

Année universitaire 2019/2020

Projet programmation par composant

Dossier de spécifications techniques du composant N°6 :

Signature

Membres de l'équipe :

AÏT-MOULOUD Lisa

ARZUR Samuel

HAMI Hocine

LEBIB Mohamed

Liste des modifications du document

Version	Date	Description des modifications
1.0	07/04/2020	Création du document
1.1.1	16/04/2020	Génération des paires de clés publique/privée
1.1.2	18/04/2020	Modification de la signature de la fonction « generate »
1.2	19/04/2020	Rendu version 1
2.0	26/04/2020	Mise à jour suite à l'implémentation du code

Documents de référence

Nom du document	Description
<i>Programmationpar_composants_2.ppt</i>	Pseudo dossier de spécifications générales concernant la structure de la Blockchain
<i>Schema_bloc.ppt</i>	Document précisant les relations d'utilisation et d'inclusion des composants de la Blockchain
<i>Les fondamentaux de la blockchain</i>	https://systematic-paris-region.org/fr/les-fondamentaux-de-la-blockchain/

1. Contexte et objectifs

Dans le cadre du cours « Programmation par composants », il nous a été demandé d'implémenter une Blockchain fonctionnelle. Et ceci, en utilisant une approche modulaire qui respecte les principes de la programmation orientée composant.

L'idée est d'implémenter l'architecture d'une Blockchain sous forme de plusieurs petits modules. Chacun de ces modules, qu'on appellera « composant » dans la suite du document, sera implémenté par un groupe différent. Afin d'assurer une lisibilité et réutilisabilité du code, ainsi qu'une facilité d'implémentation.

Le document portera donc sur l'implémentation d'une Blockchain simplifiée qui simulera des transactions **Bitcoin** simples entre plusieurs portefeuilles.

2. Blockchain et composants

La *blockchain* est une technologie de stockage et de transmission d'informations à faible coût, sécurisée, transparente et fonctionnant sans organe central de contrôle. Par extension, la *blockchain* désigne une base de données sécurisée et distribuée (car partagée par ses différents utilisateurs), contenant un ensemble de transactions dont chacun peut vérifier la validité. La *blockchain* peut donc être comparée à un grand livre comptable public, anonyme et infalsifiable.

Dans notre implémentation, la Blockchain sera simulée par un **fichier (composant 1)** qui contient un verrou. Celui-ci empêche l'ouverture, si le fichier en question et donc la blockchain contient une incohérence.

Le fichier qui simulera la Blockchain contiendra des données structurées qui représentent les **blocs (composant 4-a)**. Ces blocs en question résument les transactions (envoi et réception de Bitcoin) effectuées entre les utilisateurs. Ces derniers sont représentés par des **portefeuilles/wallet (composant 2)** et **signés** avec une clé privée (**composant 6**).

Les blocs sont identifiés par le **hash (composant 4-b)** qui se base sur tous les éléments qui y sont contenus pour sortir un code d'identification unique.

Toute transaction effectuée est d'abord mise en attente, puis elle est par la suite validée par un module appelé « **mineur** » (**composant 3**).

Les incohérences dans le contenu des blocs et dans les transactions sont détectées par un module appelé le **vérificateur de bloc (composant 5)**. Celui-ci se base sur la **signature** du bloc (**composant 6**) en utilisant des clés publiques pour vérifier si les données sont toujours intègres.

Les liens entre les composants sont résumés et schématisés dans le fichier cité dans les références [*Schema_bloc.ppt*].

3. Spécifications techniques du composant 6 : signature

Lorsqu'un utilisateur enregistré souhaite ajouter une transaction à la blockchain, il la soumet au mineur pour que celui-ci puisse la recevoir, la vérifier et l'ajouter au bon bloc. Mais pour être vérifiées, les transactions (qui sont des données) doivent être signées afin que le mineur puisse identifier incontestablement l'expéditeur.

