

BLOCKCHAIN DANS L'ADMINISTRATION FONCIÈRE

FAIT PAR
MOUSTAPHA ADRIEN MBOUMBA
SERIGNE FALLOU NDIAYE
SOKHNA OUMOU WADE
MARIÈME BOUSSO
AISSATA NDIAYE

DANS LE CADRE DU MODULE D'INGÉNIERIE CRYPTO/BLOCKCHAIN

DISPENSÉ PAR M.MENDY

ANNÉE:2022-2023

NIVEAU:M2 GLSI/SRT

INTRODUCTION

La blockchain est un registre partagé et immuable pour enregistrer les transactions, suivre les actifs et instaurer la confiance. C'est une technologie utilisée dans de nombreux secteurs notamment celui de l'administration foncière. Ce domaine est sujet à de nombreux litiges dans les tribunaux. C'est un domaine transversal en plus du code des collectivités locales on retrouve également le code forestier, le code minier, la charte du domaine irrigué, etc. Cette pluralité de régime et de code d'administration contribue à l'opacité qui entoure la gouvernance foncière, caractérisée par une perte de confiance de la société en l'autorité. Ce manque de transparence facilite la pullulation des conflits, et complique l'accès à la terre à toutes les catégories sociales que ce soit homme, femme ou jeune. Tous ces facteurs motivent l'utilisation de la blockchain dans l'administration foncière qui permettra la création d'un registre immuable des propriétaires, qui facilitera l'enregistrement des transactions telles que les ventes, les transferts de successions et les hypothèques.

C'est ainsi qu'il nous a été confié dans ce projet de définir la propriété partielle d'un terrain dans un système de gestion foncière basé sur la blockchain. Dans un premier temps, nous allons faire l'étude d'un système blockchain de consortium (cas de Quorum), ensuite étudier les signatures multiples et enfin essayer de faire la multisignature de document et validation par smart contrat (Cas de Quorum).

A1. Etude du sujet

L'administration foncière est généralement gérée à l'aide de systèmes centralisés, tels que des bases de données gouvernementales et des enregistrements papier. Les transactions immobilières sont généralement validées par des intermédiaires tels que des avocats ou des notaires, qui vérifient les dossiers de propriété et les transactions pour s'assurer de leur validité. Cependant, ce système peut être sujet à des erreurs, à des fraudes et à des retards en raison de la complexité des processus administratifs. De plus, l'accès aux informations peut être limité, ce qui peut entraver la transparence et la facilité de vérification des dossiers de propriété.

La blockchain est une technologie de stockage et de transmission décentralisée d'informations, souvent utilisée pour enregistrer les transactions de cryptocurrencies comme Bitcoin. Les données sont enregistrées dans de nombreux ordinateurs (nœuds) au lieu d'être stockées centralement sur un serveur, ce qui la rend plus sûre et plus transparente. Les enregistrements sont liés et sécurisés par la cryptographie, ce qui en fait une base de données fiable et permanente, difficile à falsifier ou à altérer. La blockchain peut également être utilisée pour stocker et gérer d'autres types de données, telles que des contrats numériques, des identités numériques, etc.

L'utilisation de cette technologie peut apporter plusieurs avantages dans l'administration foncière :

- Elle peut permettre une vérification transparente des informations sur les propriétés immobilières et les transactions, ce qui peut réduire les fraudes et les erreurs.
- Elle offre une sécurité accrue des données grâce à la cryptographie et à la décentralisation des enregistrements.
- Elle peut automatiser et accélérer les processus administratifs, tels que la vérification de la propriété et la validation des transactions immobilières.
- Les enregistrements sur la blockchain sont immuables, ce qui signifie qu'ils ne peuvent pas être modifiés ou supprimés, ce qui peut garantir la validité des informations.

2. Etude d'un système blockchain de consortium(cas de Quorum)

Pour déployer un réseau et effectuer une transaction sur Quorum, on a besoin de:

-Truffle: permet de développer des smart contracts et de les déployer sur une blockchain de test ou sur une blockchain spécifique.

-Ganache: outil permettant de simuler une blockchain en local afin de développer les contrats sur la blockchain Ethereum.

-Metamask: agit comme un portefeuille de crypto-monnaies pour stocker, acheter, vendre ou échanger des jetons basés sur Ethereum directement depuis le navigateur. Il se connecte à la blockchain Ethereum.

