Projet
Programmation par Composant

HMAC – SHA512

Auteurs :

- DOUR Marcellino
- PARDINI Raphaël

Historique des versions :

Version	Description	Date
1.0	Création du document	15/06/2022
1.1	Changement de l'interface	21/06/2022

I – DESCRIPTION

A. Contexte

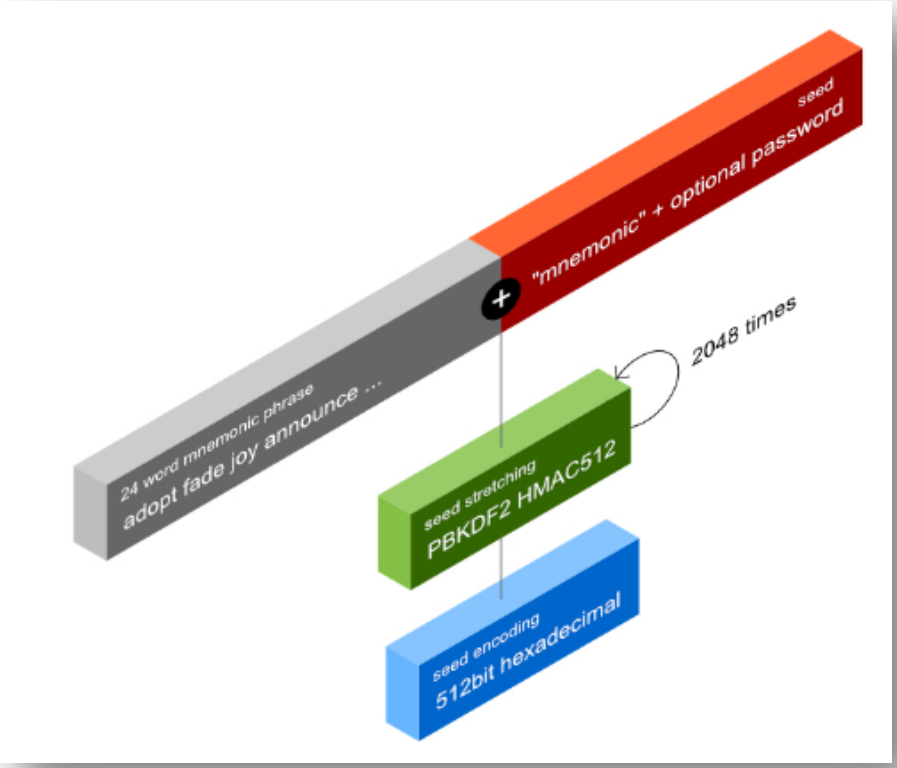
La création de ce composant intervient dans le cadre de notre cours « *Programmation par composant* », enseigné par Monsieur José LUU à l'Université Paris Dauphine. Le but du projet est de développer tous les composant d'une blockchain.

Dans notre groupe, nous allons implémenter la partie HMAC – SHA512.
C'est un type de code d'authentification de message, calculé en utilisant une fonction de hachage cryptographique en combinaison avec une clé secrète. Comme avec n'importe quel MAC, il peut être utilisé pour vérifier simultanément l'intégrité de données et l'authenticité d'un message. N'importe quelle fonction itérative de hachage, comme SHA-256 ou SHA-512, peut être utilisée dans le calcul d'un HMAC ; le nom de l'algorithme résultant est HMAC-SHA-256 ou HMAC-SHA-512. La qualité cryptographique du HMAC dépend de la qualité cryptographique de la fonction de hachage et de la taille et la qualité de la clé.

B. Interface et interaction avec chaque autre composant

Cette clé, composé de 256 bits, sera fourni par le composant 1 : BIP39. Puis, le composant 3, PKDF hmac sha512, complètera cette clé avec une clé secrète de 256 bits également.
La seed généré par la fonction de hachage sur cet input sera haché à nouveau, et cela 2048 fois.

Voici un schéma représentant les différents composants connexes et leurs interactions :



Dans notre composant, nous avons une fonction d'interface. Elle est nommée **hmac_sha512()**.

