Spécifications Composant CKD

Réalisé par
Céline Beniddir
Lina Bennaidja
Chirel Cohen

Version 4 - mis à jour par: Chirel Cohen
-modification de l'explication et le code test

Sommaire:

1. Description des fonctionnalités du composant

Ce composant est défini par des fonctions nommées Child Key Derivation (CKD) Functions. En effet, celles-ci ont pour but de dériver des clés “enfants” à partir d’une clé parent. Les clés en question peuvent être privées ou publiques.
On note donc deux types de dérivations possibles et donc deux fonctions CKD.
ckdPub pour générer un CKD à partir d’une clé public
ckdPriv pour générer un CKD à partir d’une clé privé

pour notre composant, n est fixé à 4 , il s’agit de l’ordre de la courbe qui as été fixé dans l’algorithme de génération des clés - secp256k1

Nous avons 2 types de ckd:

- CKD privé également nommé CKD hardened
- CKD public également nommé CKD Normal

Ainsi, on peut définir 4 scénarios de génération de clés enfants :

- Clé parent privée à clé enfant privée
- Clé parent publique à clé enfant publique
- Clé parent privée à clé enfant publique
- Clé parent publique à clé enfant privée (Scénario impossible)

2. Étapes du processus de fonctionnement pour chaque scénarios

Clé parent privée -> Clé enfant privée

Nous utiliserons pour ce scénario la fonction ckdPriv
cette fonction a en paramètre:
k: la clé privé
c: le chaîne code
i:l'index number - pour réaliser ce scénario, ce nombre doit être >=2^31
Cette fonction calcule une clé enfant privée à partir d'une clé parente privée. Les étapes de cette génération de clé enfant sont les suivantes :

- Vérification de la valeur de i telle que i soit entre 2^31 et 2^32-1
- Division du résultat de HMACSHA512 en 2 séquences de 32 bytes, nommées IL et IR
- nous convertissons IL et IR en 256 bit via la fonction parse()
- La clé retournée est calculée telle que ki = $parse_{256}(I_L) + k_{par} \pmod n$
- La chaîne code retournée est IR
- Nous devons vérifier si le ckd est bien valide via la fonction isValid(parse256(ki)) qui vérifie si parce 256(ki)< n , n étant le modulo , n étant l'ordre de la courbe utilisé pour la génération des clés privés et public via l'algorithme secp256k1

Clé parent publique -> Clé enfant publique

Nous utiliserons pour ce scénario la fonction ckdPub
cette fonction a en paramètre:
k: la clé publique
c: le chaîne code
i:l'index number - pour réaliser ce scénario, ce nombre doit être entre 0 et 2^31-1
Cette fonction calcule une clé enfant publique à partir d'une clé parent publique. Les étapes de cette génération de clé enfant sont les suivantes :

- Vérification de la valeur de i telle que i est bien entre 0 et 2^31-1
- Division du résultat de HMACSHA512 en 2 séquences de 32 bytes, nommées IL et IR
- nous convertissons IL et IR en 256 bit via la fonction parse()
- La clé retournée est calculée telle que ki est le resultat de $parse_{256}(I_L) + K_{par}$ (ma clé)
- La chaîne code retournée est IR
- Nous devons vérifier si la ckd est bien valide via la fonction isValid(parse256(ki)) qui vérifie si parse256(ki)<n , n étant l'ordre de la courbe utilisé pour la génération des clés privés et public via l'algorithme secp256k1

