Version : 3

Document des spécifications

ECDSA signifie « Elliptic Curve Digital Signature Algorithm »

Réalisé par : Samar RHILANE – Luiz – Taoufik

Encadré par : M. Jose Luu

1- Description

1.1- Contexte

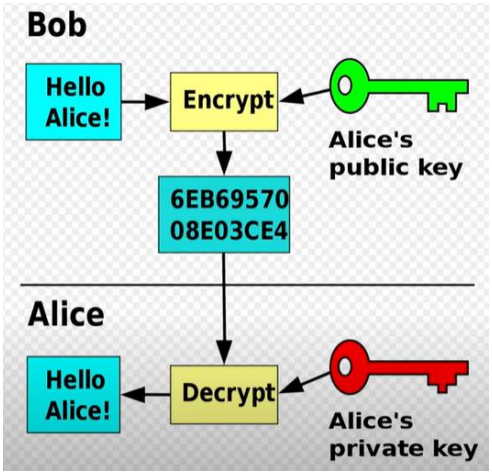
L'objectif du projet est de créer un composant ECDSA qui permet de générer une signature et de la valider

Qu'est-ce que l'ECDSA ?

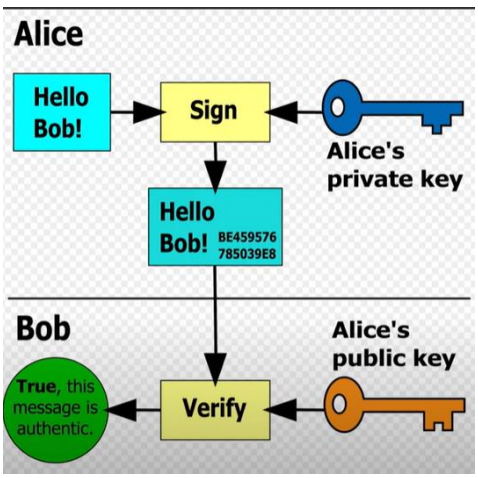
ECDSA : Algorithme de signature numérique à courbe elliptique est utilisé pour créer une signature numérique de données (un fichier par exemple) afin de nous permettre de vérifier leur authenticité sans compromettre leur sécurité. Pensons-nous comme une vraie signature, nous pouvons reconnaître la signature de quelqu'un, mais nous ne pouvons pas la falsifier sans que les autres le sachent. La différence entre une signature ECDSA et une signature réelle est qu'il est tout simplement impossible de falsifier la signature ECDSA.

La différence entre le cryptage/chiffrage et la signature

ECDSA ne crypte pas ou n'empêche pas quelqu'un de voir ou d'accéder aux données, cependant, il les protège, c'est de s'assurer que les données n'ont pas été falsifiées.



