Projet Composant

TARSIMI Ayoub
BAYEUX Florian
AUSCHER Octave
EL WARDI Mohamed-Zakaria

Document de spécification du composant “Signature Numérique”

15 février 2018

Table des matières

1. Contexte 3

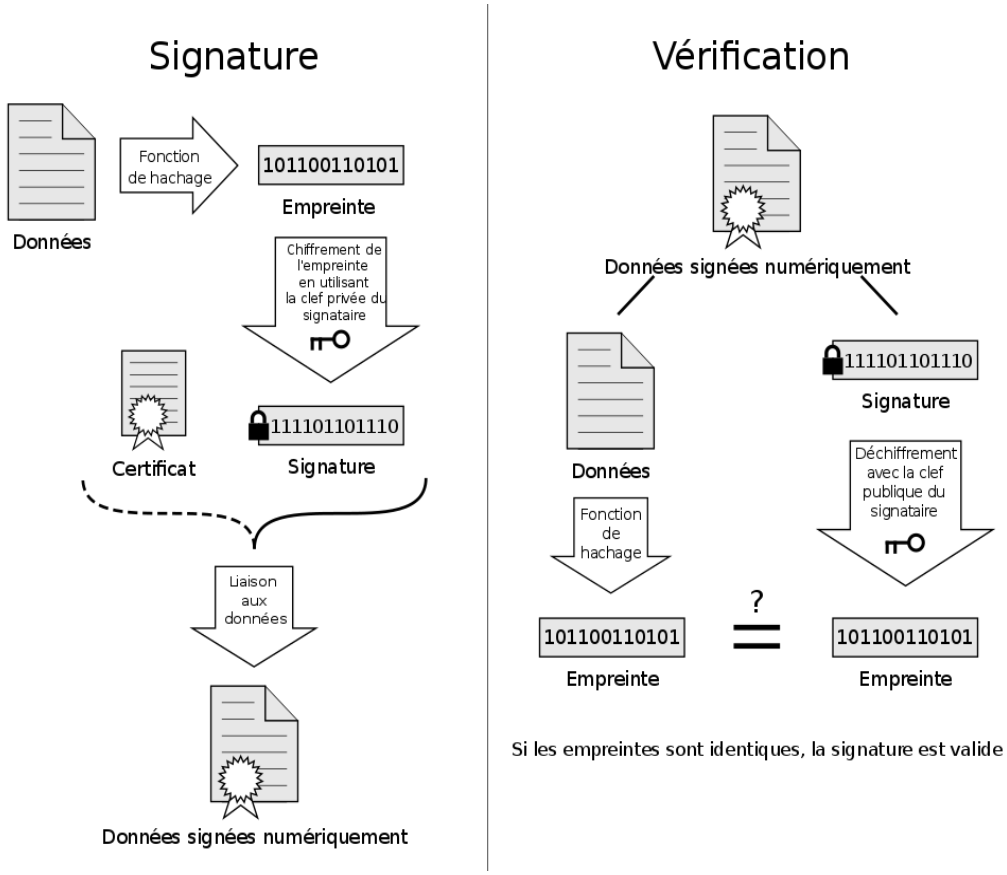
2. Schéma bloc 4

3. Les fonctions 5

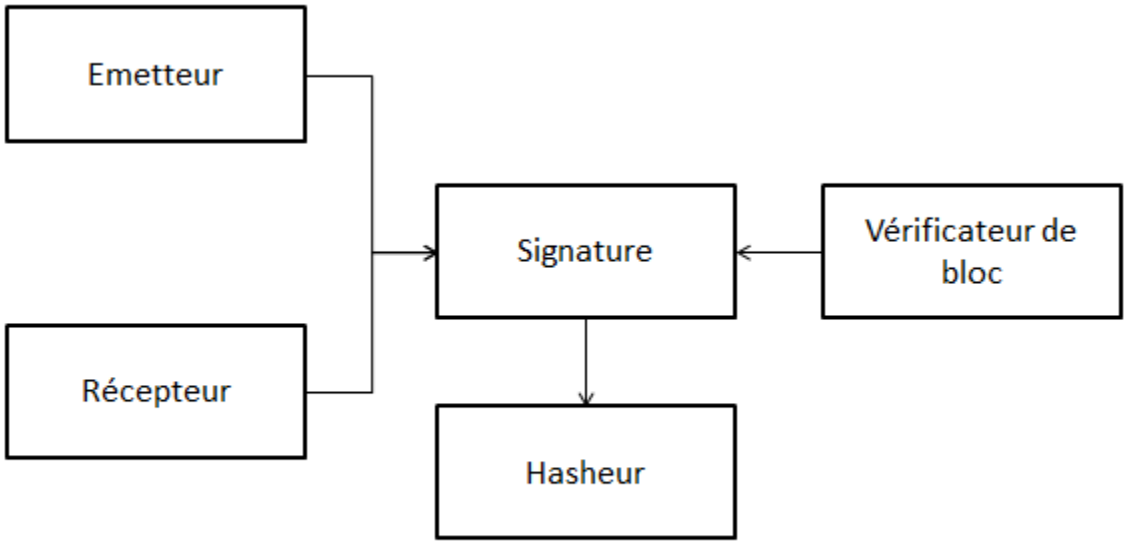
4. Tests 6

1. Contexte

La signature numérique permet au bénéficiaire de la transaction de s'assurer qu'elle provient bien d'un émetteur connu à l'avance. En d'autres termes, la signature numérique 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'emetteur, peut être certain de l'origine de la transaction. L'émetteur va hasher la transaction puis chiffrer le hashage obtenu avec une clé privée. Suite à cela, il envoie au bénéficiaire la transaction, le hashage correspondant ainsi que la clé publique liée à la clé privée. Le bénéficiaire déchiffre le transaction hashage de la transaction et s'assure ainsi de l'origine de celle-ci. Il compare à posteriori le hashage reçu avec le résultat du hashage de la transaction (non hashée) reçue.



2. Schéma bloc



Le composant Signature s'interface avec le hasheur qu'il va appeler pour valider une signature. Par ailleurs, le composant Signature s'interface d'une part et d'autre avec l'emetteur et d'autre part avec le bénéficiaire pour leur fournir une paire de clé (clé privée + clé publique). Enfin, le composant Signature est appelé par le vérificateur de bloc (qui contient en réalité une seule transaction) afin de valider ou non l'authenticité d'une transaction. Le vérificateur de bloc appelle la fonction validerSignature() du composant Signature qui renvoie true si le signataire de la transaction est bien l'émetteur, et false si le signataire de la transaction n'est pas l'émetteur.

3. Les fonctions

Le composant signature contient les fonctions suivantes :

String createPublicKey()

Cette fonction ne prend aucun paramètre en entrée, et doit renvoyer une chaîne de caractères (String) contenant une clé publique de 256 bits (64 octets, 64 caractères) générée grâce à l'algorithme ECDSA (Elliptic Curve Digital Signature Algorithm).

String createPrivateKey(String public_key)

Cette fonction prend en paramètre une chaine de caractères (String) contenant une clé publique de 256 bits (64 octets, 64 caractères) et doit retourner une chaine de caractères (String) contenant la clé privée de 256 bits (64 octets, 64 caractères) associée à la clé publique passée en paramètre.

Remarque : les fonctions createPublicKey() et createPrivateKey() servent uniquement à générer une paire de clé (clé publique, clé privée). L'utilisateur de ces deux fonctions doit impérativement penser à stocker les clés générées dans un fichier. La clé publique peut être communiqué à tous mais la clé privée est secrète.

