Faculté des Sciences de Rabat, master IPS 2021-2022

Sécurité informatique

# TP 1 : Chiffrements par blocs - Modes opératoires et bourrage.

Objectifs du TP

-utiliser openssl pour chiffrer/déchiffrer,

-étudier le remplissage (padding) du dernier bloc,

-les\_cryptogrammes.rar archive contenant les cryptogrammes à décrypter.

- utiliser xor.py programme python pour faire l’addition binaire

Outils utilisés

-openSSL, boîte à outils cryptographiques,

-Python

-HexEdit, éditeur hexadécimal

1. Présentation de **openSSL**
   1. Protocole SSL

Le protocole SSL (Secure Socket Layer) a été développé par la société Netscape Communications Corporation pour permettre aux applications client/serveur de communiquer de façon sécurisée. TLS (Transport Layer Security) est une évolution de SSL réalisée par l'IETF.

La version 3 de SSL est utilisée par les navigateurs tels que Mozilla et Microsoft Internet Explorer.

SSL est un protocole qui s'intercale entre TCP/IP et les applications qui s'appuient sur TCP. Une session SSL se déroule en deux temps :

* + 1. une phase de poignée de mains (handshake) durant laquelle le client et le serveur s'identifient, conviennent du système de chiffrement et d'une clé qu'ils utiliseront par la suite ;
    2. une phase de communication proprement dite durant laquelle les données échangées sont compressées, chiffrées et signées.

L'identification durant la poignée de mains est assurée à l'aide de certi cats X509.

* 1. openSSL

openSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS qui offre :

* + 1. Une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS.
    2. Une commande en ligne (openssl) permettant

-la création de clés RSA, DSA (signature) ;

-la création de certificats X509 ;

-le calcul d'empreintes (MD5, SHA, RIPEMD160, ) ;

-le chiffrement et déchiffrement (DES, IDEA, RC2, RC4, Blowfish, ) ;

-la réalisation de tests de clients et serveurs SSL/TLS ;

-la signature et le chiffrement de courriers (S/MIME).

Pour connaître toutes les fonctionnalités de openSSL : openssl help. La syntaxe générale de la commande openssl est

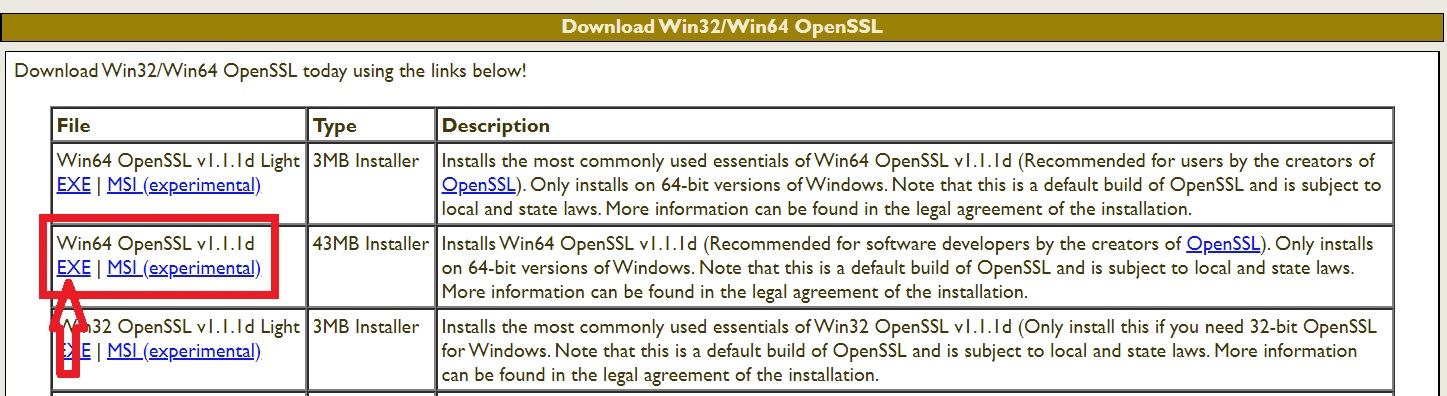
* openssl <commande> <o p ti o n s >

(le **>** étant le prompt du shell)

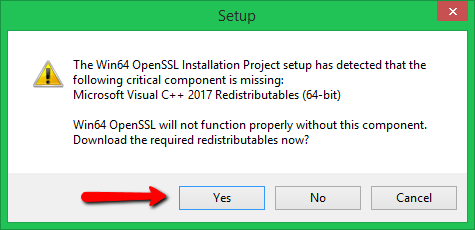
Dans le texte qui suit, les commandes invoquant openssl supposent que cette commande est dans votre variable shell PATH.

