Réseaux pour ingénieurs
GLO-2000

TP3 : Programmation socket

Professeur responsable :
Ronald Beaubrun
Ronald.Beaubrun@ift.ulaval.ca

Responsables des travaux pratiques :
Jacob Ouellet-Boudreault et Vincent Primpied
GLO2000.laboratoires@protonmail.com

Université Laval
Faculté des sciences et de génie
Automne 2023

1 Introduction

Ce troisième TP a pour but de vous familiariser avec la programmation socket, ainsi que la transmission de données via le réseau. Vous programmerez en Python un protocole cryptographique d'échange de clés via des sockets dans le but d'établir une communication sécurisée.

2 Modalités de remise

Les modalités de remise doivent être acceptées et lues directement sur le site web local de la machine virtuelle.

Assurez-vous de respecter attentivement le protocole à utiliser pour échanger des messages. **L'envoi de messages dans un format différent fera échouer les tests et impactera grandement votre note. De même pour les modules à utiliser.**

ATTENTION : Lorsque vous gérez les erreurs avec des exceptions, vous perderez 50% des points reliés à la fonction si vous utilisez un "catch all" comme le suivant :

```
try:
    some_function()
except glosocket.GLOSocketError:      # Vous pouvez faire cela
    handle_error1()
except:                               # Il ne faut __PAS__ faire cela
    handle_error2()
```

3 Programmation de l'échange de clés (12 points)

3.1 Mise en contexte

Supposons que deux individus, Alice et Bob, souhaitent échanger des informations confidentielles via un réseau, avec le protocole TCP/IP. Puisque ce protocole n'empêche pas un routeur malveillant de lire le contenu des paquets échangés entre les deux, Alice et Bob doivent chiffrer leur communication, c'est-à-dire transformer leurs messages de façon à ce qu'ils soient compréhensibles seulement par eux.

Si Alice et Bob partagent au préalable un secret (ou clé) k, alors il leur est facile de chiffrer leur communication. En utilisant une fonction de chiffrement E_k associée à une fonction de

déchiffrement D_k, Alice peut transmettre un message x à Bob en calculant le texte chiffré $c = E_k(x)$ et en envoyant c via le réseau. Bob peut alors retrouver le message en calculant $x = D_k(c)$. E_k et D_k ne peuvent être calculées qu'en connaissant la clé k, ce qui garantit que personne d'autre ne peut retrouver le message x à partir de c.

Supposons maintenant qu'Alice et Bob ne se sont jamais rencontrés auparavant. Comment peuvent-ils connaître une clé k commune sans que personne ne puisse l'intercepter ?

L'échange de clés de Diffie-Hellman répond à cette problématique. Dans ce protocole, Alice et Bob s'entendent d'abord sur les valeurs suivantes et les partagent publiquement :

— Un nombre premier aléatoire a qui servira de 'modulo'.

— Un entier aléatoire b, $0 \leq b < a$, qui servira de 'base'.

Ensuite, Alice et Bob génèrent chacun une **clé privée**, respectivement p et q, où $0 \leq p, q < a$. Alice calcule alors sa **clé publique** $P = b^p \mod a$, tandis que Bob calcule $Q = b^q \mod a$, puis ils se partagent ces valeurs via le réseau.

Alice connaît alors p et Q ; elle peut donc calculer la **clé partagée**

$$k = b^{pq} \mod a = (b^q)^p \mod a = Q^p \mod a$$

De son côté, Bob est capable d'obtenir la même valeur en calculant

$$k = b^{pq} \mod a = (b^p)^q \mod a = P^q \mod a$$

Alice et Bob partagent donc un secret k, un nombre en apparence aléatoire que personne d'autre ne peut calculer. En effet, un espion interceptant b, a, P et Q est incapable de calculer $k = b^{pq} \mod a$, puisque cela demande par exemple de déduire p de $P = b^p \mod a$, ce qui est un problème trop difficile pour un ordinateur.

3.2 Modules fournis

Deux modules vous sont fourni et sont directement importés dans le squelette qui vous est donné.

