DEPARTMENT DE SCIENCES ET INGÉNIERIE

PROJET DE RECHERCHE

Implémentation d'un Smart-Contract lié à l'Identité et de la Balance des Interactions

Auteurs:
Chan Yeong HWANG
Elina JANKOVSKAJA
Messilva MAZARI
Karima SADYKOVA

Superviseur:
Nour EL MADHOUN

En collaboration avec
Daniel MALDONADO-RUIZ
de l'EPN(École Polytechnique Nationale de Quito)

May 2022

Remerciements

Nous souhaitons adresser nos remerciements aux personnes qui nous ont apporté leur aide pour mener à bien ce projet et à l'élaboration du rapport.

Ce travail ne serait aussi riche et n'aurait pas pu avoir le jour sans l'aide et encadrement de Madame El-Madhoun, on la remercie pour la qualité de son encadrement, sa patience, sa rigueur et sa disponibilité tout au long de la réalisation de notre projet.

Nos remerciements s'étendent également à Monsieur Maldonado-Ruiz pour son aide pratique et ses encouragements. Le soutien inconditionnel de Madame El-Madhoun et Monsieur Maldonado-Ruiz durant la préparation du projet malgré leurs charges académiques et professionnelles a été d'une grande aide pour nous.

Contents

1 Introduction

Ce projet découle d'une recherche doctorale intitulée "Decentralised Identities based on Next Generation Autonomous Networks" (Identités décentralisées basées sur les réseaux autonomes de nouvelle génération), qui vise à créer un réseau autonome dans lequel les utilisateurs peuvent stocker des informations sans dépendre d'une tierce partie pour valider les données stockées. Au cours de notre projet, nous avons donc cherché à stocker et à superviser l'identité numérique d'utilisateurs au sein d'un réseau autonome. La blockchain semblait être un choix évident de réseau de par sa sécurité et la facilité de ses échanges.

Pour réaliser notre projet, nous avions alors besoin d'un moyen de communication avec la blockchain, c'est alors que le concept de "smart-contract" est apparu et a semblé évident. Au cours de notre projet, nous avons donc cherché à faire la liaison entre la blockchain et l'identité des utilisateurs à travers un smart-contract dans un effort d'automatisation de procédés tout en gardant les caractéristiques de la blockchain: confidentialité, intégrité, authentification et immuabilité.

2 Concepts du projet

2.1 La blockchain

Mais qu'est-ce que la blockchain? Fondamentalement, une blockchain est une technologie de stockage et de transmission d'informations sans autorité centrale. Si nous prenons l'exemple des échanges monétaires entre particuliers, le fait d'utiliser une blockchain permet d'effectuer l'envoie directement d'un utilisateur à un ou plusieurs destinataires sans faire recours à une banque ou une tierce partie.

Ainsi une blockchain est un registre distribué, une grande base de données qui a la particularité d'être partagée simultanément avec tous ses utilisateurs, tous également détenteurs de ce registre, et qui ont également tous la capacité d'y inscrire des données. Ces données seront protégées par un protocole informatique avec de hauts standards de transparence et de sécurité. La blockchain ne possède pas d'organe central de contrôle ce qui permet de garantir l'anonymat de chaque utilisateur et la rapidité des transactions: un échange se fera directement entre les deux partis. Une fois ajoutée à la blockchain, les données ne peuvent être supprimées ou modifiées ce qui garantit l'immuabilité. La blockchain va ainsi servir de support de stockage et de validation de données, on la retrouve dans des applications de réseaux sociaux ou de jeux.

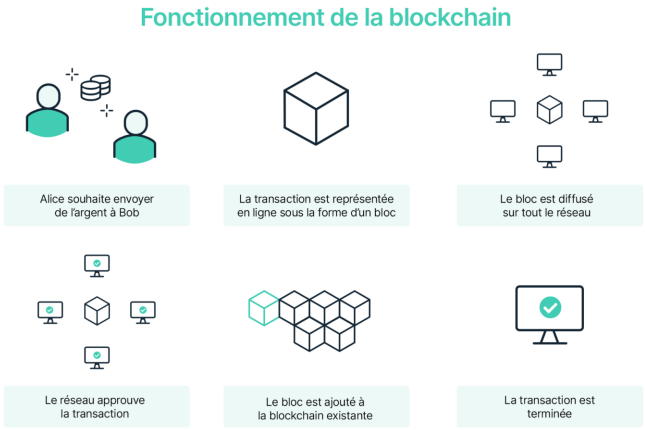

Figure 1: Schéma sur le fonctionnement de la blockchain.

Si on observe de plus près son fonctionnement sur la Figure 1, lorsqu'un utilisateur souhaite envoyer de l'argent il transmet les informations nécessaires sur le montant et le destinataire ainsi qu'une signature via le système de clé privée/-clé publique permettant de crypter et d'authentifier l'envoi. À partir des données fournies, on génère une empreinte unique qui permet d'identifier les données initiales. Ainsi, les données et l'empreinte sont regroupées dans un bloc. Avant d'arriver à son destinataire, la transaction va être diffusée sur tout le réseau. Les nœuds (les autres utilisateurs de la blockchain) vont ainsi diffuser le bloc à d'autres noeuds à la manière d'une toile d'araignée. Parmi ces nœuds, il va y avoir les "mineurs" qui

vont vérifier les informations envoyées et être récompensés en conséquence. Une fois que le bloc est validé, il est ajouté à la chaîne de blocs. Cette chaîne permet de lier tous les blocs depuis la création de la blockchain et ainsi de les authentifier. La transaction de départ est alors visible pour le destinataire ainsi que l'ensemble du réseau.

Des distinctions existent aussi au sein des blockchain. Nous avons premièrement les blockchains publiques qui sont des réseaux ouverts et décentralisés accessibles à quiconque souhaitant demander ou valider une transaction (en vérifier l'exactitude). Deux exemples courants de blockchains publiques sont les blockchains Bitcoin et Ethereum. Ensuite, nous avons les blockchains privées qui ont des restrictions d'accès. Les personnes qui veulent les rejoindre doivent obtenir la permission de l'administrateur du système. Elles sont généralement régies par une seule entité, ce qui signifie qu'elles sont centralisées. Pour finir, il y a les consortiums, appelés aussi blockchains hybrides, qui contiennent des caractéristiques centralisées et décentralisées. Comme exemple nous avons Energy Web Foundation, Dragonchain et R3.

