Version : 2

Document des spécifications

« Elliptic Curve Digital Signature Algorithm »

Réalisé par : Samar Rhilane – Luiz Motta Marchesini – Taoufik Haitam

1- Description

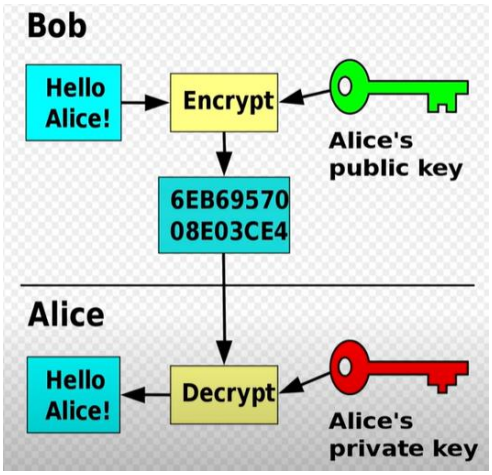
1.1- Contexte

Qu'est-ce que l'ECDSA ?

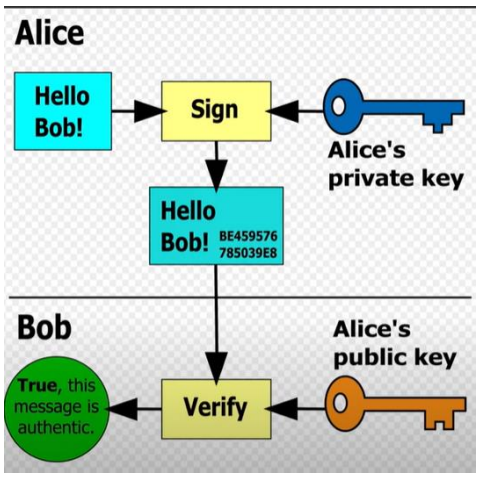
ECDSA est utilisé pour créer une signature numérique de données (un fichier par exemple) afin de nous permettre de vérifier leur authenticité sans compromettre leur sécurité. Pensons-nous comme une vraie signature, nous pouvons reconnaître la signature de quelqu'un, mais nous ne pouvons pas la falsifier 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.



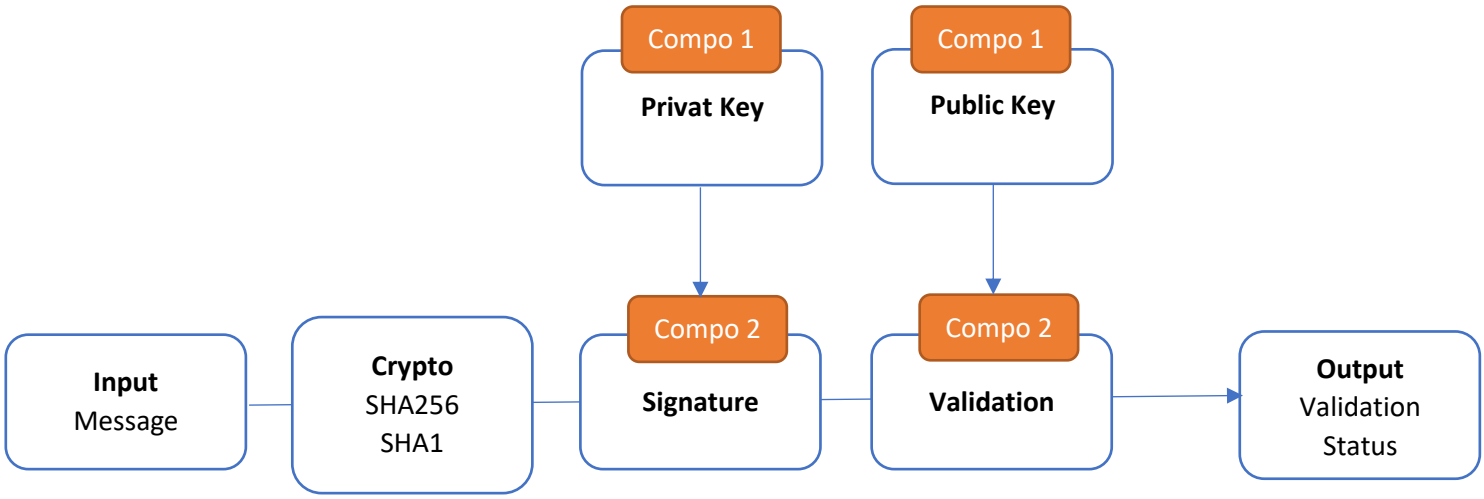
Cryptage et décryptage



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 cryptage **SHA1** ou **SHA256** qui permet de crypter 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.

Schéma d'interaction

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

hexchr2bin(const char hex) : permet de convertir un caractère hexadécimale en binaire

Ce calcul est basé sur les valeurs numériques de chaque caractère dans la table de codage de texte ASCII.

hexStringToBin(unsigned char* out, const char* hexPrivate) : permet de convertir une liste de caractère hexadécimale en binaire et en la retournant dans un tableau * out

binToHexString(char* out, const unsigned char* bin, size_t len) : permet de convertir un ensemble de caractère binaire en hexadécimale

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

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

hex_str_to_uint8(const char* string):permet de convertir une liste de caractère hexadécimale en unsigned integer de longueur 8 bits

uint8_to_hex_str(vector<uint8_t>& vecteur) : Permet de reverser un vecteur de unsigned Integer de longueur 8 bits en liste de caractères hexadécimale

SHA256(string message) : permet de crypter le message avec SHA256

SHA1(string message) : permet de crypter le message avec SHA1

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
Vérification de la validation du signature	Validation has been applied
Message nul	Message is Empty

2. Tests

`CreateSignatur(message, Private_key) : vérifier la création de la signature`

`VerifySignature(message, Public_key) : vérifier si une validation de signature a été appliquée`

`VerifyLenSignature(Signature) : vérifier la longueur de la signature`

`VerifyEmptySignature(Signature) : vérifier que la signature n’est pas vide`

`VerifyEmptySignature(message) : vérifier que le message n’est pas vide`

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