Réseaux pour ingénieurs
GLO-2000

TP3 : Programmation socket

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

Responsables des travaux pratiques :
Pierre Wendling et Philippe Dessaints
GLO2000.laboratoires@gmail.com

Université Laval
Faculté des sciences et de génie
Automne 2021

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.

Modalités de remise

- À faire en équipe de 3.
- Les fichiers à remettre sont :
  - Le fichier « TP3_Q1.py », **ne renommez pas ce fichier et n'incluez pas les modules fournis**.
  - Le fichier « TP3_Q2.pcap(ng) », contenant les paquets échangés lors de l'exécution du programme.
  - Les deux fichiers sont à rendre dans une unique archive zip ou tar nommé : « TP3_equipe_<*votre numéro d'équipe*> ».

N. B. : Une pénalité de 5% sera appliquée pour chaque point non respecté.

- Remise uniquement via le portail des cours.
- Date limite de remise est la suivante : **20 octobre 2021 à 23h00 (fuseau horaire de Québec)**.
- Tout travail remis en retard se verra attribuer la note **0**.

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

1.1 Mise en contexte

Supposons que deux individus, Alice et Bob, souhaitent se communiquer 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.

a = modulo
b = base
p = clé privée Alice
q = clé privée Bob
P = clé publique Alice
Q = clé publique Bob
Q^p mod a = Clé partagée Bob
P^q mod a = Clé partagée Bob

Alice connaît alors m 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, $b^p \mod a$ et $b^q \mod a$ est incapable de calculer $k = b^{pq} \mod a$, puisque cela demande par exemple de déduire p de $b^p \mod a$, ce qui est un problème trop difficile pour un ordinateur.

1.2 Modules fournis

Deux modules vous sont fourni et sont directement importés dans le squelette qui vous est fourni.
Le premier module, glocrypto, vous apporte les fonctions suivantes :

- **trouver_nombre_premier()** : Retourne un nombre premier sur 128bits. Note : la fonction utilise le test de Fermat, la primalité du nombre n'est donc pas garantie.
- **entier_aleatoire(modulo)** : Retourne un entier aléatoire entre 0 (inclus) et *modulo* (exclus).
- **exponentiation_modulaire(base, exposant, modulo)** : Calcule efficacement le résultat de l'opération : $base^{exposant} \mod modulo$

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

- **send_msg(destination, message)** : Encode le message et l'envoi à la destination. Lève une exception en cas d'erreur.
- **recv_msg(source)** : Récupère le message depuis la source et le décode. Retourne None si la source s'est déconnectée.

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.

1.3 Spécifications du programme

Votre programme doit respecter les contraintes suivantes :

- Le mode TCP doit être utilisé.
- Utiliser l'option `-p` ou `--port` pour désigner le port à utiliser.
- Utiliser l' option `-l` ou `--listen` pour démarrer l'application en mode serveur, sinon le mode client est utilisé par défaut.
- Utiliser l'option `-d` ou `--destination` pour désigner l'adresse IP du destinataire.
- Utiliser l'option `-6` pour démarrer l'application en mode IPv6, par défaut le programme utilise IPv4.
- Le programme s'exécute en ligne de commande. Par exemple :
  `python3 TP3_Q1.py -l -p 12345` lance l'application en mode serveur sur le port 12345. Vous devez vous assurer que l'appel contient obligatoirement l'option `-p/--port`.
- Vous devez vous assurer que les options `-l/--listen` et `-d/--destination` ne sont pas dans le même appel (on ne peut pas être un serveur et un client en même temps). Par exemple, l'appel suivant est invalide :
  `python3 TP3_Q1.py -l -d localhost -p3333`
- En cas d'erreur, le client doit quitter tandis que le serveur doit fermer la connexion avec le client et attendre le prochain.
- Chaque fois que votre programme génère un entier aléatoire, trouve un nombre premier ou exécute une exponentiation modulaire, cela doit obligatoirement être fait via le module fourni glocrypto.
- Chaque échange entre le client et le serveur doit être effectué avec le module fourni glosocket.
- Le serveur génère le modulo et la base et les envoie au client. Le modulo doit être un nombre premier et la base un entier naturel plus petit que le modulo.
- À la fin des échanges, la clé partagée doit être la même du côté serveur et client, mais ne jamais avoir été communiquée en clair sur le réseau.

Le fichier « TP3_Q1.py » vous est fourni, il contient le squelette des fonctions à implanter. Vous êtes libre d'y ajouter des fonctions auxiliaires si nécessaire pourvu que chaque fonction suive le comportement documenté. Les annotations de type vous indiquent 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.

2 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 le port 43850. 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 TP2 pour enregistrer les paquets. Nommez le fichier « TP3_Q2.pcap(ng) ».