2.2 Les contrats intelligents

À l'image d'un contrat habituel, un contrat intelligent énonce les conditions d'un accord. Mais contrairement à un contrat traditionnel, les termes d'un contrat intelligent sont appliqués sous forme de code exécuté sur une blockchain comme Ethereum. Les contrats intelligents permettent aux développeurs de créer des applications qui tirent parti de la sécurité, de la fiabilité et de l'accessibilité de la blockchain, et offrent des fonctionnalités pair-à-pair sophistiquées, allant des prêts aux assurances en passant par la logistique et les jeux vidéo.

Ce qui rend les contrats « intelligents », c'est que les modalités sont établies et signées en tant que code exécuté sur une blockchain, plutôt que sur du papier traînant sur le bureau d'un avocat. Les contrats intelligents s'appuient sur le concept de base du Bitcoin (envoi et réception d'argent sans intermédiaire de confiance, par exemple) pour automatiser et décentraliser tous les types d'opérations ou de transactions ou presque, quel que soit leur degré de complexité. Et comme ils fonctionnent sur une blockchain telle que l'Ethereum, ils garantissent une sécurité, une fiabilité et une accessibilité sans frontières.

L'importance des contrats intelligents vient du fait qu'ils permettent aux développeurs de créer une grande diversité d'applications décentralisées et de jetons. Ils sont utilisés dans tous les domaines, de la finance (pour la création de nouveaux outils financiers) à la logistique en passant par les jeux vidéo. Les contrats intelligents sont aussi stockés sur la blockchain, et comme nous l'avons vu précédemment la blockchain ne permet pas la suppression ou la modification de données, on garantit alors l'immuabilité.

2.3 La cryptographie sur les courbes elliptiques

Proposée indépendamment par Victor Miller et Neal Koblitz en 1985 une courbe algébrique va permettre de générer des clés sécurisées à l'aide d'une équation très

simple. Cette équation génère une courbe. On choisit ensuite un point de la courbe et on effectue sa tangente. L'opposé de cette tangente nous donne un point à partir duquel on va encore faire une tangente et ainsi de suite. Finalement, on aura une paire de clés très efficaces mais impossibles à retrouver.

Dans le cadre de ce projet, nous pensions tout d'abord générer des clés à l'aide de courbes elliptiques mais nous avons eu des problèmes techniques qui nous ont poussés à finalement opter pour un chiffrement RSA.

2.4 Le chiffrement RSA

Le chiffrement RSA (du nom de ses trois créateurs) est un algorithme de cryptographie asymétrique utilisé pour échanger des données confidentielles sur le réseau Internet. RSA utilise une paire de clés (des nombres entiers) composées d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer l'information envoyée sur internet. Les deux clés sont créées par une personne "X" qui souhaite que lui soient envoyées des données confidentielles. Sa clé publique est utilisée par les utilisateurs qui veulent chiffrer les données pour les renvoyer à la personne "X". La clé privée, quant à elle, reste privée et accessible qu'à la personne "X" pour lui permettre de déchiffrer les données reçues. La clé privée est aussi utilisée pour signer une donnée que "X" envoie, tandis que la clé publique permet à n'importe lequel de ses correspondants de vérifier la signature.

Une condition indispensable dans le chiffrement RSA est que cela soit impossible de déchiffrer les données à l'aide de la clé publique, en particulier de reconstituer la clé privée à partir de la clé publique.

2.4.1 Création des clés

L'étape de création des clés est à la charge de la personne "X". Elle n'intervient pas à chaque chiffrement, car les clés peuvent être réutilisées.

- Choisir p et q, deux nombres premiers distincts
- Calculer leur produit $n = pq$ appelé module de chiffrement
- Calculer $\phi(n) = (p-1)(q-1)$ - valeur de l'indicatrice d'Euler en n
- Choisir un entier naturel e premier avec $\phi(n)$ et strictement inférieur à $\phi(n)$, appelé exposant de chiffrement
- Calculer l'entier naturel d, inverse de e modulo $\phi(n)$ strictement inférieur à $\phi(n)$, appelé exposant de déchiffrement. d peut se calculer par l'algorithme d'Euclide.

Finalement, on obtient la clé publique constituée de deux nombres n et e et la clé privée décrite par nombre d. Pour déchiffrer l'information faut avoir la clé privée d et l'entier n.

2.5 La signature électronique

Une signature numérique est un mécanisme permettant de garantir la non-répudiation d'un document électronique et d'en authentifier l'auteur, par analogie avec la signature manuscrite d'un document papier. La signature numérique permet d'ajouter à tout document ou message une preuve que le document provient d'un vrai expéditeur ce qui est vérifié en utilisant sa clé publique.

Ainsi, la signature numérique peut :

- Prouver que le message n'a pas été modifié (intégrité)
- Prouver la source du message (authentification)
- S'assurer que la signature numérique n'est pas fausse et que le signataire ne peut pas la répudier (non-répudiation)

La signature numérique est réalisée par la clé privée de l'expéditeur:

- Le document n'a pas besoin d'être crypté
- Personne d'autre ne peut signer, car seul l'expéditeur possède la clé privée
- Toute modification du message (ou du document) invalidera la signature
- Pour vérifier la signature numérique, la clé publique correspondante est nécessaire

Ainsi, nous avons compris le fonctionnement de l'utilisation de la cryptographie asymétrique pour transmettre proprement des données sur le réseau Internet (Figure 2) et nous possédons maintenant tous les concepts nécessaires à la compréhension de notre projet.

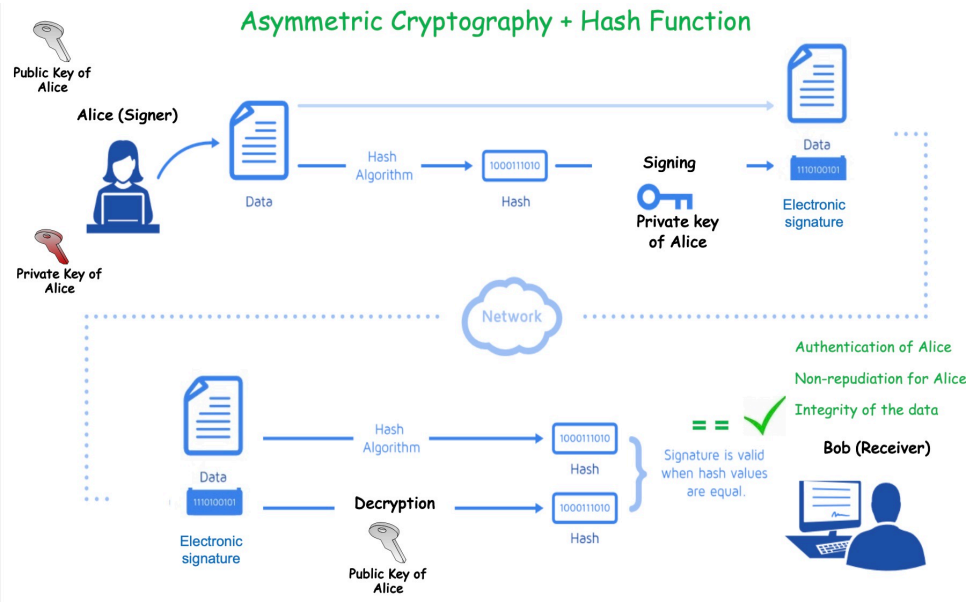

Figure 2: Description détaillée de l'utilisation de la cryptographie asymétrique avec la création d'une signature électronique

