Deploying PKI-Based Identity with Blockchain

DPKI Blockchain Implementations

Étapes détaillées pour configurer, déployer, et tester le DPKI.

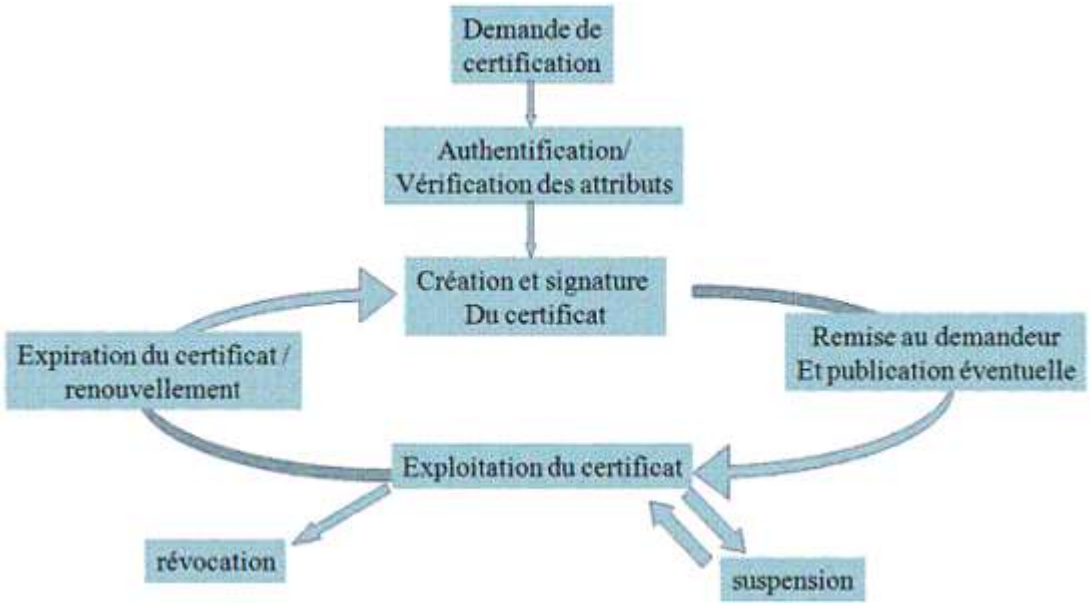
Smart Contract DPKI

Nous avons choisi d'utiliser la version 8.0.0 de Solidity pour le développement du Contract intelligent.
Pour plus de détails sur le langage, vous pouvez consulter : https://docs.soliditylang.org/en/v0.8.0/.

Contract Intelligent DPKI

Objectif du Contract

- Gérer des certificats numériques de manière décentralisée pour valider des clés publiques de manière sécurisée.
- Fournir des fonctionnalités d'enregistrement, de révocation et de vérification.