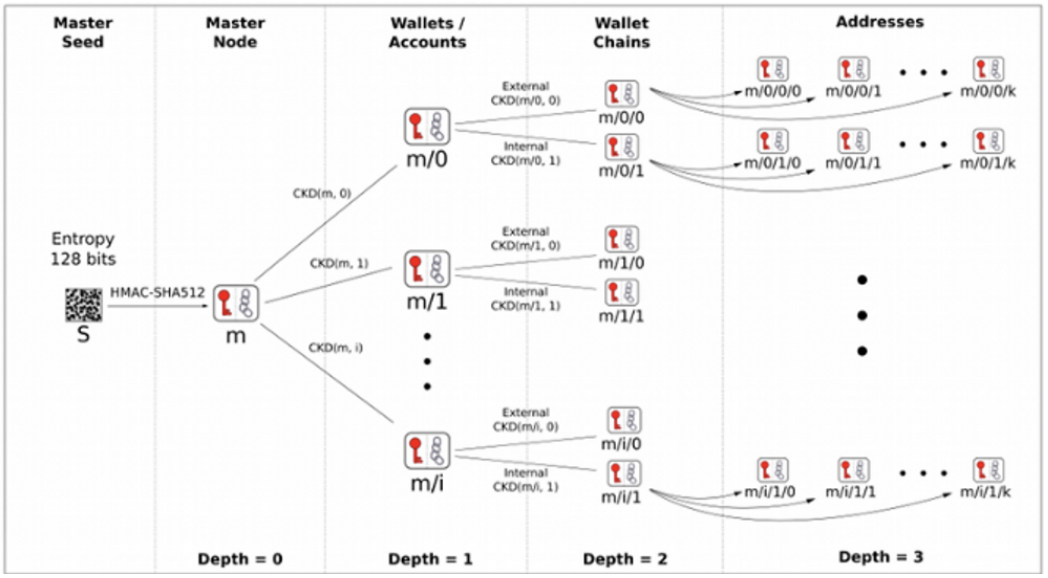
Clé parent privée -> Clé enfant publique
Nous utiliserons pour ce scénario la fonction ckdPriv

cette fonction a en paramètre:
k: la clé privé
c: le chaîne code
i:l'index number - pour réaliser ce scénario, ce nombre doit être entre 0 et 2^31-1
Cette fonction calcule une clé enfant privée à partir d'une clé parente privée. Les étapes de cette génération de clé enfant sont les suivantes :

- Vérification de la valeur de i telle que i bien entre 0 et 2^31-1
- Division du résultat de HMACSHA512 en 2 séquences de 32 bytes, nommées IL et IR
- Nous devons convertire la clé privée entré dans le paramètre en clé publique via une fonction N(k,c)
- nous convertissons IL et IR en 256 bit via la fonction parse()
- La clé retournée est calculée telle que ki = $parse_{256}(I_L) + k_{par}$ (ma clé)
- La chaîne code retournée est IR
- Nous devons vérifier si le ckd est bien valide via la fonction isValid(parse256(ki)) qui vérifie si parse(256(ki)< n , n étant l'ordre de la courbe utilisé pour la génération des clés privés et public via l'algorithme secp256k1

Il est ainsi possible de créer des clé enfants d'enfants en utilisant notre fonction
par exemple : ckdPriv(ckdPriv(k,c,i),i)

on peut créer un arbre de clé qui peut se visualiser comme ceci:



avec différents niveaux de dérivations

3. Spécification technique

L’objectif principal de ce document est alors d’expliquer les étapes de fonctionnement du composant CKD, et de définir les fonctions qui seront implémentées durant la phase de code du projet.

Les différentes fonctions à implémenter pour répondre au besoin
Les fonctions principales:

CKDpriv(k,c,i)	Array de 2 string	K :la clé privé C : chain code I : index number	Calcule la clé dérivatif child Hardened ou Normal et renvoie celle-ci avec la child chaîne code
CKDpub(k,c,i)	Array de 2 string	K :la clé publique C : chain code I : index number	Calcule la clé dérivatif child Normal et renvoie celle-ci avec la child chaîne code

Nous implémenterons également ces fonction qui seront utilisés dans les 2 fonctions principales