3 Objectifs

- Créer une structure de contrats intelligents qui peut prendre en charge le stockage des informations d'identité d'utilisateurs et associer ces informations à une identité spécifique sur la blockchain.
- Utiliser la cryptographie à courbe elliptique/RSA pour créer une paire de clés publiques et privées pour chaque utilisateur afin de garantir la sécurité des informations de chaque utilisateur.
- Définir un tableau de signatures numériques pour certifier la validation des informations stockées dans le contrat intelligent afin que chaque utilisateur de la blockchain puisse savoir que les informations stockées sont vérifiées.
- Créer une structure de vérification à l'intérieur d'un contrat intelligent afin de connaître la validité et le temps d'expiration de l'information stockée ainsi que le nombre de changements qui ont été effectués dans celle-ci.

4 Introduction du problème

Le problème consiste donc à décentraliser les données d'un utilisateur et de lui permettre de sauvegarder et gérer ses données sur le réseau. On va donc chercher à créer un espace de stockage d'identité.

Dans la Figure 3, on peut voir les schémas proposés dans le cadre de la recherche doctorale. Bien que la plupart des caractéristiques sont au-delà de la portée de ce projet, cela permet de schématiser la structure du «core blockchain»: une structure privée interne où les paires de clés liées à la sécurité du réseau sont stockées en toute sécurité. La structure appelée «main blockchain» quant à elle est celle où seront stockés les contrats intelligents conçus dans le cadre de ce projet.

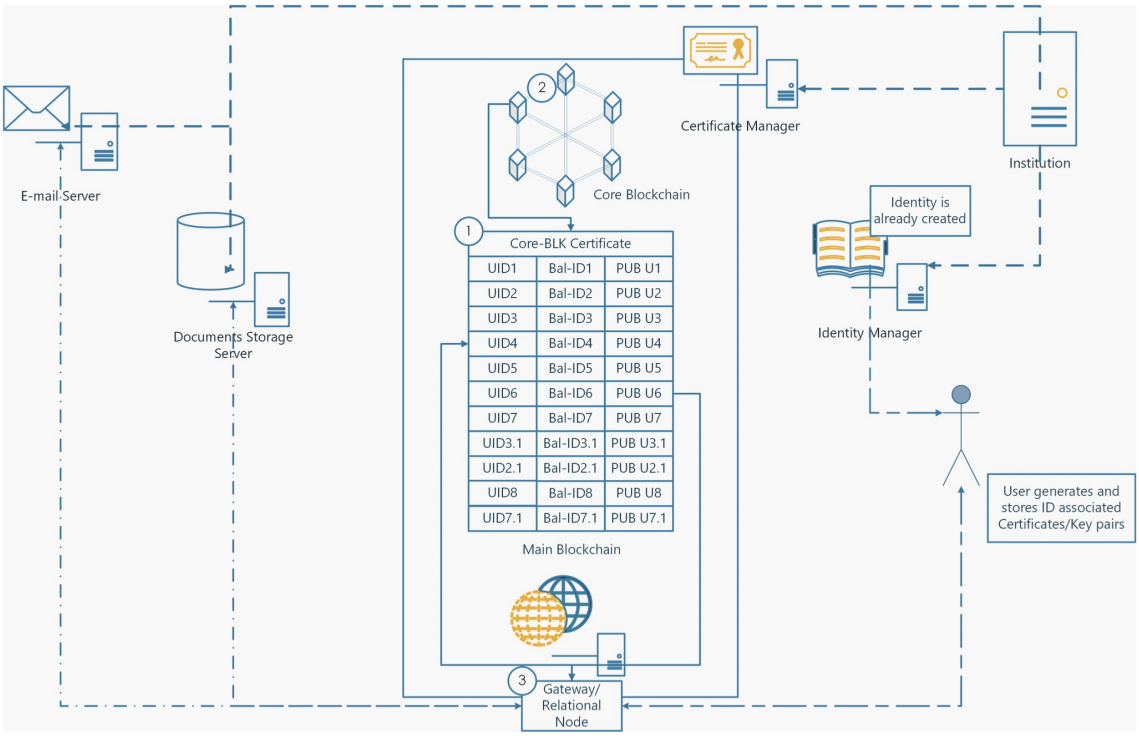

Figure 3: Schémas de la proposition de réseau autonome

5 Proposition du core du projet

Notre proposition pour résoudre le problème de décentralisation des données consiste à créer un contrat intelligent composé des éléments décrits sur la Figure 4.

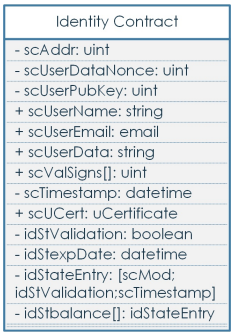

Figure 4: Diagramme UML du contrat intelligent

Ainsi, la réalisation du projet se divise en plusieurs étapes.

5.1 Conception théorique

La première étape est décrite dans la Figure 5 : l'utilisateur envoie ses informations à travers deux processus différents, l'un où l'utilisateur valide son identité en utilisant ses informations secrètes, et le second où il envoie ses informations (nom, e-mail, données supplémentaires d'identité) pour être stockées dans le contrat intelligent. La validation n'est pas destinée à être implémentée dans ce projet, donc comme le montre la Figure 6, l'acquisition des informations d'identification et le calcul de la clé publique doivent être effectués dans un module externe non lié au contrat intelligent. Lorsque les données seront stockées dans le contrat intelligent, ce dernier hachera la clé publique et l'identité de l'utilisateur pour les transformer en certificat. Il va ensuite stocker ce certificat et enregistrer sa date de création.

