Programmation par composants

Composant name : Signature
Version: 1.0

Yacine AISSAT
Naïm BENDJEBBOUR
Abderraouf HADDAD
Lounes TIOUCHICHINE

20 – 03 – 2019

Sommaire

Historique du document

Informations générales

UE :	Programmation par composants
Rédacteur(s) :	Yacine AISSAT Naïm BENDJEBBOUR Abderraouf HADDAD Lounes TIOUCHICHINE
Valideur(s) :	Jose LUU

Historique des mises à jour

Version	Date	Description
0.1	19/03/19	Création du document
1.0	20/03/19	Finalisation de la première version pré-intégration

Documents de référence

Repository GitHub :

```
https://github.com/NaimBendjebbour/C-ProgrammationParComposant.git
```

1. Présentation du composant

1.1 Introduction

Ce document rassemble les spécifications concernant le composant 6 « Blockchain » du projet de programmation par composant, dit composant Signature.

L'objectif de ce composant est double :

D'abord, il se charge de créer de nouvelles adresses, c'est-à-dire génère une paire de clés cryptographiques composée d'une clé privée et d'une clé publique.

Ensuite, ce dernier permet aussi de signer un message avec une clé privée (dont seul l'émetteur a la connaissance) et de vérifier la signature en utilisant la clé publique correspondante.

1.2 Objectifs

La signature permet au récepteur de l'information de s'assurer qu'elle provient bien d'un émetteur connu à l'avance.

En d'autres termes, la signature certifie l'identité d'un émetteur d'une transaction et donc de ne pas accepter de transactions frauduleuses ou d'origine inconnue. En effet, le bénéficiaire, à l'aide de la clé publique fourni par l'émetteur, peut être certain de l'origine de la transaction

Dans un premier temps, l'émetteur hashe la transaction puis chiffre le hashage obtenu grâce à la clé privée. Après cela, il envoie au destinataire la transaction, le hasha correspondant ainsi que la clé publique liée à la clé privée.

Le récépteur déchiffre le hashage de la transaction et confirme l'origine de celle-ci. Il compare ensuite le hashage reçu avec le résultat du hashage de la transaction (non hashée) reçue.

Le schéma suivant résume bien le rôle du composant Signature :

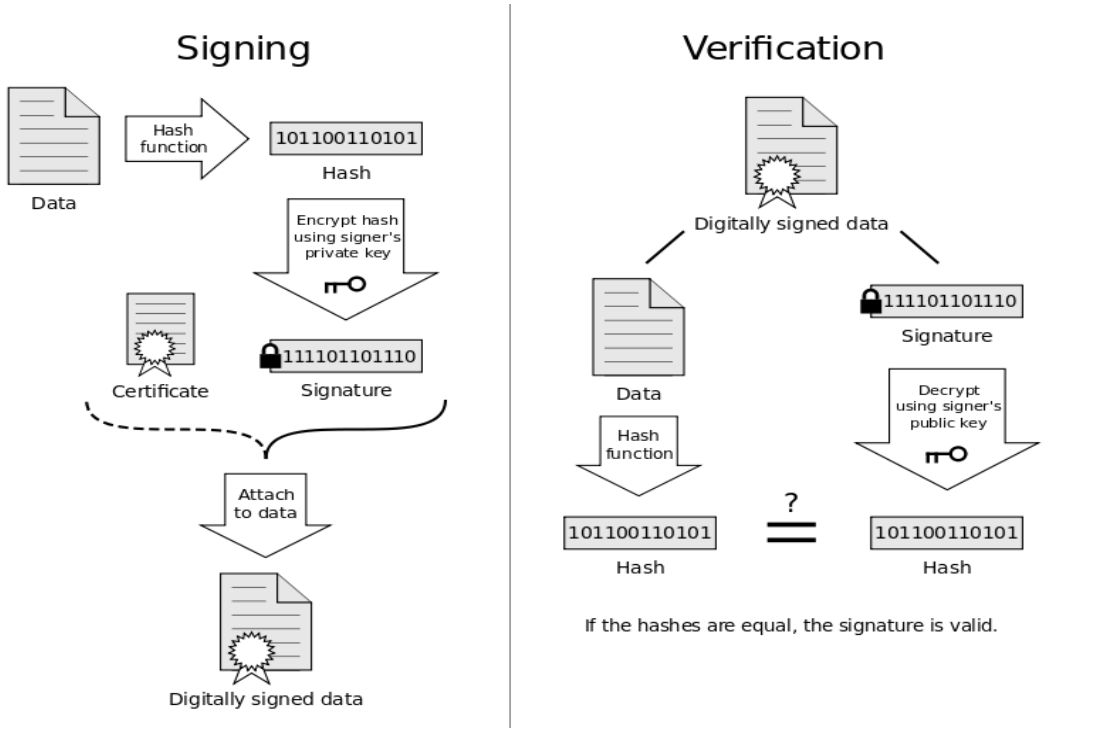

Figure 1 Schéma signature numérique

1.3 Traitement

L'objectif final est de se connecter à d'autres composants afin d'utiliser leurs solutions.