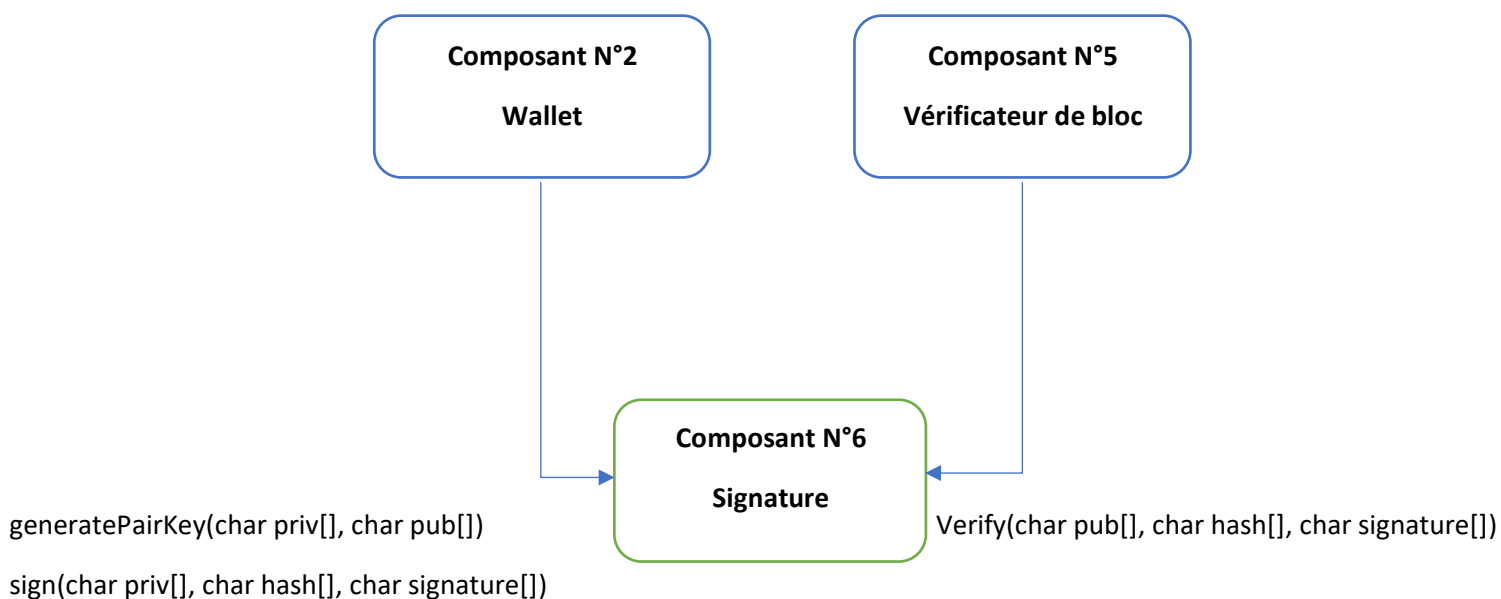
La signature est générée par une fonction cryptographique qui prend en entrée les données d'origine et la clé privée de l'expéditeur, et fournit en sortie une signature. Cette signature est vérifiable par toute personne utilisant une

autre fonction qui vérifie la cohérence entre les données, la signature et la clé publique. Par conséquent, lors de l'envoi de données l'expéditeur ajoute également la signature correspondante et sa clé publique.

Notre composant est chargé de :

- Générer les paires de clés : publique/privée
- Signer les données
- Valider une signature

Ci-dessous, une représentation schématique de l'utilisation de notre composant par les autres composants du projet.



1. Générateur de paires de clés :

Chaque transaction a recours à la cryptographie asymétrique, la génération de cette paire de clés présente le double intérêt de chiffrer ou de signer un message. Ainsi, dans ce système un utilisateur crée une suite aléatoire de chiffres, appelée clé privée. Puis à partir de celle-ci un algorithme permet de produire une seconde clé, appelée clé publique.



Dans le schéma ci-dessus on prend l'exemple du Bitcoin où il s'agit d'un algorithme de signature numérique à clé publique dit à courbes elliptiques appelé ECDSA (pour *Elliptic Curve Digital Signature Algorithm*) qui effectue les étapes suivantes pour la création des clés.

- **Préparation des clés :**
 - a) Choisir un entier s entre 1 et $n - 1$ qui sera la clé privée.
 - b) Calculer $Q = sG$ en utilisant l'élément de la courbe elliptique. (G un point de la courbe elliptique)
 - c) La clé publique est Q et la clé privée est s .