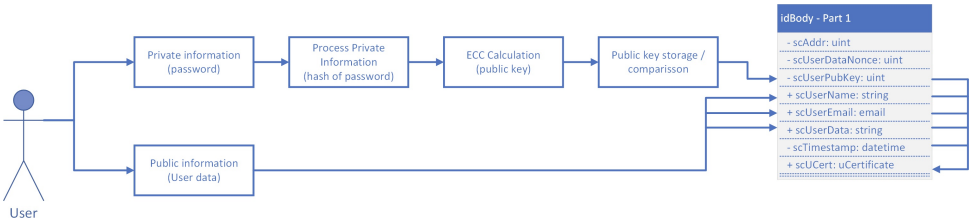

Figure 5: Acquisition des données utilisateur à stocker

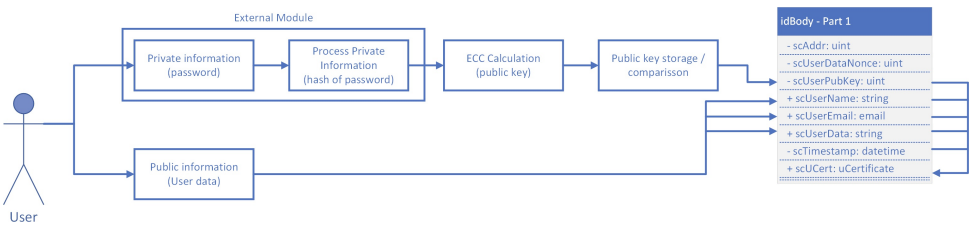

Figure 6: Mise en œuvre de l'acquisition de l'information

La deuxième étape, décrite à la Figure 7, consiste à hacher les informations et signer consécutivement par deux signatures différentes. Les deux signatures sont ensuite stockées dans un tableau pour une validation externe.

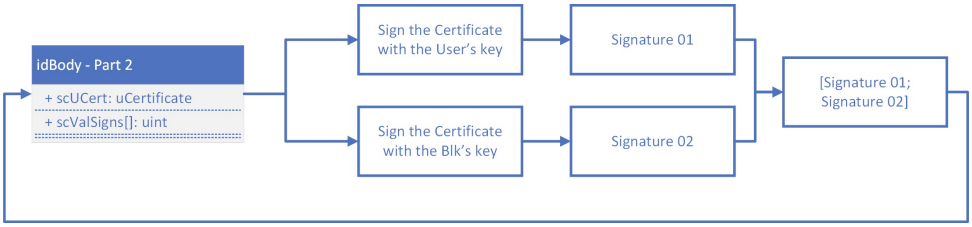

Figure 7: Procédé de signature de l'information acquise

La première signature est effectuée par la clé privée de l'utilisateur (acquise avec le module externe) et la seconde est fournie par la structure de sécurité interne du réseau, représentée par la Core Blockchain (2) sur la Figure 3. Dans ce projet, le module externe créera une paire de clés supplémentaires appelée "Network" (une clé publique et une clé privée), avec lesquelles seront effectuées toutes les deuxièmes signatures des informations dans le contrat intelligent. On pourrait se demander pourquoi on introduit une seconde signature de la part du Network. Une seconde signature de la part du Network est introduite pour valider les informations transmises par l'utilisateur auprès de la blockchain.

Pour mieux comprendre pourquoi une telle signature est nécessaire, on peut utiliser un exemple: supposons que Sorbonne université soit une blockchain et qu'un étudiant (Bob) soit un utilisateur de cette blockchain. Pour recevoir son identifiant étudiant, Bob doit rassembler toutes ses informations, les envoyer à Sorbonne université et attendre qu'une personne à l'administration valide chacune de ces informations à la main. De plus, si jamais Bob voulait changer ses informations, par exemple dû à un déménagement, il devrait contacter l'administration pour effectuer ce changement sur la blockchain de Sorbonne université.

Notre contrat intelligent permet de supprimer l'administration de la relation entre Bob et l'université. Ainsi chaque étudiant peut générer un identifiant par lui-même en envoyant toutes ses informations directement à l'université avec une signature de la forme de Signature 1 (Figure 7) faites avec ses informations et sa clé privée. Ce semi-certificat ne sera pas encore complet. Il faudra alors que l'université génère automatiquement une signature, c'est la Signature 2 (Figure 7) faites aussi avec les informations de l'étudiant mais cette fois-ci avec la clé privée de Sorbonne Université, pour signer l'information envoyée et générer alors un identifiant étudiant. Les deux signatures seront ensuite enregistrées sur la blockchain pour permettre à l'étudiant de retrouver les informations qu'il a stockées et d'assurer qu'elles ont été validée par Sorbonne Université.

Finalement, la Figure 8 représente les variables dans lesquelles on enregistre les validations et les ajouts au contrat intelligent (par exemple un nouvel utilisateur, ou un utilisateur qui aurait mis à jour ses données). L'heure de validation, d'expiration ou encore le certificat de l'utilisateur sont stockées dans un tableau pour savoir combien de fois les informations du contrat intelligent ont été modifiées

et mis à jour. Cette étape termine la création du contrat intelligent.

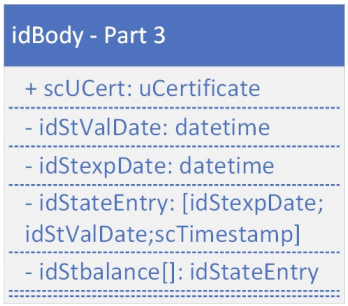

Figure 8: Mise à jour de la structure de validation

5.2 Tâches procédurales

Ainsi, les tâches à effectuer sont:

- Créer un prototype du contrat intelligent proposé sur un réseau Ethereum et définir tous les champs nécessaires à sa mise en œuvre.
- Créer un module pour générer des paires de clés publiques/privées en utilisant la cryptographie à courbe elliptique/RSA pour fournir au contrat intelligent les informations de l'utilisateur et la signature.
- Implémenter la première étape, en validant la création d'une structure de type certificat qui sera utilisée pour valider l'identité de l'utilisateur.
- Mise en œuvre de l'étape deux, où les informations de l'utilisateur sont signées et validées par le réseau.
- Mettre en œuvre l'étape trois et créer le solde de validation pour chaque utilisateur avec un contrat intelligent.

6 Implémentation de la proposition du projet

6.1 Première partie

Les trois premières semaines de notre projet ont été consacrées à la compréhension de la structure de la blockchain ainsi que de la place des contrats intelligents au sein de celle-ci. Nous avons ensuite survolé des notions basiques de cryptographie notamment le chiffrement asymétrique à deux clés. Durant ce temps, nous avons également appris à manipuler Solidity, un langage orienté objet dédié à l'écriture de contrats intelligents qui est utilisé sur diverses blockchains, notamment Ethereum.

