DeSign : Signature décentralisée

Un outil de signature électronique sur blockchain Ethereum.

Equipe : Fawzi Benhalima (chef de projet), Guillaume Heu (développeur)
Auteur : Guillaume Heu
Github : https://github.com/guillheu/DeSign

Outils utilisés

Brownie : framework de développement blockchain en Python (équivalent de truffle)
Web3j & Web3.js : bibliothèques web3 pour Java et Javascript respectivement ; Web3j génère les classes Java “wrapper” pour communiquer avec les smart contracts
Smart Contracts OpenZeppelin : gestion des roles dans le smart contract (AccessControl.sol)
Eclipse : IDE Java
Tomcat : serveur web en Java
Maven : outil de gestion de dépendences pour projets Java
MySQL & MySQLWorkbench : base de données SQL
Git & Github : gestion des sources

Fonctionalités implémentées

Smart contract

- Gestion des accès signataires & administrateur (AccessControl de OpenZeppelin)

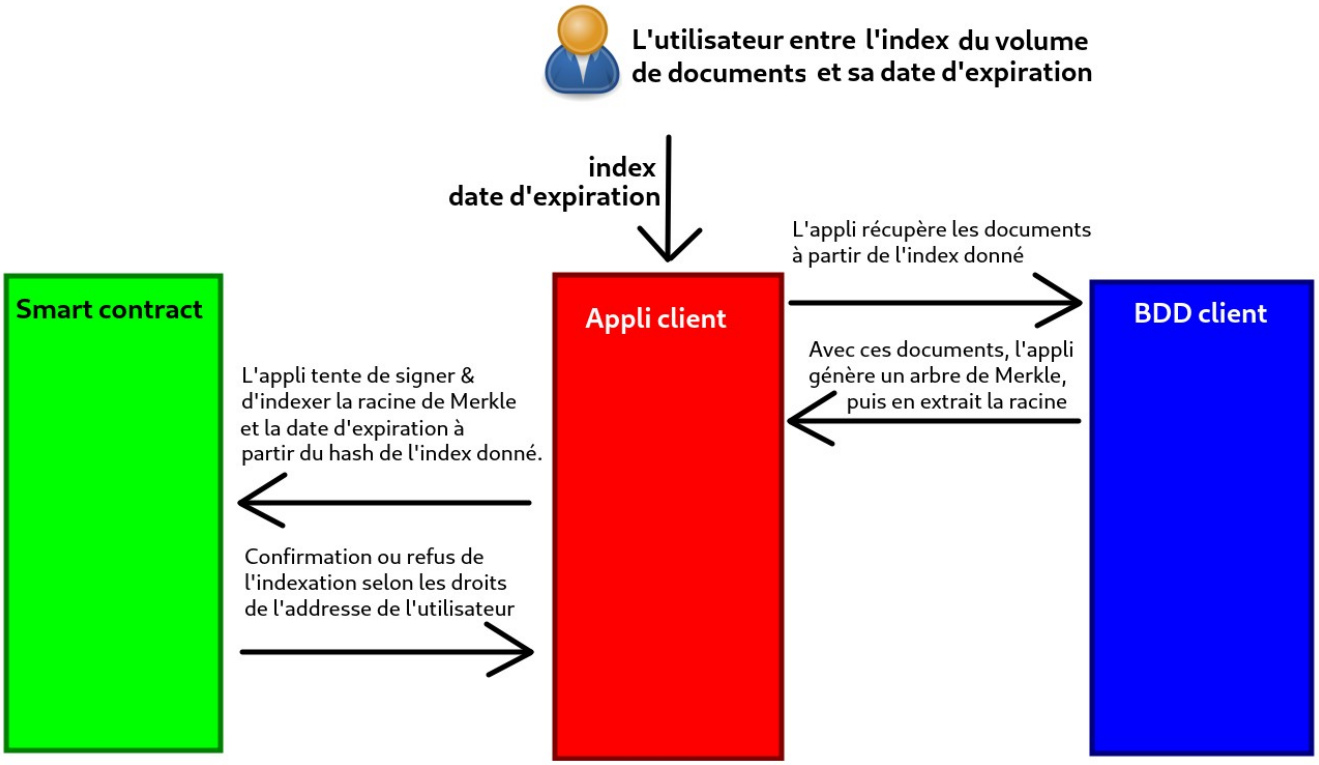
- Référencement des données du volume de documents (racine de Merkle indexée, timestamp d’expiration, adresse du signataire) à partir d’un index hashé

- Lecture des données référencées (racine de Merkle indexée, timestamp d’expiration, adresse du signataire) à partir d’un index hashé