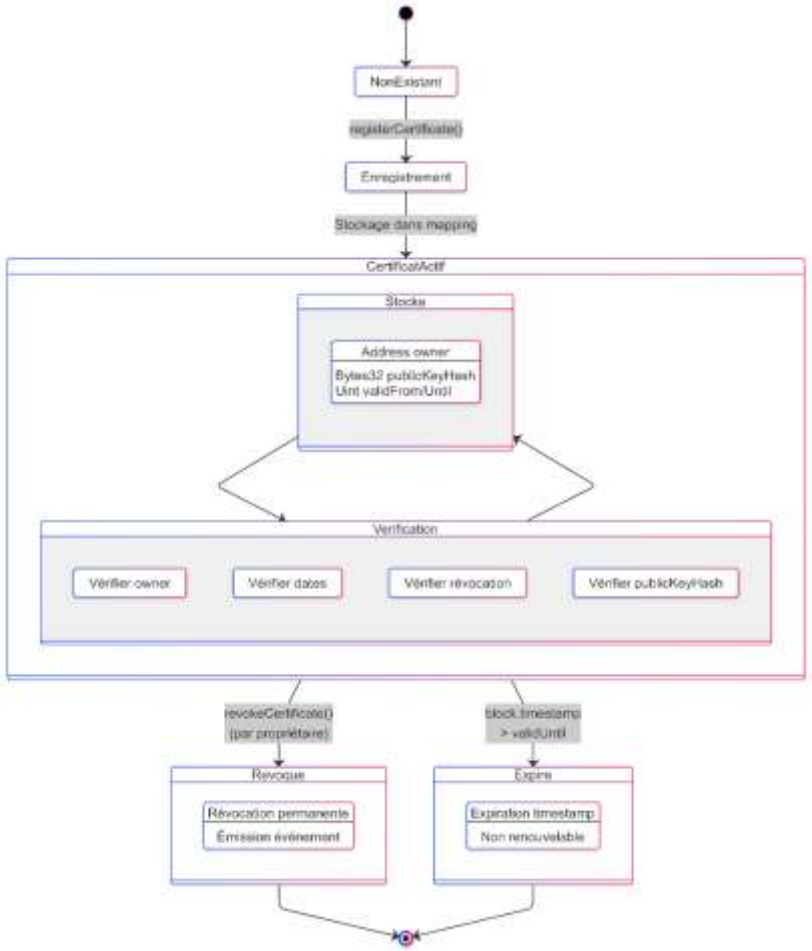


Cycle de vie d'un certificat numérique

Structure du Contract

Notre *smart contract* DPKI garantit une gestion sécurisée et fiable des certificats numériques.

- Le processus commence à l'état **NonExistant**, où aucun certificat n'existe.
- La fonction registerCertificate() permet de créer un certificat, passant par l'état **Enregistrement** pour arriver à l'état **CertificatActif**.
- À l'état **CertificatActif**, le certificat contient des informations essentielles : l'adresse du propriétaire, le hash de la clé publique et les périodes de validité. Des **vérifications** s'assurent de l'identité du propriétaire, de la validité temporelle, de l'absence de révocation et de la correspondance de la clé publique.
- Deux états finaux sont possibles :
  - **Révoqué**, déclenché volontairement par le propriétaire, irréversible et accompagné d'un événement.
  - **Expiré**, qui survient automatiquement à la fin de la validité et ne peut pas être renouvelé.



Structure du Contract

Dans cette partie, nous avons exploré le code du *smart contract* DPKI, en détaillant les étapes principales comme l'enregistrement, la vérification et la révocation des certificats.

- **Certificate** : une structure contient les informations essentielles d'un certificat (propriétaire, clé publique, période de validité, etc.).
- **Certificates:** un mapping permet de retrouver chaque certificat via son identifiant unique (hash).
- **CertificateRegistered** et **CertificateRevoked :** des événements pour notifier les actions importantes effectuées sur la blockchain.

```
struct Certificate {
    address owner; // Propriétaire du certificat (adresse Ethereum)
    bytes32 publicKeyHash; // Hash de la clé publique associée
    uint256 validFrom; // Date de début de validité du certificat
    uint256 validUntil; // Date de fin de validité du certificat
    bool isRevoked; // Indicateur si le certificat est révoqué
}

// Mapping permettant d'associer chaque certificat à son ID unique (hash)
mapping(bytes32 => Certificate) public certificates;

// Événements permettant de notifier les actions importantes sur la blockchain
event CertificateRegistered(bytes32 indexed certId, address owner, bytes32 publicKeyHash);
event CertificateRevoked(bytes32 indexed certId);
```

[5ma1l/dpki-project]

Explication du Contract Intelligent DPKI

Structure du Contract

Dans cette partie, nous avons exploré le code du *smart contract* DPKI, en détaillant les étapes principales comme l'enregistrement, la vérification et la révocation des certificats.

registerCertificate : Permet à un utilisateur de créer un certificat.

```
// Fonction pour enregistrer un certificat
function registerCertificate(bytes32 certId, bytes32 publicKeyHash, uint256 validityPeriod) public {
    // Vérifie que le certificat n'existe pas déjà
    require(certificates[certId].owner == address(0), "Certificate ID already exists");

    // Enregistre le certificat avec les informations fournies
    certificates[certId] = Certificate({
        owner: msg.sender, // Propriétaire du certificat
        publicKeyHash: publicKeyHash, // Hash de la clé publique
        validFrom: block.timestamp, // Date actuelle comme date de début
        validUntil: block.timestamp + validityPeriod, // Période de validité
        isRevoked: false // Le certificat n'est pas révoqué au moment de l'enregistrement
    });

    // Émet un événement pour notifier l'enregistrement du certificat
    emit CertificateRegistered(certId, msg.sender, publicKeyHash);
}
```

5ma1l/dpki-project

Explication du Contract Intelligent DPKI

Structure du Contract

Dans cette partie, nous avons exploré le code du *smart contract* DPKI, en détaillant les étapes principales comme l'enregistrement, la vérification et la révocation des certificats.

revokeCertificate : Révoque un certificat existant.

```
// Fonction pour révoquer un certificat
function revokeCertificate(bytes32 certId) public {
    // Vérifie que l'appelant est le propriétaire du certificat
    require(certificates[certId].owner == msg.sender, "Not certificate owner");
    require(!certificates[certId].isRevoked, "Certificate already revoked");

    // Marque le certificat comme révoqué
    certificates[certId].isRevoked = true;
    // Émet un événement pour notifier la révocation du certificat
    emit CertificateRevoked(certId);
}
```