-Web3js: une collection de bibliothèques qui vous permettent d'interagir avec un nœud Ethereum local ou distant en utilisant HTTP, IPC ou WebSocket.

Pour ce faire, on a installé Truffle à partir de npm avec la commande `npm install -g truffle`.

Une fois Truffle, on crée un projet avec la commande `truffle init`.

```
C:\Users\HP\Desktop\truffle>truffle init

Starting init...
=====





> Copying project files to C:\Users\HP\Desktop\truffle

Init successful, sweet!

Try our scaffold commands to get started:
  $ truffle create contract YourContractName # scaffold a contract
  $ truffle create test YourTestName         # scaffold a test

http://trufflesuite.com/docs
```

On note la création de ces fichiers ci -dessous:

 contracts	30/01/2023 11:48	Dossier de fichiers
 migrations	30/01/2023 11:54	Dossier de fichiers
 test	30/01/2023 11:46	Dossier de fichiers
 truffle-config	30/01/2023 12:39	Fichier source Jav...

Nous allons éditer le fichier truffle-config.js pour configurer le réseau. En effet, on va spécifier l'hôte, le numéro de port et l'identifiant du réseau. (Voir dans le dossier sur les fichiers de configuration).

Nous allons créer un contrat. Ce contrat aura pour extension sol c'est-à-dire il utilise le langage Solidity. Le langage Solidity permet d'écrire des "smart contract". Un "smart contract" est un contrat qui s'appuie sur une technologie blockchain pour garantir leur intégrité et leur inviolabilité. L'appellation du smart contract vient de sa capacité à exécuter automatiquement des instructions prédéfinies.

Pour ce faire, nous allons lancer la commande:

```
C:\Users\HP\Desktop\truffle>truffle create contract Foncier
```

Le fichier Foncier sera créé au niveau du dossier "contracts". Nous allons éditer le fichier Foncier.sol et y écrire un code permettant de gérer les titres fonciers. (Voir dans les fichiers de configuration).

Une fois, le smart contract créé, nous allons créer un fichier migration qui nous permet de déployer le smart contract via Ganache. Le fichier se crée de cette manière:

```
C:\Users\HP\Desktop\truffle>truffle create migration Foncier
```

Une fois, le fichier édité, nous allons déployer notre "smart contract" à travers la commande:

```
C:\Users\HP\Desktop\truffle>truffle migrate  
Compiling your contracts...  
=====
```

Nous allons ouvrir Ganache et voir que le "smart contract" a été bien déployé. Et on note ainsi la création du 1er réseau.

CURRENT BLOCK 1	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MUIRGLACIER	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING
BLOCK 1	MINED ON 2023-01-30 12:08:47				GAS USED 67066	

Nous avons besoin de la clé privée pour pouvoir effectuer une transaction au niveau de la portefeuille.

ACCOUNT INFORMATION

ACCOUNT ADDRESS

0x5f8f969c5FC71Dd1F83F3b480C4dbaD0a6308D6e

PRIVATE KEY

d80f363f8a8c3ab69ff85b831c540ce4cec3845515eaf60cf417f7b4be71b6eb

Do not use this private key on a public blockchain; use it for development purposes only!

DONE

Ensuite, nous allons relier Ganache à Metamask qui sera notre portefeuille de crypto-monnaies pour effectuer des transactions.

Au niveau de MetaMask, nous allons créer un réseau en se basant sur les informations fournies par Ganache.

Network name


New RPC URL

Chain ID ⓘ


Currency symbol

Block explorer URL (Optional)

0x5f...



Account 2 ✎



0x5f8f969c5FC71Dd1F83F3b480C4db
aD0a6308D6e

Et on voit qu'une transaction a été effectuée.

GAS USED
67066

1 TRANSACTION

3. Étude des concepts de signature multiple

- Signature multiple d'un document

La signature multiple d'un document sur la blockchain est un processus qui permet à plusieurs parties de signer un document numérique en utilisant la technologie de la blockchain. Ce processus est souvent utilisé pour valider les transactions financières ou les contrats intelligents.