En-tête de la fonction de génération de paires de clés :

generatePairKey(char priv[], char pub[])

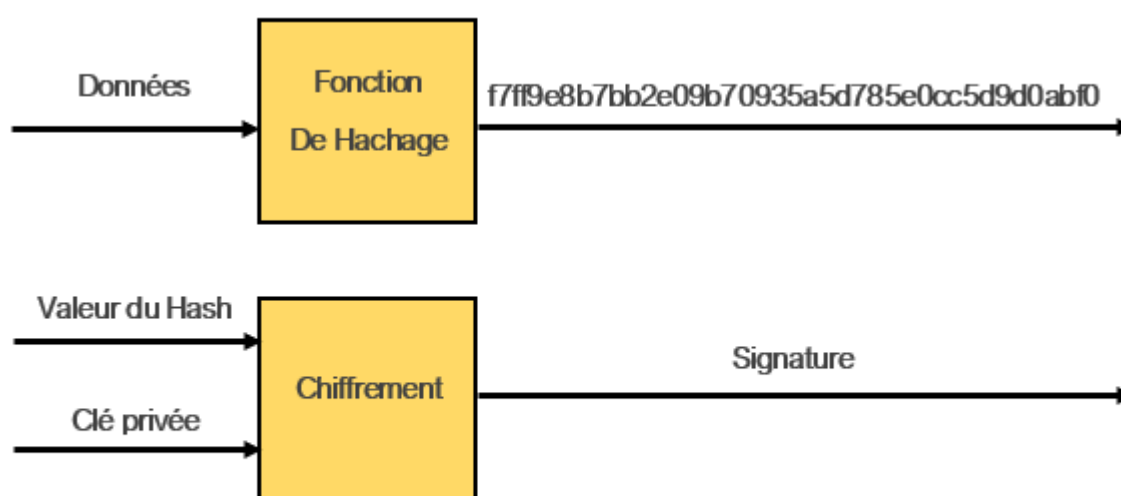
- Entrées:
 - *priv* : tableau pour accueillir la clé publique qui sera gardée dans le portefeuille/wallet pour être utilisé lors de la vérification du bloc.
 - *pub* : tableau pour accueillir la clé privée qui sert à signer le bloc

2. Signature des données :

Signer un bloc d'une Blockchain revient à effectuer les étapes suivantes :

- Récupérer le hash du bloc généré par le **composant 4 : hacheur SHA-256**. Ce dernier génère un code SHA-256 en se basant sur les données contenues dans le bloc (ID, date de création, émetteur, récepteur, montant et le hash du bloc précédent).
- Utiliser une clé privée générée par le composant lui-même pour chiffrer le hash récupéré.
- Envoyer le bloc, sa signature et une clé publique au vérificateur de bloc pour que celui-ci puisse effectuer sa vérification.

La procédure de signature d'un bloc est résumée dans le schéma ci-dessous :



Création d'une signature

En-tête de la fonction de signature d'un bloc :

La fonction qui se chargera de signer les blocs de la Blockchain sera donc à priori comme suit :

sign(char priv[], char hash[], char signature[])

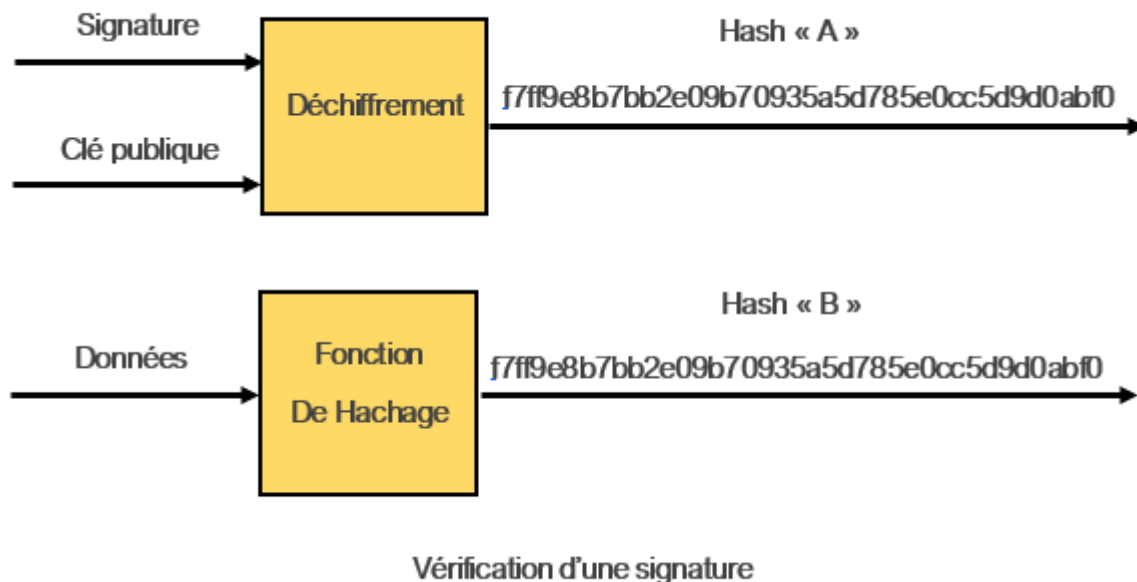
- Entrées:
 - *hash(Bloc)* : tableau du hash SHA-256 (composant 4).
 - *priv* : une clé privée générée en paire clé publique/clé privée qui est stockée dans le portefeuille/wallet.
 - *Signature* : tableau pour accueillir la signature digitale d'un bloc qui sera sous forme de hash également.