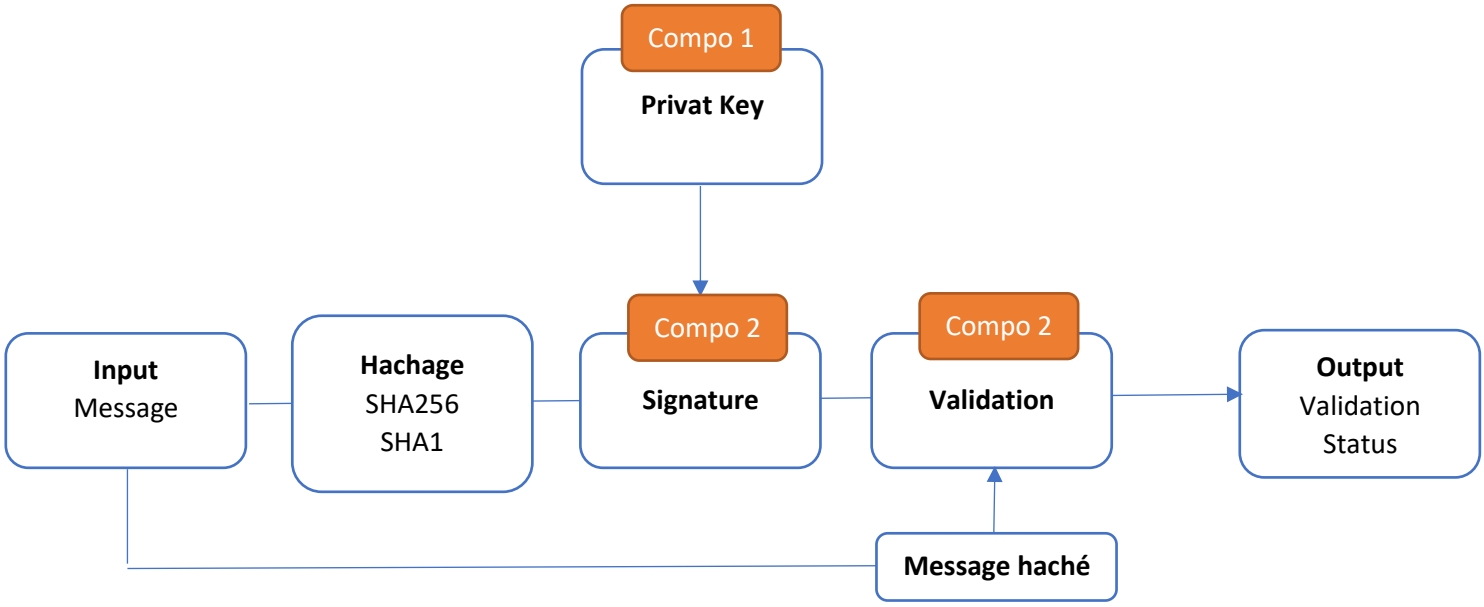
Cryptage et décryptage



Signature et vérification d'authenticité

Le but n'est pas de rendre le message super secret mais plutôt de vérifier que le message provient de la bonne destination

1.2- Schéma bloc incluant les composants connexes



1.3- Interface et interaction avec chaque autre composant

Le rôle principal du composant est de générer une signature ainsi de la valider, afin de réaliser ce composant, il est impératif d'utiliser un composant de cryptage ou bien dans notre cas nous avons appliquer une fonction de hachage prête à utiliser, que nous pouvons la changer avec un composant de hachage.

Ainsi un composant qui permet de générer la clé privée et la clé publique

Etapes de signature et validation

1) Appel du composant **composant_cle** :
   Le **composant_cl**e qui fait l'appel à une fonction **initilize**() qui permet d'initialiser une clé privée, ainsi le composant va générer les **Private_Key** et **Public_key**
2) Appel du composant principale **Signature_component** :
   - Le composant Signature va prendre en entrée le message ou fichier contenant les données,
   - Une fonction de hashage **SHA1** ou **SHA256** qui permet de hacher le message

- Une fonction **Message_Signature**() qui prend en paramètres le message et la clé privée, elle fait sortir une Signature
- Une fonction **Signature_Validation**(), qui prend en paramètres le message, la clé public, et la Signature, elle génère en sortie un message de validation si la signature est bonne, ou un message d'alerte si la signature n'est pas valide.

1.4- Résumé: déclarations de fonctions python d'interface et leurs arguments

Composant 1 : composant_cle

initialize(Number) : permet d'initialiser un nombre qui joue le rôle de la clé privé

getPrivateKey() : retourne la clé privée

getPublicKey() : retourne la clé publique

Composant 2 : Signature_compoenent

Message_Signature(string Hachage_message , string private_key) : permet de générer une signature à partir d'un message et une clé privée

La courbe choisie est paramétrée par **secp256k1** Normes pour une cryptographie efficace pour le Bitcoin

On utilise la fonction **uECC_sign()** qui prends en entrée le message hashé (SHA 2 est recommandée), la clé privée , la taille du message hashé en bits, la signature et le type de la courbe elliptique

En effet uECC nous fournit 2 type de signature le signature simple avec un module de hashage choisi, et une signature déterministique qui prends en entrée une fonction hash Contex qui permet de passer une fonction de hashaeg arbitraire à la fonction de signature déterministique

La fonction uECC_sign utiliste un générateur des valeur aleatoire

Signature_validation(string Hachage_message, string public_key, string _signature) :permet de vérifier la signature du message avec une clé publique

1.5- Cas d'erreurs

Erreur	Message d'erreur
Signature nulle	Signature is Empty
Longueur de signature n'est pas valide	Signature Length is not valid
Creation de signature	Signature has been created
Verification de la signature	Validation has been applied

Version : 3

2. Tests

`CreateSignatur(Hachage_message, Private_key) : verifier la creation de la signature`

`VerifySignature(Hachage_message, Public_key) :`

`VerifyLenSignature(Signature)`

`VerifyEmptySignature(Signature)`

Reference: Understanding How ECDSA Protects Your Data. : 15 Steps - Instructables