* 1. Installation de OpenSSL sous Windows

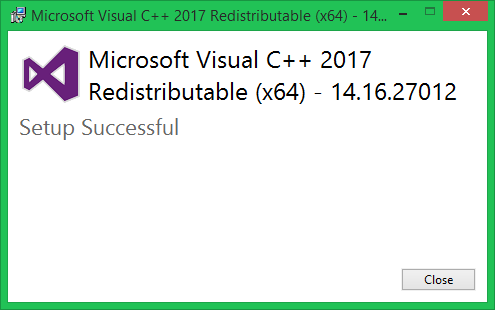
**1.** Téléchargez le dernier programme d'installation Windows OpenSSL à partir de la page de téléchargement officielle. Le lien officiel de la page de téléchargement est indiqué ci-dessous: <https://slproweb.com/products/Win32OpenSSL.html>



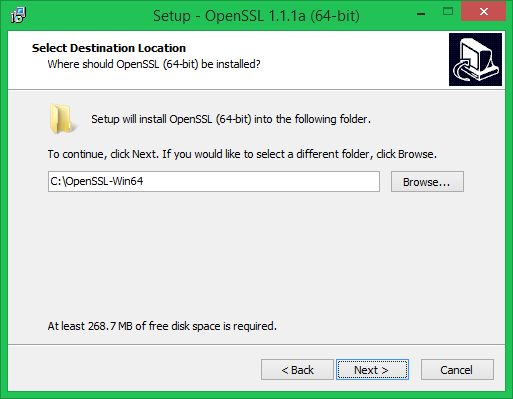
**2.** Maintenant, lancez le programme d’installation OpenSSL sur votre système. OpenSSL nécessitait que Microsoft Visual C ++ soit installé sur votre système. Si Microsoft Visual C ++ n'est pas installé sur votre système, le programme d'installation vous affichera un message, comme suit:



Cliquez sur **Oui** pour télécharger et installer le package Microsoft Visual C ++ requis sur votre système.

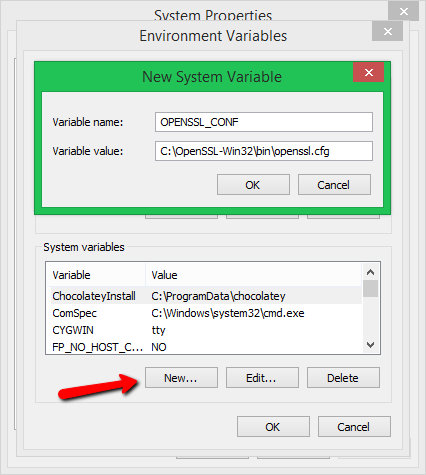


Ensuite, exécutez à nouveau le programme d’installation OpenSSL et suivez l’assistant.

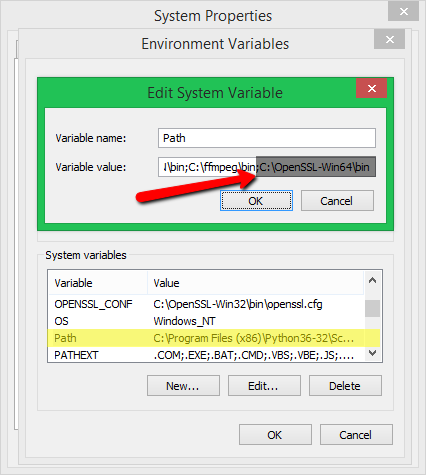


**3.** Maintenant, configurez les variables d’environnement pour qu’elles fonctionnent correctement avec OpenSSL sur votre système. Vous devez définir les variables d'environnement OPENSSL\_CONF et Path.

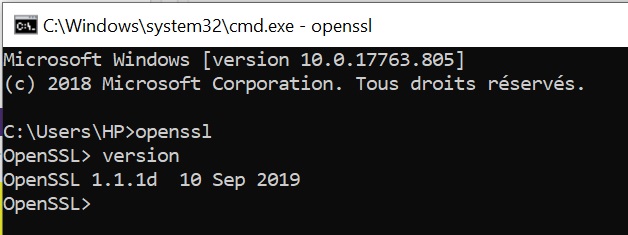
Définir la variable OPENSSL\_CONF:



Définir la variable Path:



**4.** Ouvrez une invite de commande et tapez openssl pour obtenir l’invite OpenSSL. Ensuite, exécutez la commande version sur OpenSSL afin d’afficher la version OpenSSL installée.