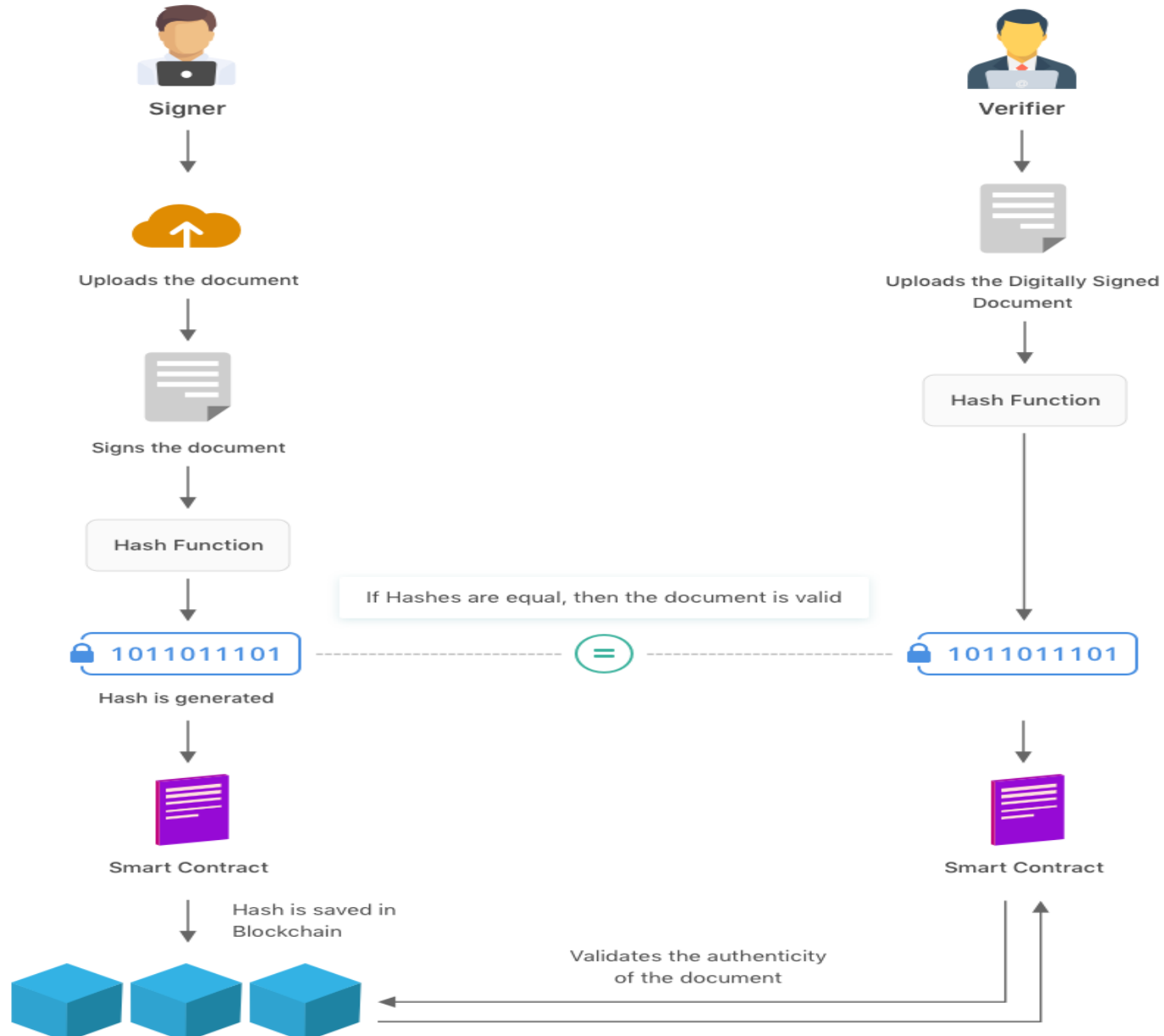
Voici comment une plateforme de signature de documents en blockchain pourrait fonctionner

La plateforme de signature de documents blockchain devra permettre aux utilisateurs de s'inscrire en fournissant une identité approuvée par le gouvernement. Après l'inscription, les utilisateurs peuvent télécharger le document à signer sous forme de pdf ou doc, ajouter les signataires et stocker les documents signés enregistrés dans la blockchain. Les signataires recevront une demande de signature par email et pourront signer en utilisant une signature dessinée ou un système de signature approuvé par le gouvernement. Les transactions de signatures sont enregistrées sur la blockchain et conservent l'historique des signatures non modifiables. Les utilisateurs peuvent également ajouter des spectateurs qui recevront le document en copie.