Cette fonction prend en entrée un hexadécimal de 512 bits sous la forme d'une chaine de caractère, et renvoie en sortie un hexadécimal de 512 bits sous la forme d'un **std:string**.

Appel depuis Python

Pour appeler cette fonction depuis Python, il faut taper les commandes suivantes :

```
>>> import hmac_sha512_component
>>> output = hmac_sha512_component.hmac_sha512(input)
```

Appel depuis C++

En préambule, la bibliothèque hmac_sha512 doit être importé :

```
>>include hmac_sha512_component.hpp
```

Pour appeler cette fonction depuis C++, voici la structure à respecter :

```
output = hmac_sha512(input)
```

avec :

- output : de type string
- intput : de type char*

C. Cas d'erreurs

Notre fonction peut calculer un hachage quelques soit l'argument passé en entrée. Ainsi, nous n'avons pas de gestion d'erreur à gérer, car notre méthode ne peut pas rencontrer d'erreur d'exécution lié à un mauvais paramètre.

II – Tests

A. Plans de tests

Afin de tester notre composant, nous avons choisi d'utiliser le plan de test suivant :

- Nous avons généré deux seed hexadécimal de 512 bits.
- Puis, nous avons utilisé la fonction HMAC-SHA512 de notre composant, et comparé les résultats avec la fonction d'un programme en ligne.

Nous avons choisi les programmes en ligne suivant :

- BIP39 → https://iancoleman.io/bip39/#french
- HMAC-SHA512 → https://emn178.github.io/online-tools/sha512.html

Nous allons appliquer ce plan de test depuis un appel python, ainsi que depuis un appel C++.

B. Programme de tests

Dans notre programme de test, nous avons stocker une liste de couple input/output. Puis, nous appliquons notre fonction sur chacun de ces inputs, et vérifier que l'output obtenu est correct.

De plus, nous vérifierons que lorsque la clé en entré est incorrect, le programme renvoie bien une exception.

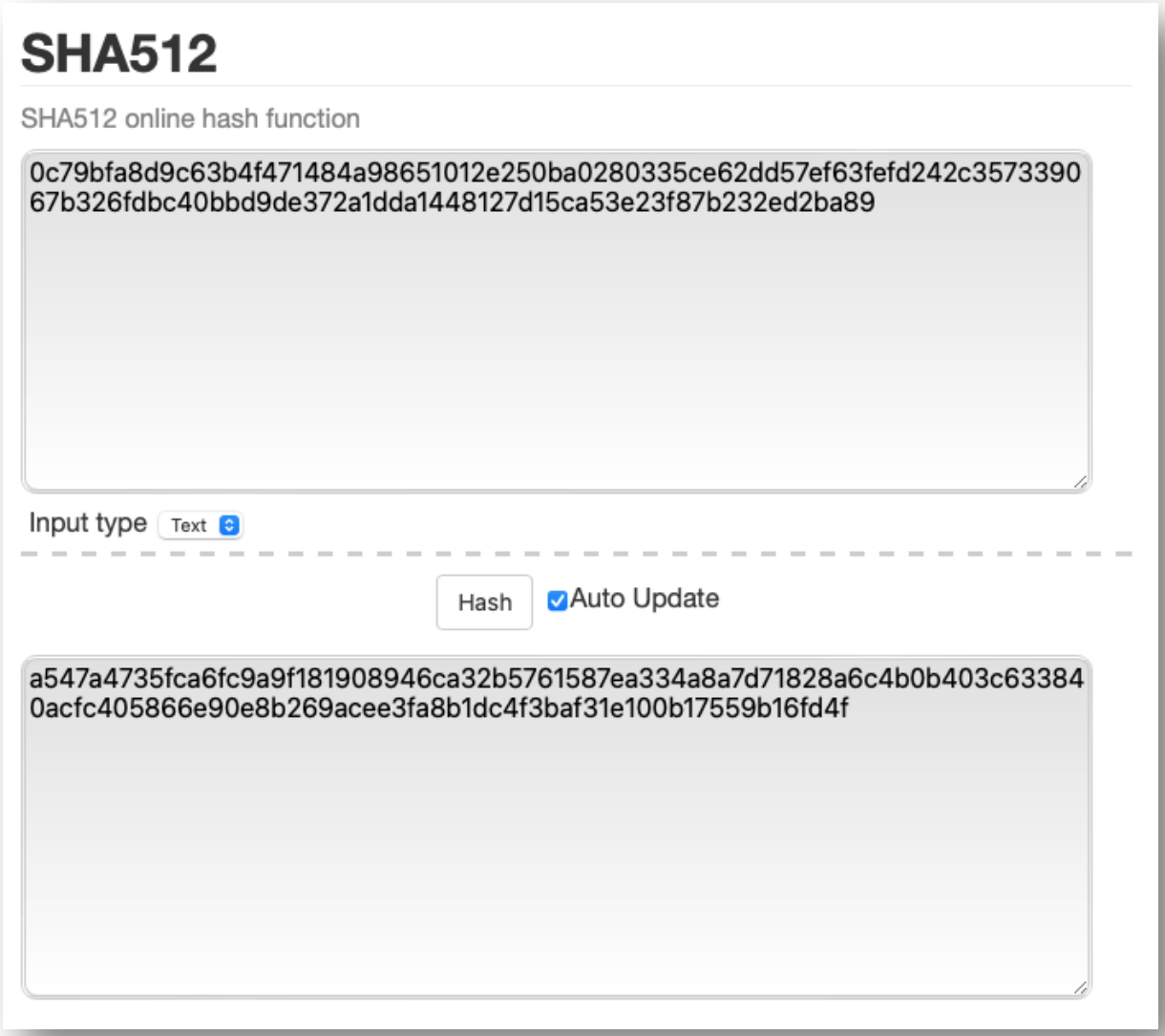
Pour exécuter les tests **d'un appel Python**, il faut renter la commande suivante dans le terminal :

```
>> python test_component.py
```