Une fois les contrats créés, il fallait pouvoir les tester et pour cela on a eu besoin de différents outils: Ganache (Figure 9) pour nous permettre d'effectuer des tests sans devoir acheter des Ether (la monnaie utilisée sur la blockchain Ethereum) et Remix IDE pour compiler et identifier les failles de sécurité de notre contrat plus rapidement et plus efficacement.

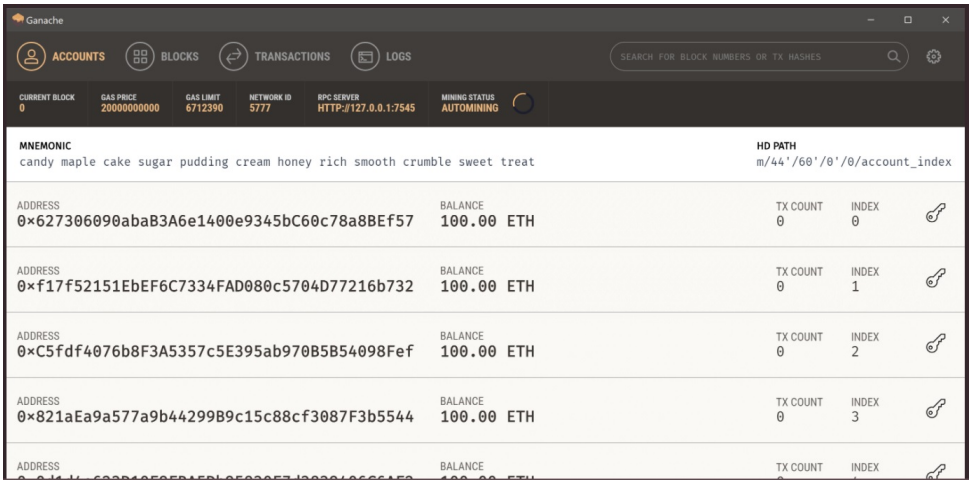

Figure 9: Interface de Ganache, l'outil de développement high-end

A la suite de cette étape préliminaire, nous avons pu définir notre projet ce qui nous a permis de commencer à le réaliser. Une fois le contrat solidity fait selon ces spécifications, nous avons cependant rencontré certains problèmes: nous n'avions pas généré la paire de clés et notre interface utilisateur n'était pas prête.

6.1.1 Clé privée et Clé publique

Nous avons choisi de générer des paires de clés à l'aide de Javascript. Ensuite on a rajouté des fonctions nous permettant de récupérer l'identité de l'utilisateur. Ces nouveaux éléments ont donc complété notre contrat. Cependant, une identité sur la blockchain n'est pas uniquement composée d'une paire de clés et de données personnelles. Comme nous l'avons vu plus haut, celle-ci nécessite aussi une signature électronique et un hachage pour sécuriser notre contrat suivant la norme x.509 (une norme spécifiant les formats pour les certificats à clé publique qui nous a été imposée dans le cadre de ce projet).

Ainsi, nous avons généré deux hachages, un du côté de l'utilisateur et l'autre du côté de Solidity et nous les avons comparés pour parer une éventuelle attaque

venant s'interposer entre nos deux programmes. Par la suite, ces hachages et la clé privée de l'utilisateur permettront la génération de la signature.

6.1.2 La signature électronique

La signature nécessite l'ensemble des informations de l'utilisateur ce qui nous a poussé à la produire dans l'interface de l'utilisateur. Par la suite, nous allons devoir générer une deuxième signature en provenance de la blockchain qui nous permettrait de doublement signer le contrat.

6.2 Deuxième partie

Dans cette deuxième et dernière étape de notre projet, nous avons choisi de nous concentrer sur la résolution des problèmes techniques et sur la finalisation des objectifs.

6.2.1 Sauvegarde de la signature sur Solidity

Tout d'abord nous avons finalisé la signature: nous avions des soucis pour envoyer la signature au contrat Solidity car notre fonction n'attendait pas l'ensemble des données de l'utilisateur. Nous avons alors eu recours aux fonctions asynchrones. Une fonction asynchrone peut contenir une expression await (attendre en français) qui interrompt l'exécution de la fonction asynchrone et attend la résolution de la promesse passée Promise. La fonction asynchrone reprend ensuite puis renvoie la valeur de résolution. Nous avons choisi d'utiliser des fonctions asynchrones car elles permettent donc d'attendre les informations de l'utilisateur. Pour résoudre ce problème nous avons donc redéfinit nos fonctions avec le mot clé "await" qui nous permet alors d'attendre l'obtention de l'information de la part de l'utilisateur au lieu d'envoyer directement sur Solidity (ce qui causait une perte d'information). Ainsi, nous pouvons alors envoyer notre signature sur Solidity (Figure 10).

Figure 10: Création de la signature à partir de la clé privé de l'utilisateur et du hash

6.2.2 Génération de la clé RSA

Précédemment, nous avions généré notre paire de clé aléatoirement, sauf que l'objectif du projet consistait à sécuriser l'information de l'utilisateur. Pour résoudre ce problème nous avons produit la clé à partir de l'algorithme de cryptographie asymétrique RSA. Ainsi nous avons généré les deux clés dans un fichier séparé à l'aide de Javascript (Figure 11).

```
/*------------------------------ Key Generation ---------------------------------*/
const crypto = require("crypto");

// The `generateKeyPairSync` method accepts two arguments:
// 1. The type ok keys we want, which in this case is "rsa"
// 2. An object with the properties of the key
const { publicKey, privateKey } = crypto.generateKeyPairSync("rsa", {
  // The standard secure default length for RSA keys is 2048 bits
  modulusLength: 2048,
});
```

Figure 11: Production de la clé RSA à partir d'une bibliothèque externe

6.2.3 Finitions sur les hachages

Nous avions déjà généré deux hachages dans la première partie du projet. Cependant nous avons décidé de renforcer la sécurité de ces deux hachages en rajoutant des données dans leur composition. Le hachage se déroule donc d'un côté sur Solidity et de l'autre sur Javascript (Figure 12). Ce hachage commence sur Solidity et est fait à partir du nom, de l'email et des données de l'utilisateur, mais aussi

avec un entier généré aléatoirement et unique appelé "Nonce". Le hachage effectué sur Solidity et Nonce vont ensuite être envoyés au fichier Javascript qui va ensuite pouvoir procéder au second hachage. Une fois les deux hachages obtenus on va les comparer, s'ils sont identiques, une tierce partie malicieuse ne s'est pas introduite entre nos deux fichiers et nous pouvons donc continuer et produire la signature de l'utilisateur.