Lorsqu'un document est soumis pour une signature multiple, chaque partie impliquée peut utiliser une clé privée pour signer le document. Une fois que toutes les signatures requises sont incluses, le document est considéré comme signé et la transaction peut être finalisée.

L'avantage de la signature multiple sur la blockchain est que la sécurité de la transaction est assurée par la blockchain elle-même, ce qui empêche toute modification frauduleuse du document une fois qu'il a été signé. De plus, la blockchain peut automatiquement vérifier que toutes les signatures requises sont présentes avant de finaliser la transaction, ce qui peut économiser du temps et des ressources.

Processus de création de la signature et la signature par les utilisateurs



Un document est haché et le hachage est stocké sur la blockchain.

Chaque signataire génère une paire de clés publique-privée. Les clés publiques sont stockées sur la blockchain, et les clés privées sont gardées secrètes par chaque signataire.

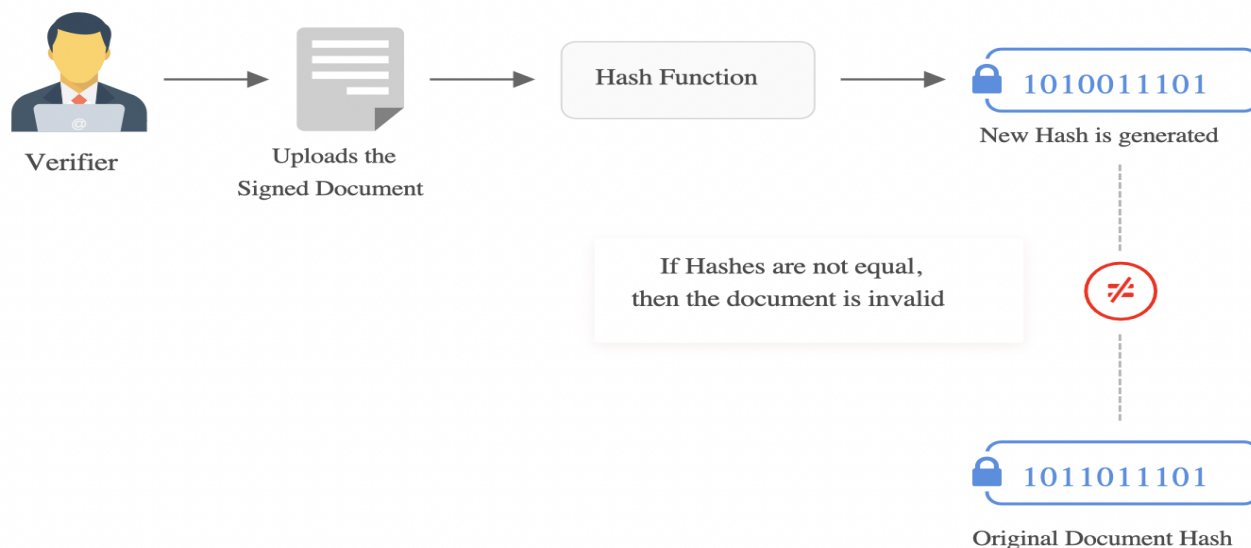
Le document est partagé avec tous les signataires, qui peuvent ensuite le signer en utilisant leur clé privée. La signature est créée en chiffrant le hachage du document avec la clé privée.

Chaque signataire envoie ensuite sa signature à un référentiel central, qui peut être un contrat intelligent sur la blockchain.

Le contrat intelligent vérifie si un nombre minimum de signatures a été reçu, et si c'est le cas, le document est considéré comme signé. Le contrat intelligent vérifie également si l'une des signatures n'est pas valide, par exemple si elle a été créée en utilisant un document différent ou une clé privée différente.

Le document signé et ses signatures sont stockés sur la blockchain à des fins d'immuabilité et d'auditabilité.

Authentification et validation du document



Une fois le document signé par toutes les entités, un hash est généré sur la blockchain. Tout utilisateur disposant du document signé peut vérifier l'authenticité du document sur la plateforme.

Les utilisateurs peuvent télécharger le document signé sur la plateforme de signature de documents blockchain. Si, après le téléchargement d'un document signé, le hachage est identique à celui généré au moment de la signature, le document est validé.

Si le hachage n'est pas le même, il peut y avoir des risques de manipulation ou d'altération du document signé.