5ma1l/dpki-project

Explication du Contract Intelligent DPKI

Structure du Contract

Dans cette partie, nous avons exploré le code du *smart contract* DPKI, en détaillant les étapes principales comme l'enregistrement, la vérification et la révocation des certificats.

verifyCertificate : Vérifie la validité d'un certificat donné.

```
// Fonction pour vérifier la validité d'un certificat
function verifyCertificate(bytes32 certId, bytes32 publicKeyHash) public view returns (bool) {
    Certificate memory cert = certificates[certId];
    // Vérifie que le certificat existe, que la clé publique correspond, qu'il est valide et non révoqué
    return (
        cert.owner != address(0) &&
        cert.publicKeyHash == publicKeyHash &&
        cert.validUntil >= block.timestamp &&
        !cert.isRevoked
    );
}
```

5ma1l/dpki-project

Déploiement du Contract

Préparation de l'environnement

Pour déployer le Smart Contract, nous avons besoin, tout d'abord, de préparer l'environnement.

Étape 1 : Installer les outils nécessaires

Node.js : un environnement d'exécution JavaScript côté serveur.

```
sudo apt install nodejs
```

Truffle : un framework de développement pour Ethereum qui simplifie la création, le test et le déploiement de Contracts intelligents. Il fournit des outils puissants pour interagir avec la blockchain Ethereum.

```
npm install -g truffle
```

Ganache : un simulateur de blockchain local, ce qui signifie qu'il vous permet de créer une blockchain Ethereum personnelle pour tester vos Contracts intelligents sans utiliser la blockchain principale.

```
npm install -g ganache-cli
```

Préparation de l'environnement

Pour déployer le Smart Contract, nous avons besoin, tout d'abord, de préparer l'environnement.

Étape 2 : Créer le projet Truffle

Créer un nouveau répertoire pour le projet et initialise un projet Truffle avec des fichiers par défaut pour travailler avec les Contracts intelligents.

```
nooby at nobody in ~
○ mkdir dpki-project


nooby at nobody in ~
○ cd dpki-project


nooby at nobody in ~/dpki-project
○ truffle init


Starting init...
================

> Copying project files to /home/nooby/dpki-project

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
```

Préparation de l'environnement

Pour déployer le Smart Contract, nous avons besoin, tout d'abord, de préparer l'environnement.

Étape 3 : Créer le Contract DPKI.sol

Le Contract **DPKI.sol** est le cœur du projet. Il contient la logique qui régit l'enregistrement, la vérification et la révocation des certificats.

- Instructions :
  - Dans le répertoire **contracts**, créez un fichier nommé **DPKI.sol**.
  - Ajoutez le code du Smart Contract.

```
pragma solidity ^0.8.0;

contract DPKI {
    struct Certificate {
        address owner;
        bytes32 publicKeyHash;
        uint256 validFrom;
        uint256 validUntil;
        bool isRevoked;
    }

    mapping(bytes32 => Certificate) public certificates;

    event CertificateRegistered(bytes32 indexed certId, address owner, bytes32 publicKeyHash);
    event CertificateRevoked(bytes32 indexed certId);

    function registerCertificate(bytes32 certId, bytes32 publicKeyHash, uint256 validityPeriod) public {
        require(certificates[certId].owner == address(0), "Certificate ID already exists");

        certificates[certId] = Certificate({
            owner: msg.sender,
            publicKeyHash: publicKeyHash,
            validFrom: block.timestamp,
            validUntil: block.timestamp + validityPeriod,
            isRevoked: false
        });

"contracts/dpki.sol" 48 lines --2%--
```

Le projet complet est dans 5ma1l/dpki-project.

Préparation de l'environnement

Pour déployer le Smart Contract, nous avons besoin, tout d'abord, de préparer l'environnement.

Étape 4 : Créer le fichier de migration

Truffle utilise **des fichiers de migration** pour déployer les Smart Contracts sur un réseau blockchain (comme Ganache).

- Instructions :
  - Dans le répertoire **migrations**, créez un fichier de migration nommé **2_deploy_dpki.js**.
  - Ajoutez le code suivant pour déployer le Contract sur Ganache.

```
1  const DPKI = artifacts.require("DPKI");
2
3  module.exports = function(deployer) {
4    deployer.deploy(DPKI);
5  };
```

Le projet complet est dans 5ma1l/dpki-project.