Le premier module, 'glocrypto', vous apporte les fonctions suivantes :

— `find_prime()` : Retourne un nombre premier sur 256bits. Note : la fonction utilise le test de Fermat, la primalité du nombre n'est donc pas garantie.

— `random_integer(modulus)` : Retourne un entier aléatoire entre 0 (inclus) et 'modulus' (exclus).

— `modular_exponentiation(base, exponent, modulus)` : Calcule efficacement le résultat de l'opération : $base^{exponent} \mod modulus$

Le second module, 'glosocket', vous apporte les fonctions suivantes :

— `send_mesg(dest_soc, message)` : Encode le message et l'envoi à la destination.

— `recv_mesg(source_soc)` : Récupère le message depuis la source et le décode.
— En cas d'erreur ces fonctions lèvent l'exception `GLOSocketError`.

Important : La correction automatique dépend de ces deux modules, vous devez les utiliser pour toutes les communications entre sockets ainsi que toutes les opérations cryptographiques. Vous ne pouvez pas modifier ces deux modules.

3.3 Spécifications du programme

Votre programme doit respecter l'ensemble des spécifications décrites ci-dessous.

Les points suivants sont strictement nécessaires :
— Le mode IPv4 doit être utilisé.
— Le mode TCP doit être utilisé.
— Chaque échange entre le client et le serveur doit être effectué avec le module fourni 'glosocket'.
— Tous les calculs doivent être effectués à l'aide du module 'glocrypto'.

Récupération d'arguments en ligne de commande : (4pts)
— L'option `-g/--destination-port` doit être systématiquement utilisé pour désigner le port.
— L'option `-s/--server` démarre l'application en mode serveur, sinon le mode client est utilisé par défaut.
— L'option `-d/--destination` indique l'adresse IP à laquelle le client doit se connecter.
— Les options `-s` et `-d` ne peuvent pas être utilisées en même temps, cependant au moins une doit être utilisée.

Protocole d'échange : (5pts)
— Le client se connecte au serveur.
— Le serveur génère un modulo puis une base et les transmet au client dans deux messages distincts.
— Chacun calcule sa clé privée et publique.
— Chacun transmet sa clé publique à l'autre.
— Chacun calcule sa clé partagée et l'affiche dans le terminal.
— La clé partagée ne doit pas être transmise sur le réseau.

Gestion d'erreur : (3pts)
— La gestion d'erreur avec les arguments se fait automatiquement à l'aide 'argparse'.
— Si le serveur ou le client rencontre une erreur à la création du socket (port ou adresse invalide), le programme fait appel à la méthode `sys.exit` avec un code différent de 0.
— Si le serveur rencontre une erreur, il ferme sa connexion avec le client et en attend un nouveau.
— Si le client rencontre une erreur, il fait appel à la méthode `sys.exit` avec un code différent de 0.

Le fichier 'TP3_Q1.py' (pour la question 3) vous est fourni, il contient le squelette des fonctions à implémenter. Vous êtes libre d'y ajouter des fonctions auxiliaires si nécessaire pourvu que chaque fonction suive le comportement documenté. Les annotations de type vous in-

diquent quels types de retours sont attendus, cependant, notez que Python ne vous force pas à les suivre, mais que la correction automatique s'attend exactement à ce résultat.

4 Reniflage de l'échange de clés (3 points)

Démarrez Wireshark et reniflez le réseau sur l'interface loopback. Démarrez votre programme Python en mode serveur et écoutez sur votre ***port d'équipe***. Par la suite, dans une autre console, démarrez votre programme Python en mode client et connectez-vous sur ce même port.

Vous devez remettre le fichier contenant la trace des paquets qui sont communiqués. Vous devez faire les mêmes opérations que dans le TP 2 pour enregistrer les paquets. Le nom du fichier n'a pas d'importance lorsque vous le soumetterez sur le site Web local de la machine virtuelle.

IMPORTANT : Vous devez utiliser le bon port "XX" relié à votre numéro d'équipe (e.g. 09, 42, ...)