- Signature multiple de transaction cas des wallets multisig

La signature multiple de transaction (Multisig) est un processus de validation de transaction sur une blockchain qui requiert plusieurs clés privées. Il est souvent utilisé pour les transactions financières et les contrats intelligents.

Dans les portefeuilles multisig, chaque utilisateur a sa propre clé privée et plusieurs clés sont nécessaires pour valider une transaction. Cela renforce la sécurité pour les fonds importants ou les transactions en commun.

La signature multiple offre une protection contre la fraude ou la perte de fonds en cas de compromission d'une clé privée, mais peut ralentir et compliquer les transactions en nécessitant la coopération de plusieurs parties.

Les portefeuilles multisig visent à améliorer la sécurité des fonds stockés. Ils sont souvent utilisés pour les fonds partagés ou les transactions importantes qui nécessitent l'approbation de plusieurs parties.


Les portefeuilles multisig peuvent également être utilisés pour les contrats intelligents et les transactions financières sur une blockchain, où il est crucial de garantir la signature de plusieurs parties pour la validation.

On va essayer de faire une multisig transaction avec la plateforme GNOSIS.

Pour ce faire , on crée un coffre fort et on l'associe à un portefeuille.

Bienvenue au coffre-fort


Le protocole de conservation décentralisée et la plateforme de gestion d'actifs collectifs les plus fiables.



Créer un coffre-fort

Un nouveau coffre-fort contrôlé par un ou plusieurs propriétaires.

+ Créer un nouveau coffre-fort



Ajouter un coffre-fort existant

Vous avez déjà un coffre-fort ? Ajoutez votre coffre-fort en utilisant votre adresse de coffre-fort.

Ajouter un coffre-fort existant

Voici notre portefeuille connecté associé à un réseau.



eth : 0xDCf4...1992  

Portefeuille	WalletConnect
Réseau connecté	Polygone

Basculer vers

☐ **Ethereum**

Changer de portefeuille

Déconnecter

Ensuite on crée un coffre fort et on lui associe plusieurs propriétaires de ce fait quand un propriétaire souhaite faire une transaction il lui faudra la confirmation des autres propriétaires. Le coffre fort créé s'appelle blockchain foncier avec 4 propriétaires et pour effectuer une transaction il faut la confirmation de deux propriétaires.

Load Safe

2

Owners and confirmations

Optional: Provide a name for each owner.

Owner name

aissata ndiaye



eth:Ox9F87...b571



Owner name

oumou wade



eth:Ox8712...6C81



Owner name

mareme bousso



eth:Ox80F5...Be9a



Owner name

fallou Adrien



eth:Ox9F7d...86AO












← Back

Next

Charger en toute sécurité

3 Revoir

Confirmez le chargement du coffre-fort.

Réseau	Ethereum
Nom	blockchain foncière
Les propriétaires	<div><div></div><div>aissata ndiaye Ox9F87C1aCaF3Afc6a5557c58284D9F8609470b571  </div></div> <div><div></div><div>Oumou Wade Ox8712128BEA09C9687Df05A5D692F3750F8086C81  </div></div> <div><div></div><div>marème bousso Ox80F59C1D46EFC1Bb18FOAaEc132b77266f00Be9a  </div></div> <div><div></div><div>Fallou Adrien Ox9F7dfAb2222A473284205cdDF08a677726d786AO  </div></div>
Seuil	2 propriétaires sur 4

← Dos

Ajouter

CRÉATION D'UN SMART-CONTRAT QUI VA VALIDER LA MULTI SIGNATURE AVEC LE LANGAGE SOLIDIFY

```
pragma solidity ^0.8.9;

contract MultiSignature {
    struct Signature {
        bytes32 hash;
        address signer;
        bytes signature;
    }

    Signature[] signatures;
    uint256 minimumSignatures;

    constructor(uint256 _minimumSignatures) public {
        minimumSignatures = _minimumSignatures;
    }

    function addSignature(bytes32 documentHash, address signer, bytes memory signature) public {
        require(signatures.length < minimumSignatures, "Not enough signatures");
        Signature memory s = Signature({
            hash: documentHash,
            signer: signer,
            signature: signature
        });
        signatures.push(s);
    }

    function isSigned() public view returns (bool) {
        return signatures.length >= minimumSignatures;
    }

    function getSignatures() public view returns (Signature[] memory) {
        return signatures;
    }

    function getMinimumSignatures() public view returns (uint256) {
        return minimumSignatures;
    }
}
```

Ce code définit un contrat intelligent pour la signature de documents multi-signatures. La structure `Signature` stocke le hash du document, l'adresse du signataire et la signature. Le tableau des signatures stocke toutes les signatures d'un document. La variable **minimumSignatures()** définit le nombre minimum de signatures requises pour que le document soit considéré comme signé.