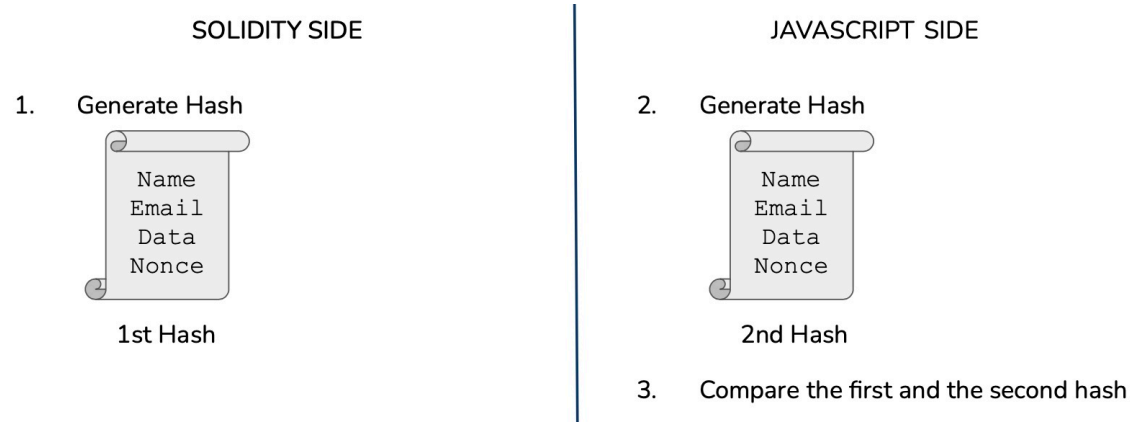

Figure 12: Production du Hash User

Finalement, nous avons rajouté un deuxième hachage pour le Network qui effectue la double vérification (Figure 13)

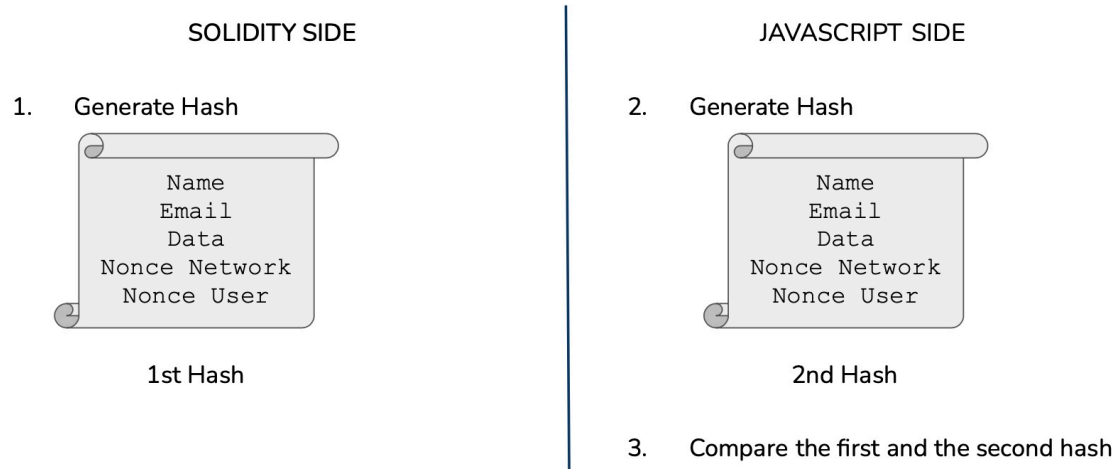

Figure 13: Production du Hash Network

7 Simulation du projet/Guide d'utilisation

Un contrat intelligent est spécifique à un seul réseau, celui dans lequel il est déployé. Déployer un contrat intelligent consiste à envoyer sur la blockchain une transaction contenant le code du contrat intelligent compilé sans spécifier de destinataire. Lorsqu'il est déployé, il crée une instance (compte de contrat) sur le réseau, ce qui le rend disponible pour les utilisateurs.

Pour pouvoir déployer notre contrat sur le réseau Ethereum, nous avons utilisé l'IDE en ligne Remix-Ethereum (https://remix.ethereum.org/).

On a aussi eu recours à GANACHE, qui est une blockchain personnalisée pour le développement d'Ethereum (téléchargeable depuis le site https://trufflesuite.com/ganache/) afin d'exécuter notre code et tester son bon fonctionnement.

Les étapes à suivre pour déployer le contrat intelligent sont les suivantes :

1. Compiler le code Solidity sur Remix-Ethereum en veillant à choisir la bonne version du compilateur. (Figure 14)

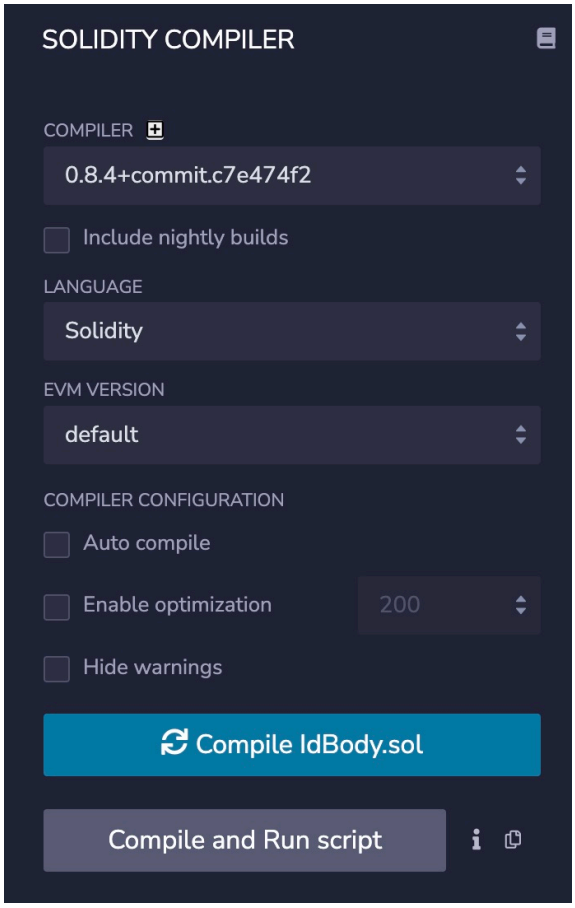

Figure 14: Compilateur Solidity

2. Après la compilation du code, copier l'ABI du contrat pour pouvoir l'utiliser dans les fichiers Javascript (index.js et network.js). (Figure 15)

