a = [0x01, 0x05, 0x1F]

```
1  Test.:
2  01-05-1F
```

5 MiniHttps

Vous êtes presque venus à bout de ce TP. Maintenant, vous allez devoir mettre à profit vos fonctions de chiffrement et d'envoi de message via HTTP afin d'envoyer des messages de manière sécurisée et de recevoir une réponse qui l'est tout autant.

5.1 Constructeur

Vous devez initialiser les attributs _client et _key de la classe MiniHttps.

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

5.2 EncryptFullMessage

Vous savez chiffrer des messages de 128 bits. Mais comment faire si votre message est plus long ? Vous êtes obligé de chiffrer chaque partie de 128 bits du message séparément puis de les concaténer pour former le message chiffré en entier.

```
1  public byte[] EncryptFullMessage(string message);
```

5.3 DecryptFullCipher

```
1  public string DecryptFullCipher(byte[] cipher);
```

Faites maintenant la même chose mais en déchiffrant un cipher pour renvoyer la string du message d'origine.

5.4 EncryptAndSend

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

Vous devez pour cette dernière fonction, chiffrer un message, l'envoyer au serveur distant, puis renvoyer la réponse déchiffrée.

6 Pawn me daddy !

6.1 Objectifs

Le but de cette section est d'utiliser votre client HTTP pour résoudre trois petits CTF. L'application que vous allez "attaquer" est une application développé par OWASP [16]. Si le la suite de ce TP vous passionne [17], vous trouverez un florilège de ressources sur le site web de l'organisation pour aller plus loin.

Important

On vous demande de mettre dans des fichiers "reqXX" les URLs qui vous on permis de réussir l'exercice. Exemple si l'URL **https ://duckduckgo.com/ ?q=votaitest&ia=web** [a] vous permet de réussir un exercice c'est ce dernier qu'il faut A LA PREMIÈRE LIGNE du fichier req associé.

a. Au hasard

6.2 Injection de fichier

Le but de cet exercice est d'écrire une requête HTTP permettant de lire le contenu du fichier **/etc/-passwd** sur le serveur. Pour cela il vous faut envoyer une requête avec la méthode **GET** à l'adresse suivante : **`http://httpdtest.ml/vulnerabilities/fi/?page=include.php`** [18]. Vous copierez votre URL dans le fichier **req01** ainsi que votre démarche de recherche.

6.3 Injection de commande

Cette fois ci il faudra exécuter une commande sur le serveur via une requête avec la méthode **GET**. L'adresse est la suivante **`http://httpdtest.ml/vulnerabilities/exec?ip=0.0.0.0&Submit=Submit`** [19]. Il vous faudra dans le fichier **req02** votre requête ainsi que le contenu du fichier flag_2 et votre démarche pour réussir cet exercice.

6.4 Injection SQL

Pour le dernier exercice il faudra récupérer le mot de passe du compte **Test.**. Pour cela il faudra effectuer une injection SQL [20] via le paramètre **id**. Voici l'URL que vous devez utiliser `http://httpdtest.ml/vulnerabilities/sqli/?id=2&Submit=Submit`.

16. `https://owasp.org/`
17. Et il vous passionnera !
18. `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`
19. `https://owasp.org/www-community/attacks/Command_Injection`
20. `https://owasp.org/www-community/attacks/SQL_Injection`

Important

Les mots de passe sont stockés sous forme de Hash. Néanmoins on attend de vous de retrouvez le mot de passe en clair. La méthode hachage utilisée est une vieille méthode très facilement cassable (Indice : 5)

Tout comme les deux autres exercices vous donnerez l'URL dans le fichier **req03** sans oublier les explications !

7 Bonus

Les clés AES sont utilisées au format binaire pour faire les opérations de chiffrement et déchiffrement. Une représentation simple du binaire est la Base 64.[21] En effet, elle permet d'encoder un nombre binaire de 24 bits (3 bytes) en 4 caractères contenus dans l'alphabet de la base 64.

On retrouve ainsi les lettres majuscules, minuscules, les chiffres ainsi que les 2 caractères + et / dans cet ordre. (A correspondant à 0 et / correspondant à 63. Enfin le caractère = qui permet de compléter la string en base 64 si la conversion n'aboutit pas à une longueur multiple de 4. On appelle cette opération le padding.[22]

7.1 StringToBase64

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

Vous devez ici, encoder une string dans son équivalent en base 64.

Conseil

Vous devez utiliser les opérateurs binaires de shift pour convertir les caractères de la string en base 64.

7.2 Base64ToString

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

Vous devez ici, décoder une string en base 64 vers sa string originelle.

Conseil

Vous devez lancer une exception si la string n'est pas dans un format base 64 valide.

8 Erratum

Nous n'avons pas pu voir la partie serveur dans ce TP, ce qui fait que vous êtes seulement capable d'envoyer des messages chiffrés et de recevoir une réponse que vous pouvez déchiffrer. Mais vous ne pouvez pas encore recevoir spontanément des messages.

21. https://en.wikipedia.org/wiki/Base64
22. https://en.wikipedia.org/wiki/Base64#Output_padding

There is nothing more deceptive,
than an obvious fact.