La fonction **addSignature()** permet aux signataires d'ajouter leur signature au document. La fonction exige que le document n'ait pas déjà été signé et que la signature ait été créée en utilisant le même hachage de document et la même clé privée du signataire.

La fonction **isSigned()** renvoie un booléen indiquant si le document a été signé par le nombre requis de signataires.

La fonction **getSignatures()** renvoie les signatures qui ont été faites.

La fonction **getMinimumSignatures()** renvoie le nombre minimal de signatures qui permettent de valider la transaction.

FICHER MIGRATION

```
migrations >  1675097018_multi_signature.js > ...  
1  var MultiSignature = artifacts.require("MultiSignature");  
2  
3  module.exports = function(deployer) {  
4    deployer.deploy(MultiSignature, 3);  
5  };  
6
```

Dans le fichier de migration, nous allons initialiser le nombre de signatures minimums pour valider la transaction à 3.

FICHIER TEST

```
test > JS MultiSignature.js > ...
1  var MultiSignature = artifacts.require("MultiSignature");
2
3  contract("MultiSignature", function(accounts) {
4      var documentHash = web3.sha3("Test document");
5      var signer1 = accounts[1];
6      var signer2 = accounts[2];
7      var signer3 = accounts[3];
8
9      it("should add signatures", async function() {
10         let multiSignature = await MultiSignature.deployed();
11         await multiSignature.addSignature(documentHash, signer1, "signature1");
12         await multiSignature.addSignature(documentHash, signer2, "signature2");
13         let signatures = await multiSignature.signatures(0);
14         assert.equal(signatures.hash, documentHash, "hash is incorrect");
15         assert.equal(signatures.signer, signer1, "signer is incorrect");
16         signatures = await multiSignature.signatures(1);
17         assert.equal(signatures.hash, documentHash, "hash is incorrect");
18         assert.equal(signatures.signer, signer2, "signer is incorrect");
19     });
20
21     it("should validate transaction with enough signatures", async function() {
22         let multiSignature = await MultiSignature.deployed();
23         await multiSignature.addSignature(documentHash, signer3, "signature3");
24         let isSigned = await multiSignature.isSigned();
25         assert.equal(isSigned, true, "transaction should be signed");
26     });
27
28     it("should not validate transaction with insufficient signatures", async function() {
29         let multiSignature = await MultiSignature.deployed();
30         try {
31             await multiSignature.validateTransaction();
32             assert.fail("transaction should not be valid");
33         } catch (error) {
34             assert.include(error.message, "Not enough signatures", "error message should indicate");
35         }
36     });
37 });
```

Après avoir déclaré chacun des contrats, nous allons procéder aux tests.

1er test: **"should add signatures"** → Cette fonction va nous permettre de vérifier si les signatures sont ajoutées ou pas.

2e test: **"should validate transaction with enough signatures"** → va voir la transaction peut être validé lorsque le nombre de signature minimum est atteints

3e test: **"should not validate transaction with insufficient signatures"** → va voir si la transaction ne sera pas validé lorsque le nombre minimum de signature n'est pas atteint

CONCLUSION

En conclusion, l'utilisation du blockchain dans le domaine foncier peut améliorer la transparence, la sécurité, l'automatisation et l'interopérabilité des transactions foncières, ce qui peut réduire les coûts et les délais, ainsi que renforcer la confiance dans le secteur. Cependant, il est important de noter que l'adoption du blockchain dans le secteur foncier peut être limitée par des défis tels que la réglementation, les normes, la technologie et la culture d'entreprise.

Note: Nous tenons à souligner que nous avons eu quelques soucis avec l'installation de Signserver. Bien qu'on ai installé les prérequis , nous n'arrivons pas à démarrer le serveur d'application.

Nous avons aussi essayé avec l'image de signserver sur docker mais après avoir démarré le container, nous n'avons pas pu accéder à l'interface web de signserver avec localhost:8080.