Fonction	Type Return	Paramètres	Rôle de la fonction
isHardened(i)	boolean	I : the index number	Renvoie 1 si i est entre 2^31 et 2^32-1 Sinon renvoie 0
split()	arrayListe of 2 string	Un string provenant de HMAC-SHA512	Sépare en 2 parties de 32 bit le résultat de HMAC-SHA512
parse256()	un string de 256 bit	un string de 32 bit	Convertit un élément de 32 bit en élément de 256 bit

N(c,k)	c:le chaine code k:la clé privé	retourne une clé public	convertit une clé privé en clé public
isValid(parse256(ki))	parse256(ki): il s'agit du résultat child key derivation qui est convertie en 256 bit via la fonction parse256	retourne un boolean	retourne 1 si le parametre est valide (<n) et retourne 0 si le paramètre est >= n

Aussi , nous utiliserons le code du composant 2 HMACSHA512 (chainecode, (cleprive+index))

4. Les tests

Voici comment notre code doit être tester:

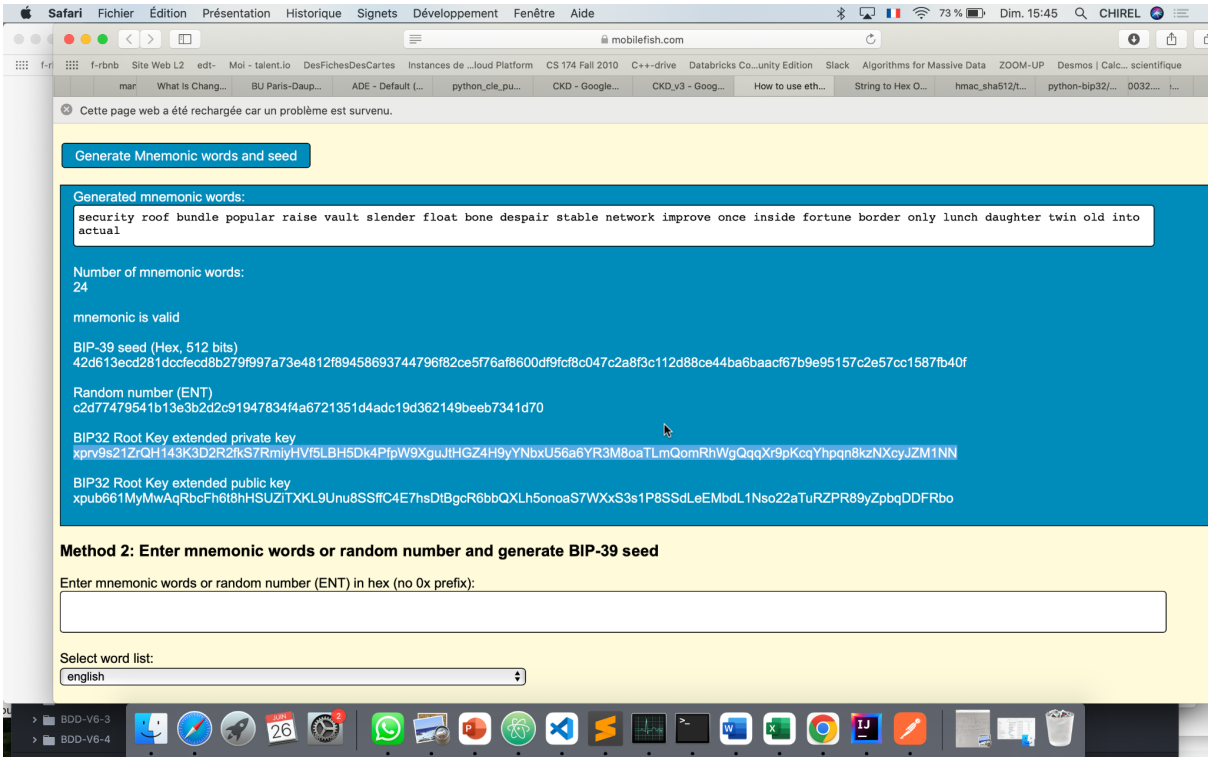
Dans un premier temps, voici une explication du code:

Utilisation de la première fonction CKDpriv(c,k,i)
générer une clé privée
#utiliser la fonction CKDpriv(c,k,i) pour générer le ckd
utiliser un i entre 2^31 et 2^32-1-> on aura en retour (ckd Hardened , IR(child code))
utiliser un i entre 0 et 2^31-> on aura en retour (ckd normal, IR(child code))
générer une clé public
#utiliser la fonction CKDpub(c,k,i) pour générer le ckd
utiliser un i entre 2^31 et 2^32-1-> on aura un retour erreur
utiliser un i entre 0 et 2^31-> on aura en retour (ckd normal, IR(child code))

Puis voici un exemple de code python que l'on peut utiliser pour tester notre composant:

Nous utilisons uniquement la generation de clé privé à clé enfant privé dans notre test:

Nous nous sommes servis d’un générateur de bip32:
https://www.mobilefish.com/download/ethereum/hd_wallet.html



42d613ecd281dccfecd8b279f997a73e4812f69458693744796f82ce5f7af8

Nous prenons uniquement le chaîne code qui est fournis dans ce cas par l’extraction de la partie gauche du Bip39seed

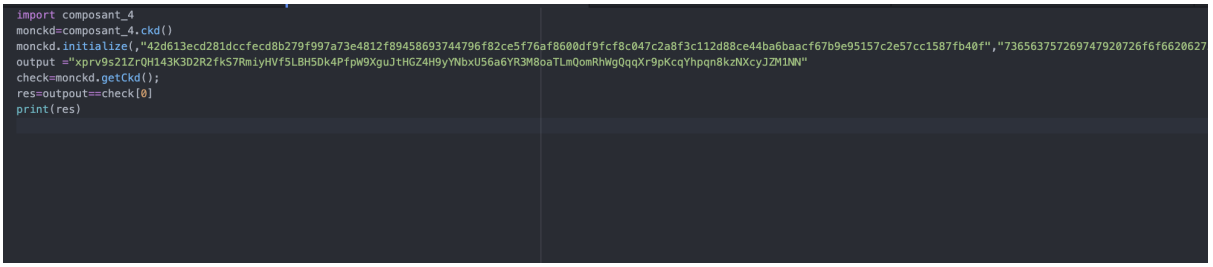
Code des tests:

```
import composant_4

monckd=composant_4.ckd()

monckd.initialize("42d613ecd281dccfecd8b279f997a73e4812f89458693744796f82ce5f76af8600df9fcf8c047c2a8f3c112d88ce44ba6baacf67b9e95157c2e57cc1587fb40f","600df9fcf8c047C2a8f3c112d88ce44ba6baacf67b9e95157c2e57cc1587fb40f",3*10**9)

output="xprv9s21ZrQH143K3D2R2fkS7RmiyHVf5LBH5Dk4PfpW9XguJtHGZ4H9yYNbxU56a6YR3M8oaTLmQomRhWgQqqXr9pKcaqYhpqn8kzNXcyJZM1NN"
check=monckd.getCkd();
res=outpout==check[0]
print(res)
```


import composant_4
monckd=composant_4.ckd()
monckd.initialize(,"42d613ecd281dccfecd8b279f997a73e4812f894586937447 96f82ce5f76af8600df9fcf8c047c2a8f3c112d88ce44ba6baacf67b9e95157c2e57cc1587fb40f","736563757269747920726f6f6620627
output ="xprv9s21ZrQH143K3D2R2fkS7RmiyHVf5LBH5Dk4PfpW9XguJtHGZ4H9yYNbxU56a6YR3M8oaTLmQomRhWgQqqXr9pKcqYhpqn8kzNXcyJZM1NN"
check=monckd.getCkd();
res=outpout==check[0]
print(res)