Le programme renvoie ***True*** si le composant fonctionne bien, et ***False*** sinon.

Voici des captures d'écran de l'application de ces tests :

Capture d'écran d'un couple input/output via un programme en ligne



Capture d'écran des résultats de notre programme test python avec les mêmes paramètres

```
  GNU nano 5.4                          test_component.py
import hmac_sha512_component

input = ["9b3fea674ae24cc4a0b8fd41c35d8d4ee1739d7ff9ce100ca4acf595a56a839a3df422726e48103c7d1ce>
output = ["0c79bfa8d9c63b4f471484a98651012e250ba0280335ce62dd57ef63fefd242c357339067b326fdbc40b>
correct = True

for i in range (len(input)):
    if output[i] != hmac_sha512_component.hmac_sha512(input[i]):
        correct = False

print(correct)
```

Le fichier test s’exécute correctement.

```
marcellinodour@instance-1:~/hmac_sha512/hmac_sha512_component$ python test_component.py
True
```

De même, nous avons effectué un fichier test pour **un appel C++** :

```
  GNU nano 5.4                          test_component.cpp
#include "hmac_sha512_component.hpp"
#include <iostream>

using namespace std;

int main(){

    char input[2][256] = {"9b3fea674ae24cc4a0b8fd41c35d8d4ee1739d7ff9ce100ca4acf595a56a839a3df4
    string output[2] = {"0c79bfa8d9c63b4f471484a98651012e250ba0280335ce62dd57ef63fefd242c357339
    bool result = true;

    for(int i = 0; i < sizeof(input)/sizeof(input[0]); i++){
        if(hmac_sha512(input[i]).compare(output[i])!=0){
                result = false;
        }
    }

    if(result){
        cout << "Test succes" << endl;
    }else {
        cout << "Test failed" << endl;
    }

    return 0;
}
```

Le fichier test s’exécute correctement. (Test succes ou Test failed)

```
marcellinodour@instance-1:~/hmac_sha512/hmac_sha512_component$ ./test_component
Test succes
```