Projet par composants
M2 MIAGE IF APP
2015-2016
Université Paris Dauphine

# BLOCKCHAIN LOCALE :

# Spécification Composant 6

***Générateur de paire de clés***

***Signature de données***

***Vérification signature***


**Réalisé par :**

**Groupe1**

- **ENNAJIMI Yassine**
- **MOUZITA Coccley**
- **PADONOU Jimmy**
- **SOUSSOU Kamel**

| Version doc | Date | Auteur(s) | Modifications |
|---|---|---|---|
| 1.0 | 27/01/2016 | Jose Luu | Version initiale |
| 1.1 | 18/02/2016 | Groupe 1 | Objectif du composant |
| 1.2 | 26/02/2016 | Groupe 1 | Etapes et description des erreurs |
| 1.3 | 28/02/2016 | Groupe 1 | Rédaction du plan de test |
| 1.4 | 01/03/2016 | Groupe 1 | Mise à jour cas d'erreur et plan de test |
| 1.5 | 08/03/2016 | Groupe 1 | Revue des spécifications |
| | | | |

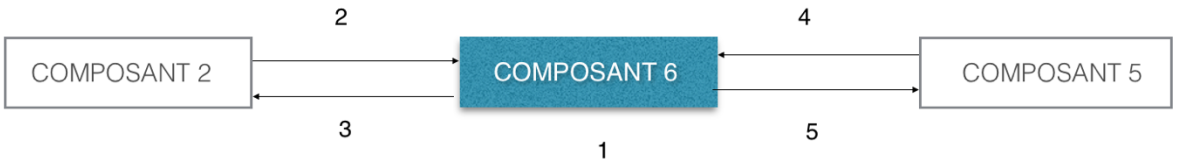
Ce document rassemble les spécifications concernant le composant 6 « Blockchain » du projet de classe.

# I. Fonctionnement du composant

L'objectif du composant 6 est premièrement de créer de nouvelle adresse c'est-à-dire génère une paire de clés cryptographiques composé d'une clé privée et d'une clé publique.

Ce composant permet également de signer un message avec une clé privée (que seul l'émetteur connait) et de vérifier la signature en utilisant la clé publique correspondante.

Voici un schéma explicatif du mode de fonctionnement du composant demandé :




1) Génère la paire de clés (Publique & Privée)
2) Réception de Appel du composant 2 pour la signature de données
3) Retour de la signature de données
4) Réception d'Appel Valider signature venant du composant 5
5) Retour de Valider signature

Chacune des étapes du schéma seront explicitées afin de comprendre le fonctionnement du composant et son mode de réalisation.

# II. Les étapes

## 1) Génération de la paire de clés

Notre composant génère une paire de clés **unique** (privée et publique) pour chaque utilisateur à partir d'une **source d'aléa** (nombre aléatoire, le temps…). La clé publique sera connue de tous et la clé privée par un seul utilisateur.

Pour créer une transaction, il est nécessaire d'avoir une clé privée pour l'émetteur et la clé publique du destinataire ainsi que le montant.

La méthode renvoie une paire de clé (chaine de caractères) composée de la clé privée et de la clé publique sépare par un caractère. Cette chaine est de la forme : « **Clé 1** + *Séparateur* + **Clé 2** »

***String generateKeys() ;***

## 2) Réception de la demande du composant 2(Wallet)

Le « Wallet » qui compose une transaction fournit à notre composant plusieurs paramètres permettant la signature des données :

- hash
- La clé privée de l'émetteur

***String signData (string hash, string privateKey);***

Cette méthode retourne une signature à partir du hash et de la clé privée en utilisant l'algorithme ECDSA. Cette dernière permettra d'authentifier l'émetteur de la transaction par le destinataire.

## 3) Retour de la demande au composant 2

Le résultat de la fonction ***signData*** est renvoyé au « Wallet ».

## 4) Réception de la demande du composant 5

Le composant 5 s'occupe de la vérification des blocs et transactions. La vérification d'une transaction implique la vérification de la signature.

Le composant 5 fait appel à la méthode ***checkSign*** du composant 6 en passant la clé publique et la signature de la transaction en paramètres.

***bool checkSign (String clePub, String sign, string hash)***

> Si hash **identique à** decrypt(clePub, sign) Alors
> Renvoi Vrai
> Sinon
> Renvoi Faux
> Fin

## 5) Retour de la demande du composant 5

Le résultat de la fonction ***checkSign*** est renvoyé au composant 5.

En renvoyant "Vrai", la transaction est vérifiée et correcte et dans le cas contraire la transaction n'est pas correcte.

## III. Description des erreurs

Chacune des erreurs du composant est gérée par les exceptions « throws » ci-dessous :

Etape 2

| **Nom Fonction :** string signData(String hash, String privateKey***)*** | |
|---|---|
| ***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*** |

Etape 4

| **Nom Fonction :** bool checkSign( String clePub, String sign, String hash) | |
|---|---|
| ***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*** |

## IV. Plan de tests

Les fonctions des différents composants sont testées lors de l'exécution de la fonction principale(main) afin de vérifier si le comportement du composant répond aux attentes.

Aucun message d'erreur ne sera affiché si le comportement du composant est correct et dans le cas contraire un message d'erreur sera indiqué.

- String signData (String hash, String privateKey*)*

Si l'un des paramètres (hash, privateKey) est nul alors le message d'erreur suivant est retourné :

« NULL VALUE » ;

Si la fonction est appelée sans aucun paramètre, le message suivant est retourné:
« NOT ENOUGH ARGUMENT »;

S'il y'a plus de 2 paramètres (hash, privateKey) alors le message d'erreur suivant doit être renvoyé :
« TOO MUCH ARGUMENT »;

Si la fonction est appelée avec un autre type que celui attendu, le message suivant est retourné :

« INVALID TYPE ARGUMENT » ;

- bool checkSign (String clePub, String sign, String hash)

Si l'un des paramètres (clePub, sign, hash) est nul alors le message d'erreur suivant doit être renvoyé :
« NULL VALUE »;

S’il y’a plus de 3 paramètres (clePub, sign, hash) alors le message d’erreur suivant doit être renvoyé :
« TOO MUCH ARGUMENT »;

S’il manque au moins un des paramètres(clePub, sign, hash) 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 »;