Voici un schéma explicatif du mode de fonctionnement demandé :

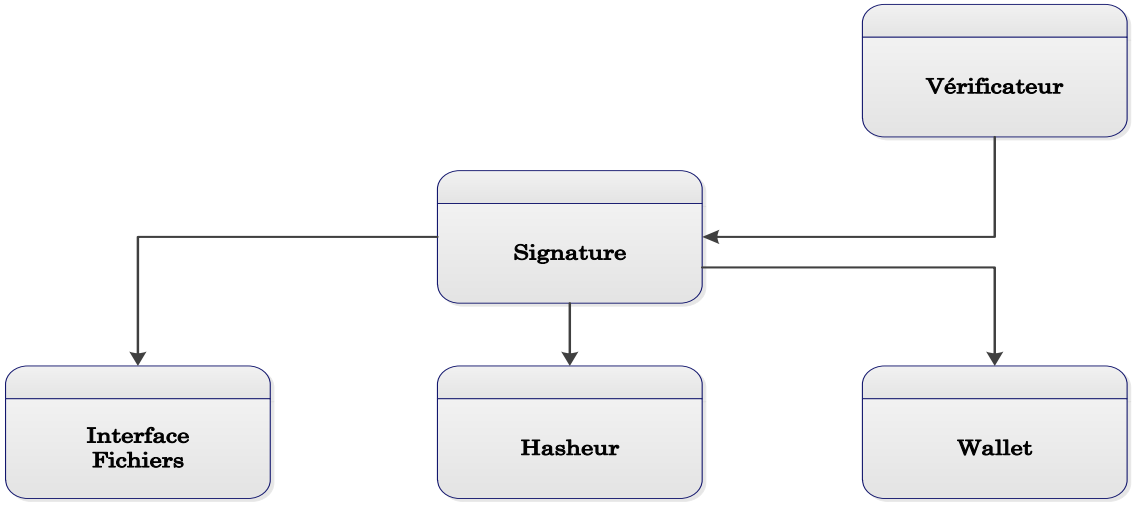

Figure 2 Schéma du Blockchain

Notre composant s'interfacera avec plusieurs autres :

- Le hasheur, qu'on va appeler afin de valider ou non une signature.
- Le vérificateur, qui va nous appeler afin de valider ou non l'authenticité d'une transaction
- L'interface fichiers, afin d'enregistrer nos clés créées sur des fichiers .key
- Le Wallet, afin de composer une transaction grâce à nos clés.

2. Fonctions du composant

2.1 Clefs genererClefs()

La fonction `genererClefs()` doit renvoyer un object Clefs contenant deux attribut de type string : la clef privé ainsi que la clef public .

Ces deux clefs ont une longueur de 256 bits générée grâce à l'algorithme ECDSA (Elliptic Curve Digital Signature Algorithm).

Cette fonction ne prend aucun paramètre en entrée.

Les deux attributs de l'object Clef se nomment : private_key et public_key.

Afin de tester, nous générons les clefs dans deux fichiers distincts.

Vous avez la possibilité de tester à l'aide du .exe fourni dans le dossier Release.

2.2 Signature signerMessage(String msg, String cle_prive)

Cette fonction doit retourner un objet de type signature (contenant une seule donnée membre de type chaîne de caractères (String)) correspondant à la signature de la transaction passée en paramètre.

Elle prend en paramètre une chaine de caractère correspondant à la transaction à signer ainsi qu'une clé privée de 256 bits.

2.3 Bool validerMessage(String msg, String cle_publique, String signature)

Cette fonction doit renvoyer une valeur booléenne (true ou false) correspondant à la validation ou non de la signature.

Elle prend en paramètre une chaine de caractère correspondant à la transaction qu'il faut authentifier, une chaine de caractère correspondant à une clé publique de 256 bits ainsi qu'un objet de type Signature correspondant à la signature qu'il faut authentifier

3. Cas d'erreurs

3.1 Lors de la signature des messages

Fonction: Signature signerMessage(String msg, String cle_prive)	
Erreur: Données nulles	***Null Value***
Erreur: Données manquantes	***Not Enough Argument***
Erreur: Données en plus	***Too Much Argument***
Erreur: Mauvais type d'argument	***Invalid Type Argument***

3.2 Lors de la validation d'un message

- Si l'un des paramètres est nul alors le message d'erreur suivant doit être renvoyé :

« NULL VALUE »;

- S'il y'a plus de 3 paramètres alors le message d'erreur suivant doit être renvoyé :

« TOO MUCH ARGUMENT »;

- S'il manque au moins un des paramètres alors le message d'erreur suivant doit être renvoyé :

« NOT ENOUGH ARGUMENT »;

- Si la fonction est appelée avec des types différents que ceux attendus, le message suivant est retourné :

« INVALID TYPE ARGUMENT »;