Applications clients

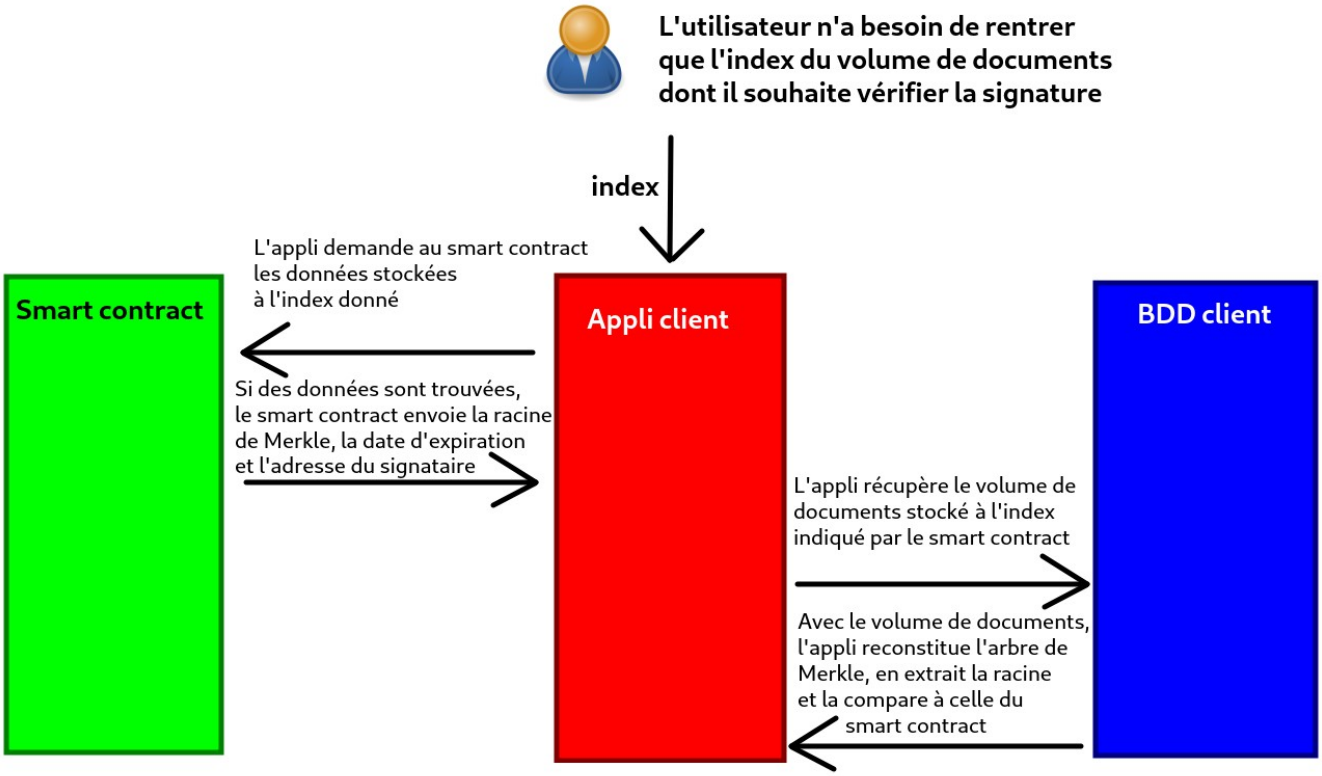
- Deux clients “lourds” : ligne de commande ou interface web (utilisant Tomcat) :
    - Appel des fonctions de lecture et d’écriture du smart contract déployé à partir de données humainement lisibles
    - Connexion à une base de données SQL de stockage des documents (formatée spécifiquement)
    - Connexion à n’importe quel noeud faisant tourner une EVM (Réseau Ethereum/Tezos/Ganache...)
    - Signature des transactions à partir d’une clé privée donnée à l’application
    - Génération d’une preuve de signature (fichier json)
- Client léger (page web statique déployée sur IPFS) :
    - vérifie la validité des couples document + preuve de signature (ne nécessite pas de wallet pour navigateur comme metamask)
    - Configutation simple via un fichier humainement lisible