3. Vérification de la signature :

Vérifier un bloc se résume en 4 étapes. Le vérificateur de bloc doit recevoir un bloc et une clé publique.

- La clé publique est utilisée pour déchiffrer la signature du bloc. Cette opération donne en sortie un hash qu'on appellera « **hash à vérifier** ».
- D'un autre côté, il faudra appliquer l'algorithme de hash du composant 4 au bloc reçu pour re-générer le bon hash du bloc. On appellera le code obtenu « **hash réel** ».
- La dernière étape serait donc de comparer le « **hash à vérifier** » et le « **hash réel** ».
Si les deux sont identiques, cela prouve que le bloc n'a pas été altéré durant le transit et que les données qu'il contient sont toujours intègres. Sinon, le bloc est considéré comme non valide.

La procédure de vérification d'une signature d'un bloc est résumée dans le schéma ci-dessous :

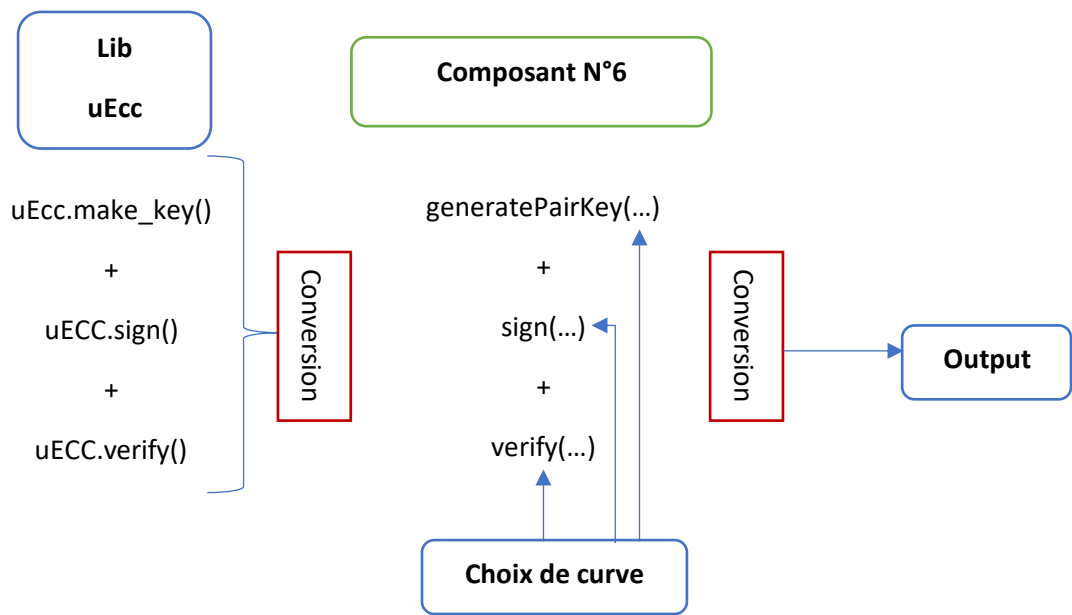


En-tête de la fonction de vérification de signature d'un bloc :

La fonction qui se chargera de vérifier les signatures des blocs de la Blockchain sera donc, à priori, comme suit :

Verify (char pub[], char hash[], char signature[])

- Entrées :
 - *pub* : une clé publique générée en paire clé publique/clé privée et qui correspond à la clé privée avec laquelle a été signé le bloc
 - *signature* : le résultat du chiffrement du bloc avec la clé privée
 - *hash* :
- Sortie :
 - *boolean* : true si la vérification est correcte, false sinon.



4. Négociations

Après concertation avec les autres groupes nous en sommes venus à la conclusion que c'est le groupe du composant n°2 qui va générer les paires de clés (**), puis ils vont appeler la fonction qui permet de signer avec la clé privée de la paire générée (**) et joindre la clé publique à la transaction avant de l'envoyer aux autres composants.

Aujourd'hui on a un problème de types dans nos fonctions. Nous manipulons des tableaux de `uint8_t` alors que les autres composants nous transmettent et ont besoins de `char[]`.