1. Chiffrement symétrique avec **openSSL**

C'est la commande enc qui permet de chiffrer/déchiffrer avec openssl :

* openssl enc <opt ions >

Parmi les options, on doit indiquer le système de chiffrement à choisir dans la liste :

|  |  |  |  |
| --- | --- | --- | --- |
| **Option** | **Signification** | **Option** | **Signification** |
| aes-128-cbc | AES 128 bis in CBC mode | bf-cbc | Blowfish in CBC mode |
| aes-128-ecb | AES 128 bis in ECB mode | Bf | Alias for bf-cbc |
| aes-192-cbc | AES 192 bis in CBC mode | bf-cfb | Blowfish in CFB mode |
| aes-192-ecb | AES 192 bis in ECB mode | bf-ecb | Blowfish in ECB mode |
| aes-256-cbc | AES 256 bis in CBC mode | bf-ofb | Blowfish in OFB mode |
| aes-256-ecb | AES 256 bis in ECB mode | des-cbc | DES in CBC mode |
| cast-cbc | CAST in CBC mode | des | Alias for des-cbc |
| Cast | Alias for cast-cbc | des-cfb | DES in CBC mode |
| cast5-cbc | CAST5 in CBC mode | des-ofb | DES in OFB mode |
| cast5-cfb | CAST5 in CFB mode | des-ecb | DES in ECB mode |
| cast5-ecb | CAST5 in ECB mode | des-ede-cbc | Two key triple DES EDE in CBC mode |
| cast5-ofb | CAST5 in OFB mode | des-ede | Alias for des-ede |
| idea-cbc | IDEA algorithm in CBC mode | des-ede-cfb | Two key triple DES EDE in CFB mode |
| Idea | same as idea-cbc | des-ede-ofb | Two key triple DES EDE in OFB mode |
| idea-cfb | IDEA in CFB mode | des-ede3-cbc | Three key triple DES EDE in CBC mode |
| idea-ecb | IDEA in ECB mode | des-ede3 | Alias for des-ede3-cbc |
| idea-ofb | IDEA in OFB mode | des3 | Alias for des-ede3-cbc |
| rc2-cbc | 128 bit RC2 in CBC mode | des-ede3-cfb | Three key triple DES EDE CFB mode |
| rc2 | Alias for rc2-cbc | des-ede3-ofb | Three key triple DES EDE in OFB mode |
| rc2-cfb | 128 bit RC2 in CBC mode | desx | DESX algorithm |
| rc2-ecb | 128 bit RC2 in CBC mode | rc5-cbc | RC5 cipher in CBC mode |
| rc2-ofb | 128 bit RC2 in CBC mode | rc5 | Alias for rc5-cbc |
| rc2-64-cbc | 64 bit RC2 in CBC mode | rc5-cfb | RC5 cipher in CBC mode |
| rc2-40-cbc | 40 bit RC2 in CBC mode | rc5-ecb | RC5 cipher in CBC mode |
| rc4 | 128 bit RC4 | rc5-ofb | RC5 cipher in CBC mode |
| rc4-64 | 64 bit RC4 | **base64** | **Base 64** |
| rc4-40 | 40 bit RC4 |  |  |

Remarque : base64 n'est pas un système de chiffrement, mais un codage des fichiers binaires avec 64 caractères ASCII. Ce codage est utilisé en particulier pour la transmission de fichiers binaires par courrier électronique.

* 1. Chiffrement avec mot de passe

Pour chiffrer le fichier **myFile** avec le système Blowfish en mode CBC, avec une clé générée par mot de passe, le chiffré étant stocké dans le fichier **myFile.chiffre**, on utilise la commande :

* openssl enc –bf-cbc −i n myFile −out myFile.chiffre Pour déchiffrer le même message, on utilise la commande :
* openssl enc −bf−cbc −d −i n myFile.chiffre −out myFile.dechiffre

Vérification linux

* diff myFile myFile.dechiffre

Vérification windows

* fc myFile myFile.dechiffre

Exercice 1.

Question 1. Chiffrez le fichier de votre choix avec le système de votre choix dans le mode de votre choix, puis déchiffrez-le.

Question 2. Comparez les tailles des fichiers clairs et chiffrés. Donnez une explication sur la différence de ces tailles.

Question 3. Tentez de déchiffrer un cryptogramme en utilisant un mauvais mot de passe. Comment réagit openSSL ?

Exercice 2. Le fichier cryptogram10 a été chiffré avec le système AES en mode CBC, la clé de 128 bits ayant été obtenue par mot de passe.

Question 1. Le mot de passe codé en base 64 est Y3J5cHRv . À l'aide de la commande openssl

appropriée, décodez le mot de passe.

Question 2. Déchiffrez ensuite le cryptogram10.

