Annexe

Transactions avec le *BNB Token* sur la *BinanceSmartChain* en tant qu'example de transfert de propriété sur les réseaux décentralisés et d'utilisation de *clé publique* et *clé privée*

Pour pouvoir interagir avec la *BinanceSmartChain* (BSC) via un script *Python*, on utilise la bibliothèque *web3* qui s'installe comme suit (doit être exécutée sur la console ou le terminal ; si elle est déjà installée, cette étape n'est pas nécessaire) :

[ ]:
```
#pip install web3
```

L'étape suivante consiste à importer *web3* dans le fichier de *Python*. *Web3* est ainsi la "glue" entre le script et la *BinanceSmartChain*, notre blockchain ou le *Distributed Ledger Technology* (DLT) en question. Nous utiliserons la BSC et son token, le *BNB Token*, pour démontrer le transfert de propriété selon une "Token-Based Approach" et l'utilisation de **clés publiques** et **privées**, comme décrit dans le dossier. Il est important de savoir que le BSC est une copie exacte d'*Etherium*, c'est pourquoi la bibliothèque *web3* développée pour *Etherium* peut également être utilisée ici. En outre, il est possible d'importer d'autre module qui peuvent être utiles aux commances émergentes dans cette démonstration de transfert de propriété.

[1]:
```
from web3 import Web3
import json
import time
import config
```

Le module *web3* installé nous permet d'interagir avec la BSC. Tout d'abord, une variable doit être configurée avec le "endpoint" de la SmartChain afin de pouvoir créer une variable *web3* par la suite.

[ ]:
```
bsc = "https://bsc-dataseed.binance.org/"
web3 = Web3(Web3.HTTPProvider(bsc))

print(web3.isConnected())
```

Une fois cette étape terminée, il est possible d'obtenir des informations sur la blockchain et les transactions qui y sont effectuées. Par example, si on veut savoir combien de token contient mon compte, on peut procéder comme suit : * Créer une variable **account_1**, étant l'adresse dont nous voulons connaître le solde. On définit account_1 via la *clé publique* de cette adresse. L'adresse publique est accessible à tous. * Le montant que nous obtenons affiché par défaut est en "wei", qui est la plus petite dénomination de crypto-monnaie (la conversion exacte des dénominations de monnaie peut être trouvée sur https://web3py.readthedocs.io/en/stable/examples.html#converting-currency-denominations). Par conséquent, nous convertissons les *wei* en *ether* (rappelez-vous: *BNB* est une copie exacte d'Etherium), qui est une dénomination avec 18 décimales, pour voir le monant réel sur le compte.

[ ]:
```
account_1 = "Insérer une adresse publique account_1"

balance = web3.eth.get_balance(account_1)
```

```
print(balance)

humanReadable = web3.fromWei(balance, 'ether')
print(humanReadable)

nonce = web3.eth.get_transaction_count(account_1)
```

Puisqu'une transaction implique toujours deux parties, une deuxième variable de compte (le compte destinaire) **account_2** doit être définie avec sa *clé publique* correspondante, qui reçoit la transaction de **account_1**. Comme mentionné dans le dossier, l'adresse de l'expéditeur ainsi que celle du destinaire sont connues par la *clé publique*. Pour qu'il soit possible d'envoyer une transaction à partir du compte de l'expéditeur, nous avons également besoin de sa *clé privée*, qui peut être définie par la une variable "private" dans un script séparé (la *clé privée* doit toujours rester secrète). Comme décrit dans le dossier, la propriété, et la possibilité de transférer la propriété, est réalisée par cette paire de *clés publiques* et *privées* qui sont utilisées pour créer et vérifier les transactions souhaitées, agissant ainsi comme des empreintes digitales électroniques. Cependant, dans ce *Jupyter Notebook*, je ne publie pas ma *clé privée* personelle, mais j'utilise une variable ficitve.

[ ]:
```
account_2 = "Insérer une adresse publique account_2"
balance_2 = web3.eth.get_balance(account_2)
humanReadable = web3.fromWei(balance_2, 'ether')
print(humanReadable)

private = "privatekey from account_1"
```

La transaction concrète sur la *BinanceSmartChain* se compose de **trois étapes**: la *création d'un objet de transaction*, la *signature de la transaction avec la clé privée* et l'*envoi effectif de la transaction*.

1. Création de l'objet de transaction: * Tout d'abord, une variable **nonce** doit être définie pour chaque transaction. **Nonce** est un nombre qui n'est utilisé qu'une seule fois, et qui fait référence au premier nombre qu'un "mineur" de blockchain doit décrouvrir avant de "résoudre" un bloc dans la blockchain. Plus simplement, le **nonce** est le nombre de transaction de l'adresse de l'expéditeur. * Il faut ensuite préciser où nous voulons envoyer notre transaction. Cela doit inclure la valeur que nous voulons envoyer du **account_1** au **account_2**. Cette valeur doit être convertie en *wei* au préalable, sinon nous enverrions le monant desiré en *ether* (ce qui serait très cher !!!!). * Définir le **gas** and le **gasPrice**: * Le **gas** est le "fuel" dans le réseau *Ehtereum*. Les commandes qui exécutent la machine virtuelle sont payées avec du **gas**. Donc, si la transaction doit être validée rapidement, il est logique de payer un prix du **gas** plus élevé afine que les validateurs (*mineurs*) soient incités à valider la transaction. Le mécanisme du **gas** et du **gasPrice** est important ici car il garantit que les frais sont calculés de manière équitable et appropriée. * Nous utilisons un **gas** par défaut de 21000. Le **gasPrice** réel est fixé à 50 et converti en "gwei", à nouveau un dénominatuer.

[ ]:
```
transaction = {
    'nonce': nonce,
    'to': account_2,
    'value': web3.toWei(0.001, 'ether'),
    'gas': 21000,
    'gasPrice': web3.toWei('50', 'gwei')
    }
```

2. Signer la transaction

La signature de la transaction dans ce contexte signifie autant que la vérification de la transaction avec la *clé privée*.

[ ]:
```
signed_transaction = web3.eth.account.signTransaction(transaction, private)
```

3. Envoyer la transaction * Pour compléter les trois étapes et finaliser la transaction, il faut maintenant créer une variable qui nous donne le **transaction hash**. Le *hash* est, pour ainsi dire, le code pour l'identification unique d'une certaine transaction, avec lequel il est possible de savoir si et comment la transaction a été effectuée. * Ici, nous envoyons une *raw transaction*, c'est-à-dire une transaction avec la signature attachée. * Pour pouver consulter le *hash*, la transaction doit finalement être convertie en nombre hexadécimaux.

[ ]:
```
transaction_hash = web3.eth.sendRawTransaction(signed_transaction.rawTransaction)

print(web3.toHex(transaction_hash))
```

Si aucune erreur n'a été faite, la transaction devrait avoir été effectuée avec succès et nous recevons un *transaction hash* qui reflète la transaction effectuée Pour prouver que ce script a fonctionné, la transaction peut être vérifiée à l'aide du *hash* sur la page https://bscscan.com/ ou directement dans le script *Python* comme suit :

[ ]:
```
trans = web3.toHex(transaction_hash)
time.sleep(10)
tracing_transaction = web3.eth.get_transaction(trans)
print(tracing_transaction)
balance = web3.eth.get_balance(account_1)
humanReadable = web3.fromWei(balance, 'ether')
print(humanReadable)
```