Figure 15: Génération de l'ABI par le compilateur

3. Lancer GANACHE et choisir Quickstart, cela permettra d'avoir accès à 10 comptes sur le réseau Ethereum avec 100eth sur chacun afin de pouvoir déployer le contrat. Nous aurons besoin de l'adresse d'un des comptes que nous allons utiliser, cette adresse sera utilisée pour initialiser la valeur de la constante userAccount dans les fichiers Javascript. (Figure 16)

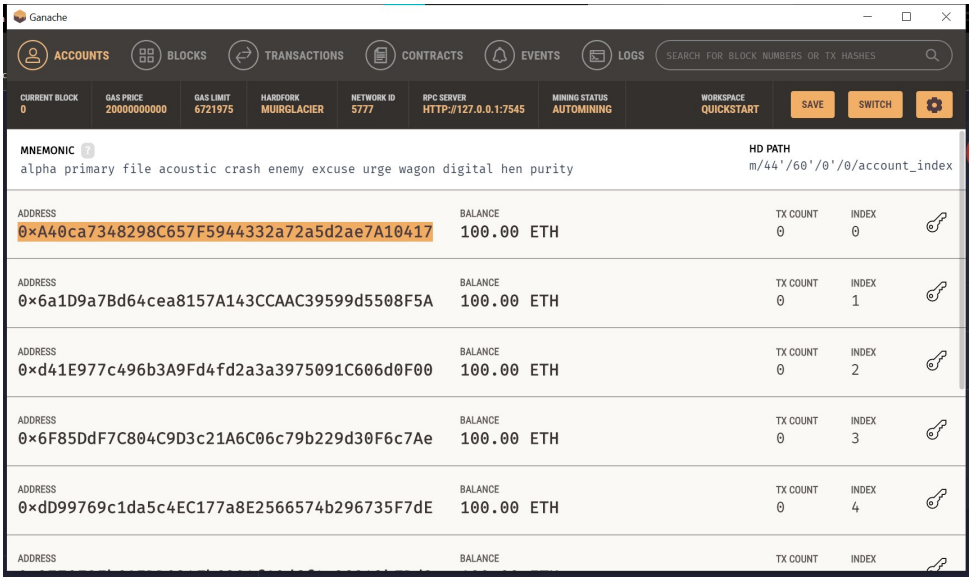

Figure 16: Interface de GANACHE

4. Déployer le contrat Solidity sur le réseau Ethereum cliquant sur "Deploy". (Figure 17)

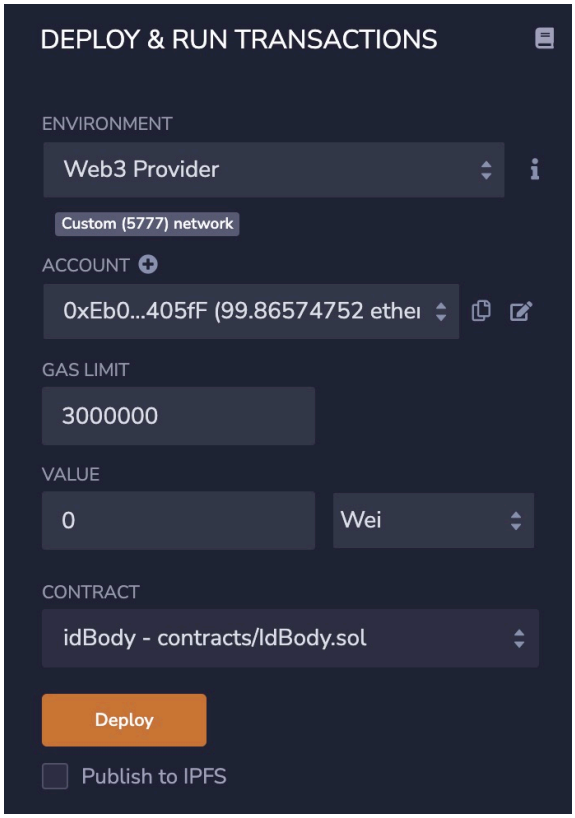

Figure 17: Rubrique "Déployer et Exécuter les transaction" sur Remix-Ethereum

5. En choisissant Web3 Provider comme environnement et en copiant le RPC SERVEUR depuis Ganache sur le champ "Web3 Provider Endpoint" (Figure 18), on pourra utiliser les comptes disponibles sur Ganache sur Remix-Ethereum.

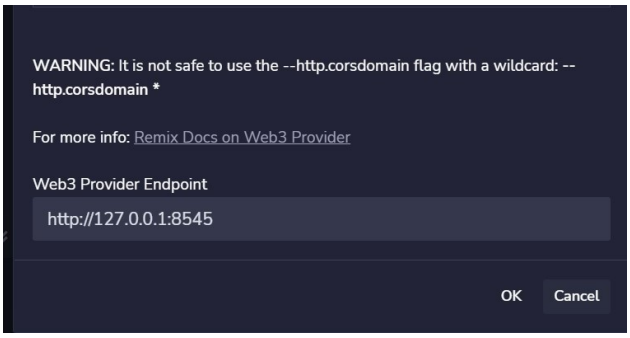

Figure 18: Requête d'ajout d'un noeud externe

6. À la fin de cette étape, l'adresse du contrat sera générée. Copier celle-ci pour initialiser la variable myContractAddress dans les fichiers Javascript. (Figure 19)