Préparation de l'environnement

Pour déployer le Smart Contract, nous avons besoin, tout d'abord, de préparer l'environnement.

Étape 5 : Configurer Truffle pour se connecter à Ganache

Truffle doit savoir comment se connecter à **Ganache** pour déployer et interagir avec le **Contract**.

- Instructions :
  - Ouvrez le fichier **truffle-config.js**.
  - Ajoutez la configuration suivante pour connecter **Truffle** à **Ganache.**

```
module.exports = {
    networks: {
      development: {
        host: "127.0.0.1", // Adresse de Ganache
        port: 8545, // Port de Ganache
        network_id: "*" // Utilise le réseau local
      }
    },
    compilers: {
      solc: {
        version: "0.8.0" // Spécifie la version de Solidity à utiliser
      }
    }
};
```

Le projet complet est dans 5ma1l/dpki-project.

Préparation de l'environnement

Pour déployer le Smart Contract, nous avons besoin, tout d'abord, de préparer l'environnement.

Étape 6 : Lancer Ganache

Dans le terminal, exécutez la commande suivante pour démarrer Ganache :

```
nooby at nobody in ~/dpki-project
○ ganache-cli
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
==================
(0) 0xBe7D774974C5121B9b079a80A2C393D07D580Fa7 (100 ETH)
(1) 0x1C497fFA69C228E0036571Ac4518B45Ec1456931 (100 ETH)
(2) 0x3c2e62725217cD24CD7d9f943811fd19728a5eA2 (100 ETH)
(3) 0x9Fb5379aAd7fa6B289fc5166952587d282c35290 (100 ETH)
(4) 0xeFe9Be8995F0246b3cBAFcbbe490d7d6b30da71a (100 ETH)
(5) 0x1C0ee36219e624780411e4dE51D1C582f9cF6184 (100 ETH)
(6) 0xDb8EFb732Da21E7e769913cB65FEF27e1E9DF3D9 (100 ETH)
(7) 0xAdD595D128a976Ef8Dc1f1b53e6Fb34984240387 (100 ETH)
(8) 0x45bcdBF8070CDe9A58Ea91B1497e420520a5c830 (100 ETH)
(9) 0x3744261Bb7763a797a9Ef3a30cF523aee9ba6C23 (100 ETH)

Private Keys
==================
(0) 0x7bac113a765b7e00e455b6509e2f2aba7170f01a8a8abcea79683f5620866457
(1) 0x2a6ad7b4b178624aa7acd56192af5cd912c081f2646cd6f160d2e8a16a9e2525
(2) 0x107d2853612a32ff5254e5e3f9d7c76dda612369205a324f1064cec5b216cc77
```

Déploiement du Contract

Étape 7 : Déployer sur Ganache

Dans le terminal, déployez le Contract sur Ganache avec la commande suivante :

```
nooby at nobody in ~/dpki-project
○ truffle migrate --network development


Compiling your contracts...
===========================
> Compiling ./contracts/dpki.sol
> Compilation warnings encountered:

    Warning: SPDX license identifier not provided in source file. Before publishing, c
X-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifie
 Please see https://spdx.org for more information.
--> project:/contracts/dpki.sol


> Artifacts written to /home/nooby/dpki-project/build/contracts
> Compiled successfully using:
   - solc: 0.8.0+commit.c7dfd78e.Emscripten.clang


Starting migrations...
======================
> Network name:    'development'
> Network id:      1732446520902
> Block gas limit: 6721975 (0x6691b7)
```

Tester le Contract

Nous avons proposé le scénario suivant : créer les clés privées et publiques, puis enregistrer, vérifier, et enfin révoquer.

Tester le Contract

Étape 1 : Génération des Clés Privée et Publique

Dans le terminal, exécutez la commande suivante pour générer les clés :

```
nooby at nobody in ~/dpki-project
○ openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048

nooby at nobody in ~/dpki-project
○ openssl rsa -pubout -in private_key.pem -out public_key.pem

writing RSA key
```

Étape 2 : Hachage de la clé publique

Nous avons haché la clé publique pour nous préparer à la phase de test, à l'aide de la commande suivante :

```
nooby at nobody in ~/dpki-project
○ PUBKEY_HASH=$(openssl dgst -sha256 -binary public_key.pem | xxd -p -c 32)


nooby at nobody in ~/dpki-project
○ echo "Public Key Hash: 0x$PUBKEY_HASH"

Public Key Hash: 0x9b793248894ec6ec56a60e3cc272286481a8bb70830e19cfdb8b03cec9e80944
```