Processus d’écriture :
Signature & indexation d’un volume de documents



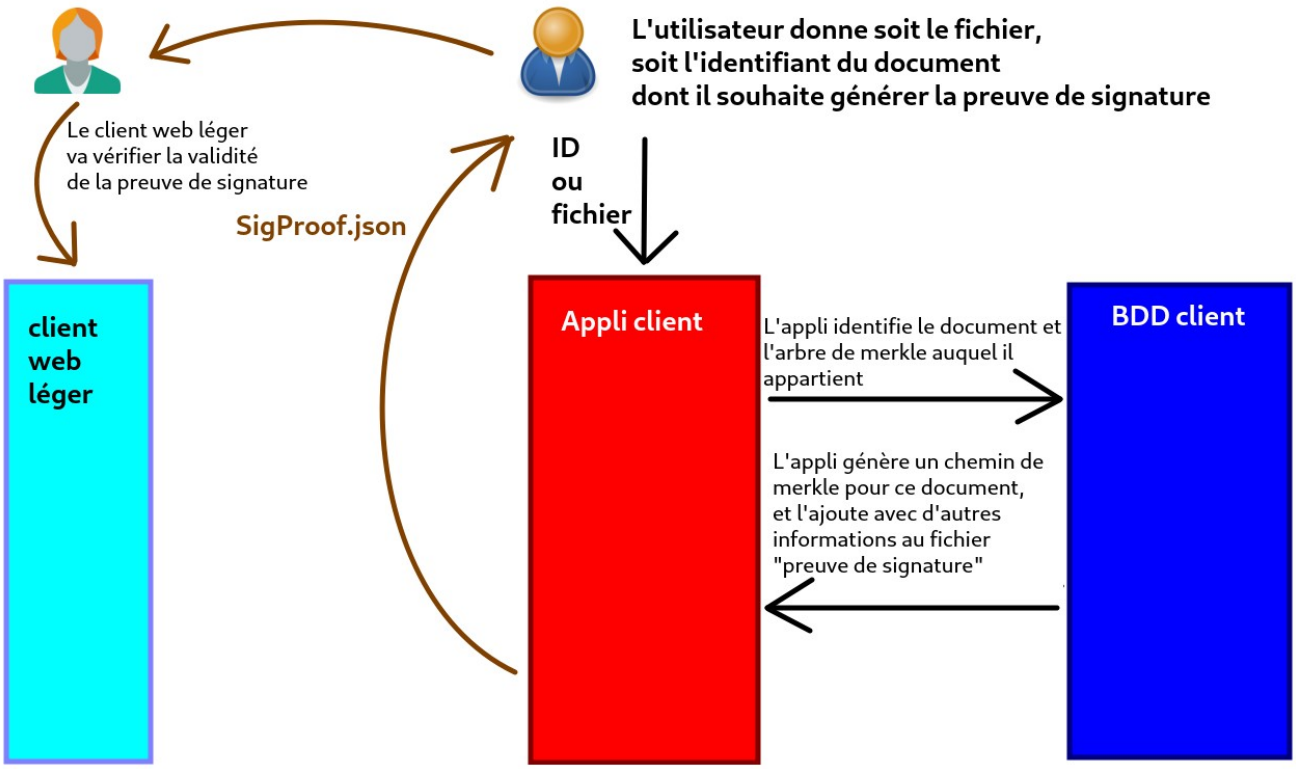
Si le processus d’écriture réussit, la signature devient indélébile, irrévocable, inaltérable et publique. En revanche, les documents ne le sont pas.

Processus de lecture :
Vérification de la signature et de la validité des documents



Les racines de Merkle sont comparées ; si elles ne correspondent pas, alors la signature ne peut pas être vérifiée et l'intégrité ou la configuration de la base de données client est remise en cause.
Si elles correspondent, alors la signature est validée pour tous les documents contenus dans le volume de documents

Processus de validation :
Génération et vérification d'une preuve de signature



N'importe quel utilisateur ayant le document original ainsi que la preuve de signature peut vérifier la validité de la signature stockée sur la blockchain ainsi que l'adresse du signataire.
Le client web léger va vérifier la signature par le même processus que la vérification par client lourd, et utilisera le chemin de Merkle de la preuve de signature pour retrouver la racine de Merkle au lieu de questionner une base de données.

Voici un exemple de fichier de preuve de signature :


```
{
  "merklePath": [
    {
      "hashFrom": "RIGHT",
      "hashWith": "0x74b87db5cdd912723071d4058159a947bb42159b37ff396c30417e3c3ffbee54"
    },
    {
      "hashFrom": "LEFT",
      "hashWith": "0xa7a593010f7580dcc67289b0a2cc367d3fe234d6dd6d1089fbf946ac9803dbd1"
    }
  ],
  "indexHash": "0xdca6fc1c8ddb45ca3b236328b54917a2d1988fc7a2a11e913391920bffc5ff9f",
  "contractAddr": "0xD8D74044703C2f98B38E048c639F2c32860cA278",
  "nodeURL": "https://kovan.infura.io/v3/7cdcc900133c425fab136c45f004893b",
  "hashAlgo": "SHA-256"
}
```


Le chemin de Merkle (ou Merkle path) nous permet de prouver l'appartenance d'un fichier à l'arbre de Merkle dont on connait la racine sans avoir à reconstruire l'entièreté de l'arbre.

Développement futur : quelles fonctionalités ?

- API REST
- ERC 725/735 pour l'autentification, l'autorisation et la gestion d'identités
- demande de signature d'un tiers directement sur la blockchain
- multi-signature
- importer un fichier wallet au lieu de rentrer sa clé privée sur le fichier de configuration
- gas price & gas limite variables
- intégrer plus de types de stockages (système de fichiers local, bases de données non-SQL…)
- possibilité pour le client web "light" d'utiliser une connexion au réseau via metamask au lieu du lien vers noeud donné dans la preuve de signature
- auto-génération & modification du fichier de config directement depuis les clients lourds (wizard d'initialisation)
- Décentraliser les preuves de signature (télécharger les fichiers json directement sur IPFS par exemple)