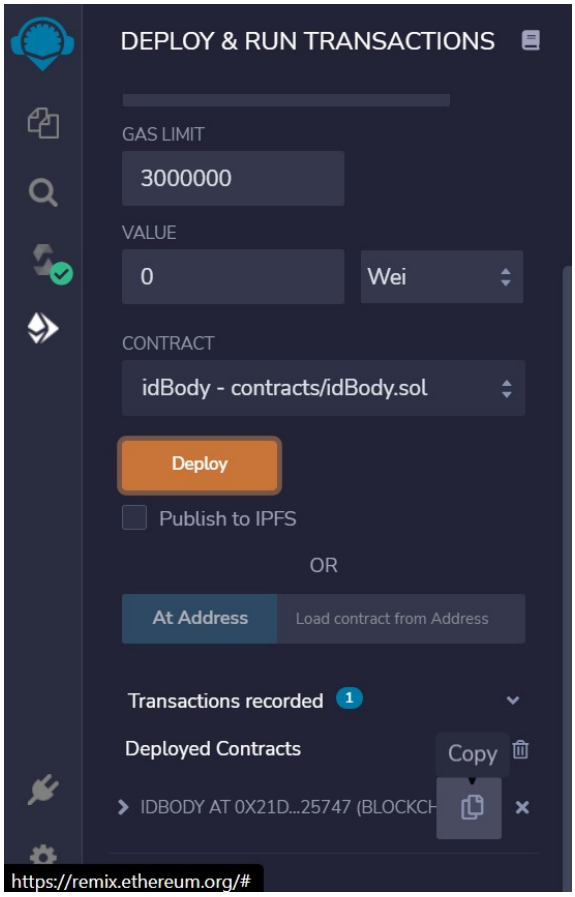

Figure 19: Rubrique "Déployer et Exécuter les transaction" sur Remix-Ethereum

7. Modifier l'ABI, l'adresse du compte utilisateur et l'adresse du contrat dans les fichiers Javascript. (Figure 20)

```
const userAccount = "0xEb057b79BEEC627559B7C87De8323a464D9405fF";

/*-----------------------------------------------------------------------------*/
function startApp() {
    var myContractAddress = "0xb1896B36fE4822Fe5259417f0DDEC482E5c6ae07";
    var myABI = JSON.parse(`[
    {
        "inputs": [],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "addr",
                "type": "address"
            }
        ],
        "name": "addressTostring",
        "outputs": [
            {
                "internalType": "string",
                "name": "",
                "type": "string"
```

Figure 20: Fichier index.js

8. Exécuter les fichiers Javascript en cliquant sur run. On exécute d'abord le

fichier index.js, puis le fichier network.js. (Figure 21)

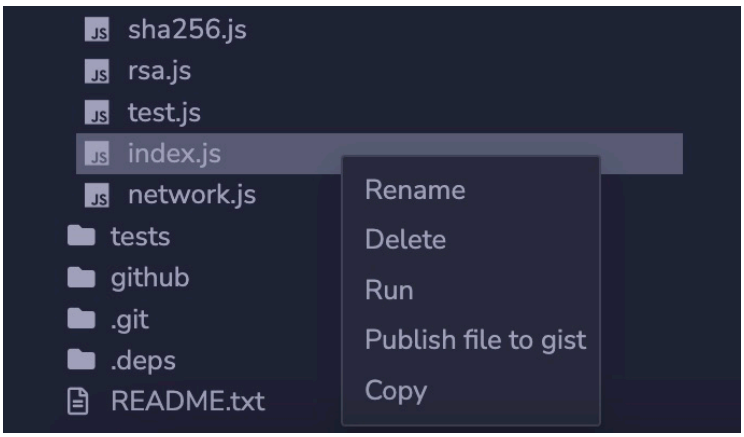

Figure 21: Explorateur de fichiers sur Remix-Ethereum

9. Finalement, la Figure 22 représente le certificat obtenu après avoir rentré les données de l'utilisateur. On peut notamment y retrouver la signature électronique de l'utilisateur, ce qui prouve qu'il a bien été authentifié et ses données sont désormais stockées dans le contrat intelligent.

```
Validated authentication !
Display the user's certificate :
Smart Contract address : 0xea9D0688B39e8b25589Af25014924604c414DEe4
User public key : 44049658924710850168290751168099379067835205214271526574485818573166303392512
User name : Tom
User E-mail : tom.olivier@gmail.com
User data : Ceci sont des données personnelles de Tom.
Network Time Stamp : 1652558815
User Time Stamp : 1652557956
Network data nounce : 64254
User data nounce : 34663320647
Network signature : 0x677735917bc824a5a2ed1bef6c66d52e29cb899368bcd797f2be4abb76
User signature : 0xb369bc6e0cd2ed4e3028ef968684d1d9434b10bc977824c53c95a3a64b
```

Figure 22: Certificat x.509 de l'utilisateur

8 Conclusion

Nous avons ainsi réussi à générer une interface utilisateur qui nous permet de stocker l'information du client de manière décentralisée sous la forme d'un certificat x.509 sur la blockchain à l'aide d'un contrat intelligent. Les techniques de cryptographie utilisées rendent ces données plus sécurisées et fiables, offre un gain de temps, une baisse des coût grâce à l'exécution automatique du contrat intelligent et diminuent le risque d'erreur.

Ces contrats pourraient révolutionner les relations contractuelles dans la mesure où le caractère automatique de celui-ci pourrait entraîner la suppression du contrat traditionnel. Leur potentiel va au-delà du simple transfert d'actifs, il peut y avoir plusieurs contrats intelligents liés les uns aux autres pour fournir des services entre eux.

Finalement, ce projet nous a permis de nous familiariser avec le domaine de la recherche et de la blockchain qui nous étaient inconnus jusqu'à maintenant, de travailler en équipe, et enfin de relever un challenge sur des problématiques concrètes.

References

[1] A Pure-solidity implementation of the SHA1 hash function by Nick Johnson. https://github.com/ensdomains/solsha1. Visité le 15/02/2022.

[2] Simon Pontie.Simon Pontie. Architecture d'un crypto processeur ECC sécurisé contre les attaques physiques. Journées Nationales du Réseau Doctoral en Microélectronique (JNRDM'14), May 2014, Lille, France. pp.4. ffhal-01091030f https://hal.archives-ouvertes.fr/hal-01091030/document. Visité le 25/02/2022.

[3] A native js function for hashing messages with the SHA-1 algorithm. https://github.com/pvorb/node-sha1. Visité le 16/02/2022.

[4] « Cryptographie - Certificats » issu de l'encyclopédie informatique « Comment Ça Marche ». www.commentcamarche.net. Visité le 28/02/2022.

[5] Secure Hash Algorithms. Brilliant.org. Retrieved 17:22, March 14, 2022, from https://brilliant.org/wiki/secure-hashing-algorithms/. Visité le 23/01/2022.

[6] Qu'est-ce qu'un contrat intelligent? Coinbase https://www.coinbase.com/fr/learn/crypto-basics/what-is-a-smart-contract. Visité le 14/05/2022.

[7] Chiffrement RSA. Wikipedia https://fr.wikipedia.org/wiki/Chiffrement_RSA. Visité le 14/05/2022.

[8] Comment fonctionne la signature numérique? https://blogs.pme.ch/massimo-musumeci/2020/08/06/comment-fonctionne-signature-numerique/. Visité le 14/05/2022.