* 1. Chiffrement avec clé explicite

Pour chiffrer le fichier **myFile** avec une clé explicite, il faut utiliser les options -K et -iv

-K (K majuscule) suivi de la clé exprimée en hexadécimal ;

-iv (iv en minuscules) suivi du vecteur d'initialisation exprimé en hexadécimal [1](#_bookmark0) .

L'exemple qui suit montre la commande pour chiffrer **myFile** avec Blowfish en mode CBC avec un vecteur d'initialisation de 64 bits exprimé par 16 chiffres hexa, et une clé de 128 bits exprimée par 32 chiffres hexa.

−K 0123456789ABCDEF0123456789ABCDEF

−i v 0123456789ABCDEF \

* openssl enc −bf−cbc −i n myFile −out myFile.chiffre \

Exercice 3.

Question 1. Chiffrez le fichier clair11 avec le système Blowfish en mode OFB, en utilisant le vecteur d'initialisation (option -iv) et la clé (option -K) de votre choix.

Question 2. Chiffrez le fichier clair correspondant au cryptogram10 avec le même système, la même clé et le même vecteur d'initialisation que dans la question qui précède.

Question 3. Utilisez le programme écrit en python xor.py pour faire un "xor" des deux fichiers clairs. Utilisez le même programme pour faire un "xor" des deux chiffrés. Faîtes un diff (fc sous windows) entre les deux fichiers obtenus. Que constatez-vous ?. Le résultat aurait-il été le même si on avait utilisé un système de chiffrement autre que Blowfish ? Explication.

Usage : py xor.py <fichier1> <fichier2> <fichier3>

<fichier1> et <fichier2>: fichiers à xorer

<fichier1> doit être de taille <= à <fichier2>

<fichier3> : fichier résultat

# Le bourrage (padding)

Lorsque la taille de la donnée n'est pas un multiple de la taille d'un bloc, il est nécessaire de compléter le dernier bloc avec quelques bits complémentaires : c'est le bourrage (ou padding ).

Une façon de remplir, définie dans le RFC2040, consiste à compléter le dernier bloc par autant d'octets que nécessaire, chaque octet ayant pour valeur le nombre d'octets ajoutés.

Par exemple, s'il manque trois octets au message *m* = *o*1*o*2*o*3*o*4*o*5 pour obtenir un bloc de huit octets, on ajoute trois octets égaux à 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *o*1 | *o*2 | *o*3 | *o*4 | *o*5 | 03 | 03 | 03 |

Table 1 Complétion d'un bloc avec trois octets

S'il se trouve que la taille de la donnée à chiffrer est un multiple de la taille d'un bloc, on ajoute un bloc entier dont chaque octet a pour valeur la taille en octet d'un bloc.

Par exemple, pour des blocs de huit octets, on ajoute le bloc

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 08 | 08 | 08 | 08 | 08 | 08 | 08 | 08 |

Table 2 Complétion par un bloc entier de huit octets

Exercice 4. En chiffrant des fichiers de taille différentes, avec le système Blowfish (blocs de 64 bits) avec une clé et un vecteur d'initialisation fournis dans la ligne de commande, observez la taille des chiffrés obtenus.

Lorsqu'on déchiffre un cryptogramme avec openssl, les octets ajoutés lors du bourrage sont supprimés. Avec l'option -nopad utilisée dans la ligne de commande de déchiffrement, ces octets ne sont pas supprimés.

Exercice 5. En chiffrant des fichiers de taille différentes, avec le système Blowfish (blocs de 64 bits) avec une clé et un vecteur d'initialisation fournis dans la ligne de commande, et en les déchiffrant avec l'option -nopad, observez les octets de bourrage. (Attention, les octets de bourrage ne correspondent pas à des caractères imprimables. Il vous faudra utiliser hexedit ou xxd pour visualiser le contenu du fichier déchiffré en hexadécimal.) C'est cette façon de bourrer le dernier bloc qui permet de contrôler que le déchiffrement s'est bien déroulé. En effet, si après déchiffrement le dernier bloc clair ne se termine pas par *n* octets identiques valant *n*, alors on peut en déduire au moins l'une des causes suivantes :

1. la clé utilisée lors du déchiffrement est incorrecte,
2. le dernier bloc chiffré est erroné,
3. en mode CBC, l'avant dernier bloc est erroné.

Exercice 6. Chiffrez avec le système Blowfish en mode CBC et en précisant clé et vecteur d'initialisation dans la ligne de commande, un fichier de façon à obtenir un cryptogramme de deux blocs.

Puis, vérifiez chacune des causes d'erreur mentionnées ci-dessus.

Vérifiez aussi qu'avec l'option -nopad, openssl n'effectue pas le contrôle de bourrage.