Tester le Contract

Étape 3 : Accéder à la Truffle Console

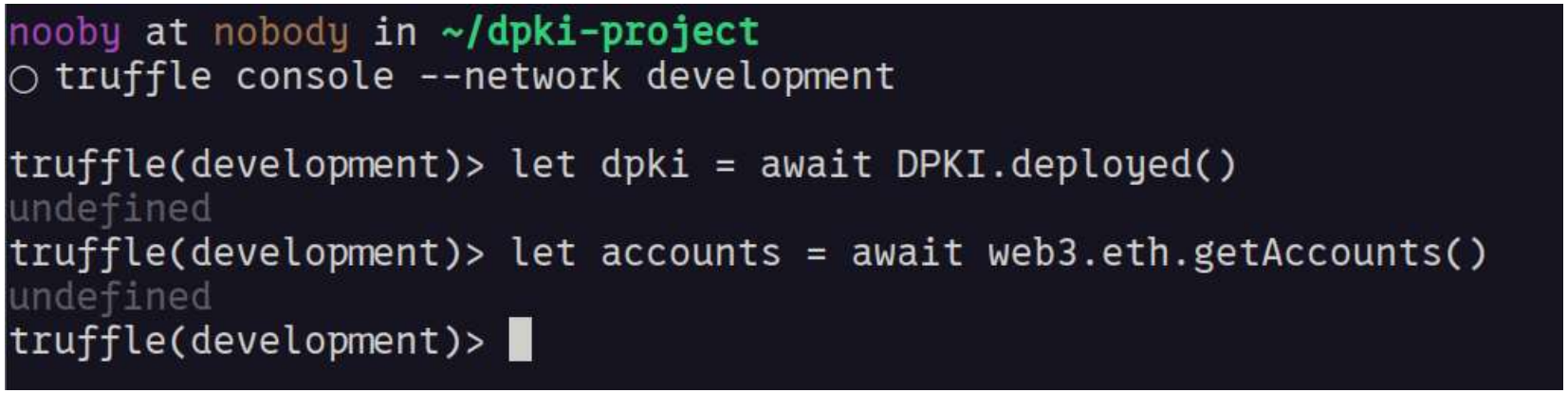
Truffle Console nous permet d'interagir avec votre Contract déployé sur Ganache.

Étape 4 : Récupérer l'instance du Contract déployé

Avant de pouvoir appeler des fonctions de Contract, nous devons obtenir une instance du Contract déployé.

Étape 5 : Récupérer les comptes Ethereum disponibles

Nous avons besoin des comptes disponibles pour effectuer des transactions (comme enregistrer un certificat ou révoquer un certificat).



Tester le Contract

Étape 6 : Enregistrer un Certificat

Nous avons enregistré un certificat sur la blockchain en utilisant un identifiant unique (certId) et le hash de clé publique (étape 2).

```
truffle(development)> let certId = web3.utils.sha3("test1")
undefined
truffle(development)> let publicKeyHash = "0x9b793248894ec6ec56a60e3cc272286481a8bb70830e19cfdb8b03cec9e80944"
undefined
truffle(development)> let validityPeriod = 365 * 24 * 60 * 60
undefined
truffle(development)> await dpki.registerCertificate(certId, publicKeyHash, validityPeriod)
{
  tx: '0xacd9650e667b5df487b71c86129e14c7dd8b2eff1f8974d8cf129d2b58e5f06e',
```

Étape 7 : Vérification du Certificat

Nous avons vérifié si un certificat est valide, en utilisant l'ID du certificat et le hachage de la clé publique

```
truffle(development)> let isValid = await dpki.verifyCertificate(certId, publicKeyHash)

truffle(development)> console.log("Certificate is valid:", isValid)
Certificate is valid: true
undefined
```

Tester le Contract

Étape 8 : Révocation d'un Certificat

Si un certificat est compromis ou n'est plus valide, il peut être révoqué pour empêcher son utilisation.

```
truffle(development)> await dpki.revokeCertificate(certId)
{
  tx: '0xb8ab995a3f44e2fbec3b3e10cc186687c1f76970cbd4556a813edb4c411535f0',
  receipt: {
    transactionHash: '0xb8ab995a3f44e2fbec3b3e10cc186687c1f76970cbd4556a813edb4c411535f0',
```

Étape 9 : Vérification du Certificat

Nous avons vérifié si le certificat est revoqué, en utilisant l'ID du certificat.

```
truffle(development)> let cert = await dpki.certificates(certId)
undefined
truffle(development)> console.log("Certificate is revoked:", cert.isRevoked)
Certificate is revoked: true
```