esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

Cryptographie asymétrique

Projet : Blockchain

Professeur : David POINTCHEVAL

Auteur :
Jessica FAVIN

Le projet

Mon programme est découpé en 4 classes :

1. BlockChain

La classe principale représentant la chaîne de blocs. Elle comporte un attribut **chain** qui est une liste chaînée de bloc. Le type LinkedList m'a paru le plus proche de la représentation d'une blockchain car il lie bien chaque bloc au suivant en conservant l'ordre d'insertion.
La fonction **main** se trouve dans cette classe. Cette fonction effectue les opérations demandées dans le sujet telles que la création de blocs, l'ajout à la chaîne la création de porte-monnaie ou encore la vérification de l'intégrité d'une transaction demandée.

2. Block

Cette classe représente chaque bloc de la chaîne. Elle a comme attribut le hash de bloc précédent **previousBlock**, le message de transaction **tx**, son propre hash **hash**, son sel **salt** s'il a été généré, et le message de transaction signé **signedTx** si celui-ci a été fourni. Les seuls blocs n'utilisant pas de sel sont ceux de la partie "Hash Chain".

3. Wallet

Cette classe représente les porte-monnaie et possède comme attribut une clé publique **pubK** et une clé privée **privK** afin de signer et vérifier la signature des messages de transactions.
La courbe "secp256k1" est utilisée pour la génération des clés.
La signature des messages de transactions est gérée dans la fonction privée **sign** de Wallet afin de s'assurer que la clé privée reste uniquement utilisée au sein de cette classe. Cette fonction retourne le tableau de BigInteger résultant de la signature du message par la clé privée. Avant d'être signé le message est hashé en SHA1 comme l'indique la documentation.
La clé publique elle est bien accessible depuis toutes les classes grâce à la fonction getPublic().

4. Utils

Cette classe contient les fonctions utiles : les fonctions de hashage **sha1** et **sha3**, la fonction calculant un sel, **findSalt**, permettant d'obtenir la séquence "0000" en fin de hash ainsi que la fonction de vérification de la signature **verify**. Cette dernière fonction prend en argument la clé publique ayant signé le message, le message signé et le message en clair et vérifie que c'est bien la clé privée associée qui a signé le message.
Comme lors de la signature le message est hashé en SHA1 avant d'être utilisé par la fonction de vérification.

Difficultées rencontrées

La principale difficulté a été de comprendre la signature ECDSA avec Bouncy Castle. Ayant au départ un peu de mal avec les notions de chiffrement à courbe elliptiques orienter mes recherches m'a été difficile au début. Cependant une fois les principes mieux compris il m'a été plus facile de comprendre quoi faire et avec quels types d'objet. Malgré cela la documentation Bouncy Castle est très dense et même en sachant quoi chercher cela a parfois été compliqué.

Conclusion

Ce projet m'aura permis de bien mieux appréhender les notions de chiffrement utilisés dans les blockchain et de mieux visualiser leur fonctionnement. En effet les articles en parlant, parle rarement de l'aspect mathématique qui va avec et qui est pourtant à la base du fonctionnement d'un système de blockchain.