NOTES DE DÉVELOPPEMENT - PROJET PROTOCOLES INTERNET

DUPONT-LATAPIE Ulysse 21981115
GUINARD Océane 21952743

Choix d'implémentation :

Nous avons décidé de fournir à l'utilisateur une interface CLI, où l'entrée de commandes avec aide fournie permet de consulter l'annuaire, de s'y enregistrer, de configurer des options de cryptographie ou de debug. Le CLI propose deux modes, le mode REST pour la récupération d'informations et le mode UDP pour la connexion aux pairs et la récupération de données. Nous savons que son fonctionnement est un peu verbeux et lourd, et nous avons envisagé des améliorations, notamment des mises en cache des adresses, clés ou hashes pour éviter la recopie une fois en mode UDP, mais elles n'ont pas été implémentées faute de temps.

La connexion REST s'effectue depuis une liaison TCP ; pour ce qui est de l'UDP et l'enregistrement au serveur, on utilise une socket UDP. Nous avons choisi d'utiliser des builders de requêtes et readers, adaptés aux opérations souhaitées et types de message. Cela permet une robustesse en écoute, car notre pair peut répondre aux requêtes avant de parler, par exemple si l'on lui demande sa clé alors qu'il souhaite récupérer une donnée. Il est également capable d'attendre pour pallier d'éventuels délais réseau.

Notre pair ne réémettra pas un paquet non-acquitté, ce qui peut causer des problèmes de handshake.

En ce qui concerne les arbres de Merkle, nous avons fait une structure de donnée assez simple où les parents contiennent leurs enfants, eux-mêmes contenant une référence au parent : s'il s'agit d'un chunk, il contient des données, sinon il contient uniquement les informations des enfants.

Lors d’un download, on télécharge les noeuds 1 à un 1 de façon récursive : si jamais on reçoit quelque chose d'incohérent que ce soit un noeud pas demandé, des données incorrectes par rapport au hash, ou un paquet Datum avec un type inconnu, on affiche une erreur pour l’utilisateur. Cette erreur n’est cependant pas transmise sous forme d’Error ou ErrorReply.

Nous entretenons la connexion active avec des keepalives depuis une goroutine, pour éviter les timeouts.

Un listener passif reçoit et traite les requêtes des autres pairs : ainsi, nous pouvons partager une arborescence ou renseigner des informations.

Extensions :

Notre pair supporte les signatures cryptographiques ; nous avons également fourni un support d’erreur riche et verbeux, désactivable si besoin. Nous pouvons générer des clés, vérifier des signatures, les communiquer, signer nos messages. Cependant, durant nos tests, le serveur REST a refusé nos signatures, les classifiant d’erronées : nous supposons une erreur d’implémentation mineure.

Nous supportons un système d’export de clés au format PEM.

De même, nous supportons en théorie la traversée de NAT, qui n’a elle aussi pas été suffisamment testée. Les quelques tests effectués semblent exhiber un envoi des requêtes, mais leur réception n’a pas été constatée.

Notre pair supporte une arborescence et est capable de l’exporter, à l’aide d’un listener passif, mais ceci n’a pas été testé suffisamment, en effet un testing d’upload demande généralement une traversée de NAT.

Notre pair ne peut pas supporter plusieurs connexions parallèles.

Support d'erreur :

Notre pair est capable de détecter des erreurs, et même de répondre à certaines avec des Error ou ErrorReply, notamment dans les cas suivants :

- Requêtes avant validation du handshake entre pairs
- Signature invalide pour un pair annonçant une clé publique
- Hash ne correspondant pas à la donnée demandée
- Type de message inconnu

Problèmes connus :

- Notre pair corrompt systématiquement le dernier octet de chaque fichier téléchargé.
- Nos signatures sont rejetées par le pair REST : nous n'avons pas su trouver l'erreur dans notre implémentation cryptographique, qui reprenait l'annexe du sujet.
- Nos fichiers et connexions ne sont pas systématiquement fermées.
- Nos goroutines ne sont pas systématiquement terminées.
- Interface verbeuse sans mise en cache des données pour réutilisation simple.