Signature signMessage(String data, String private_key)

Cette fonction prend en paramètre une chaine de caractère correspondant à la transaction à signer ainsi qu'une clé privée de 256 bits (64 octets, 64 caractères) et 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. Cette fonction doit appeler le hasheur afin d'hasher la transaction avant de signer le hash obtenu.

Bool validateSignature(String data, String public_key, String signature)

Cette fonction 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 512 bits ainsi qu'un objet de type Signature correspondant à la signature qu'il faut authentifier. Cette fonction doit renvoyer une valeur booléenne (true ou false) correspondant à la validation ou non de la signature. Cette fonction fait

également appel au hasheur pour hasher la transaction passée en paramètre d'entrée. Si le hash de la transaction passée en paramètre d'entrée est égal au hash qui a été déchiffrée avec la clé publique donnée en paramètre, alors la fonction renvoie vraie. En effet, cela veut dire que l'auteur de la signature est bien l'émetteur de la transaction (ou du moins la personne dont la clé privée a été communiqué sur le canal). Sinon, elle renvoie faux.

4. Tests

On peut décomposer le composant Signature en deux parties :

- Génération de la paire de clé (clé publique, clé privée)
- Signature de la transaction et validation d'une signature

Comme vu pendant la séance, il est compliqué de tester le bon fonctionnement de la fonction de génération de la paire de clé (clé publique, clé privée) car elle faite à l'aide d'un algorithme de cryptographie elliptique dont nous connaissons mal le fonctionnement. En revanche, si les tests que nous allons présenter par la suite fonctionnent, alors nous pourrons affirmer que la fonction de génération de la paire de clé (clé publique, clé privée) fonctionne également.

Nous avons donc deux fonctions de test :

- testValidateSignature()
- testSignMessage()

La première fonction sert à tester le bon fonctionnement de la validation de Signature. On génère une paire de clé, on signe deux messages différent : chaine1 et chaine2. On teste deux cas différents :

validateSignature(« chaine1 », clé, « chaine1)

validateSignature(« chaine1 », clé, « chaine2 »)

Dans le premier cas, la fonction doit renvoyer vrai, dans le deuxième cas la fonction doit renvoyer faux.

La deuxième fonction sert à tester le bon fonctionnement de la signature d'un message. On génère deux clés différentes, on chiffre le même message et on regarde si les deux signatures sont égales ou non. La fonction renvoie faux si les deux signatures sont égales et vrai si les deux signatures sont différentes.