Rapport GS15

Pierre Sarazin
Benoit Estrade
Master SSI 2

Kasumi

Toutes les exigences ont été respectées

Problèmes rencontrés :

- L'encodage de l'information était de loin le problème le plus important. On a essayé tout un tas d'encodages. Les zéros en bit de poids forts étaient parfois supprimés lors de la concaténation. Il nous a été impossible de trouver une solution pratique et fiable. Les accents n'ont pas marché dans toutes nos solutions :
    - Solutions testés :
        - Encodage par 8 bit pour que la concaténation s'y tienne ;
        - Base 64 : Cela s'est avéré inutile dans notre cas.
    - Solution implémentée : On a implémenté un offset calculé par la fonction de chiffrement. Cet offset correspond, pour chaque bloc de 64 bits, au nombre de bits à zéro en poids fort. La fonction de déchiffrement recevait deux listes : une liste contenant les blocs de 64 bits chiffrés et une deuxième liste contenant l'offset. Les deux listes font la même taille. C'est la seule méthode qui, une fois implémentée, fonctionnait avec les caractères spéciaux et avec un taux de réussite de 100%.
- Pour la manipulation des bits, il a été utilisé à tort et à travers la conversion des entiers en *String*. Pour la suite du projet, nous avons changé pour utiliser uniquement des entiers avec les opérations logiques propres à python (beaucoup moins verbeuses).
- La fonction d'inversion sur un corps de Gallois s'est faite très simplement à l'aide de la bibliothèque "pyfinite".

```
def inversion_I(a):
    F = ffield.FField(15)
    a_inverse = F.Inverse(a)
    return a_inverse
```

RC4

La création de Sbox dépend de la clé de chiffrement. Elles sont régénérées à chaque chiffrement.

Génération de clé publique/privée

Toutes les exigences ont été respectées

Pour la génération d'un entier premier 'n' de 512 bits, nous avons utilisé le test de Miller-Rabin qui s'exécute avec un facteur de précision 'k' fixé à 10 ce qui nous donne une très forte probabilité que 'n' soit pseudo-premier.

Nous avons eu quelques difficultés pour trouver un élément générateur 'alpha' de Zn. Nous avons essayé de vérifier dans un premier temps si pour un alpha quelconque, alpha^((p-1)/q) était différent de 1 pour chaque facteur premier 'q' de p-1. Cette solution bien que fonctionnelle demandait beaucoup trop de ressources et de temps à l'exécution. Nous avons finalement résolu ce problème en effectuant une combinaison des facteurs premiers ce qui a permis de radicalement diminuer la liste des facteurs premiers à tester. Pour que le

résultat soit un peu plus aléatoire, nous ajoutons plusieurs éléments générateurs à une liste qui est ensuite mélangée puis un élément générateur est récupéré aléatoirement.
Il a ensuite été facile de générer des couples de clés et de les stocker dans des fichiers.

Hash SHA-1 & Fonction éponge

Toutes les exigences ont été respectées

- Du fait que ce soit SHA-1 l'algorithme décrit dans le cours, nous l'avons repris.
- SHA-1 n'a pas particulièrement été modifié pour la construction de la fonction éponge.

Génération de signature

Toutes les exigences ont été respectées

El-Gamal

Le mécanisme de signature El-Gamal n'a pas posé de problème. Il est simple de signer un message avec la clé privée de l'émetteur et de vérifier la signature avec sa clé publique. Cette signature est plus randomisée que RSA puisque chaque signature générée pour un même message ne sera pas la même. C'est donc cette méthode de signature qui a été choisie pour la chaîne de blocs.

RSA

Pas de difficulté rencontrée. La génération du duo de nombre premier était complexe. Une série de ces duos était stockée pour faciliter l'échange.

Chaîne de Blocs

Toutes les exigences ont été respectées

La programmation orientée objet nous a semblé être le choix le plus judicieux pour coder la blockchain. Les utilisateurs, la blockchain et les blocs sont plus facilement manipulables.

- **La Class User** : Elle permet de gérer la création des utilisateurs avec leurs attributs : nom, argent, clé publique et clé secrète. Chaque utilisateur peut effectuer une transaction s'il a les fonds nécessaires.
- **La Class Block :**
  - Elle permet d'instancier les blocs avec les paramètres du bloc précédent. On a rajouté un attribut timestamp pour la date de création du bloc et un "count" pour compter le nombre de calculs effectués avant le minage du bloc. L'attribut "confirmed transactions" est une liste contenant toutes les transactions du bloc.
  - La fonction de minage repose sur la 'difficulté' attribuée à la blockchain, c'est-à-dire au nombre de zéros qui doivent figurer à la fin du hash du bloc calculé. A chaque nouveau calcul du hash, notre 'salt' qui correspond à 100 bits générés aléatoirement se renouvelle. Le calcul du hash du bloc prend bien en compte tous les attributs du bloc (index, previous_hash, timestamp, transactions[ ], salt et count).
- **La Class Blockchain**:
  - Elle contient l'ensemble des blocs minés et une liste de transactions "en attente" qui sont ajoutées au bloc qui n'est pas encore miné. C'est une fois que le bloc est miné et ajouté à la blockchain que cette liste est remise à zéro. Elle comprend aussi une liste qui contient toutes les signatures digitales des transactions pour leur vérification.
- **Transactions** : Chaque transaction est signée (mécanisme El-Gamal) et la signature vérifiée avant son ajout à un bloc. Nous avons aussi donné la possibilité à un utilisateur de vérifier la signature de n'importe quelle transaction inscrite dans la chaîne. Nous avons fixé le nombre maximum de transactions à 2 par bloc.
- **La vérification de la blockchain** :
  - Vérification du premier bloc qui est spécifique (index=0, pas de previous_hash)

    - Vérification de tous les blocs suivant avec vérification de : l'index, du hash précédent, du hash du bloc qui est recalculé pour la comparaison et de la preuve de calcul (le hash doit finir avec le nombre de bits à 0 fixés).

Nous avons également implémenter une interface qui permet :

- D'ajouter une nouvelle transaction à la blockchain
- De vérifier la signature de n'importe quelle transaction
- De vérifier la validité de la blockchain à tout moment
- D'afficher les utilisateurs et leur portefeuille
- D'ajouter un utilisateur
- D'afficher la blockchain

Il a été un peu fastidieux d'ajouter correctement les transactions rentrées manuellement par l'utilisateur (après génération de la blockchain) à un bloc mais nous n'avons pas rencontré de difficulté particulière autrement.