Aix*Marseille université

01/10/2017

Projet Blockchain SXP

Master 1 Informatique - Génie Logiciel - Projet Phase 1

Doulkifouli ABDALLAH-ALI
Jalal RABAHI
Trang NGUYEN NGOC BAO
Charly LAMOTHE

Table des matières

Présentation du sujet

Notre client est une communauté hippie-cool qui souhaite utiliser les avantages de l'informatique qui lui permettrait de mieux gérer ses affaires (troc, ...).

Une SSII a été contactée et a développé un client P2P pour satisfaire cette demande. Sur ce client, il est possible d'effectuer différentes actions comme créer un profil, publier des offres, formuler ses demandes ou rechercher parmi celle des autres. Pour garantir la confidentialité des données, ces dernières sont signées et encryptées.

Le prototype qui remplit cette fonction se nomme Secure eXchange Protocol (abrégé en SXP) et est découpé en modules très indépendants. Ils interagissent à travers leurs APIs, toutes documentées par un wiki.

Présentation de notre projet

L'objectif du projet est de travailler sur le module de signature de contrat en utilisant la méthode Blockchain.

Actuellement, la signature d'un contrat entre différents hôtes se fait en utilisant la méthode Sigma. Comme alternative, notre client souhaite utiliser la méthode Blockchain, en utilisant Ethereum et ses SmartContract.

L'année précédente, un groupe y a déjà réfléchi. Le résultat de leur travail est la signature d'un contrat de test entre deux utilisateurs.

Dans la pratique, notre travail consistera de comprendre leur travail, reproduire leurs tests, améliorer le protocole en réfléchissant plus en profondeur aux SmartContract, et préparer l'intégration à SPX, en rendant la synchronisation à la blockchain légère, tout en préservant les 80% de taux de couverture de code.

Cahier des charges

Note : Les documents UML et les algorithmes ont été rédigés en anglais, afin de rester dans cette langue le plus possible dans les parties techniques, étant donné que le projet SXP a été réalisé en anglais dans le but d'être plus accessible aux lecteurs.

Les objectifs ci-dessous correspondent à nos charges, à réaliser par ordre croissant. Il y en a sept, dont la dernière est en bonus, si nous avons assez de temps pour la réaliser.

- Gestion de compte Ethereum,
- Déploiement d'un contrat,
- Durée limite de la signature du contrat,
- Signature de contrats avec N pairs,
- Alléger la synchronisation avec la blockchain,
- Implémenter notre solution dans SXP,
- Bonus : Anonymiser les pairs durant la signature d'un contrat.

Charge	Description
Gestion de compte Ethereum	Afin qu'un client puisse utiliser la blockchain Ethereum comme protocole de signature de contrats, un compte Ethereum sera créé automatiquement. Ce compte génèrera une paire de clés. Ces clés devront être stockées de manière sécurisée dans la base de données.
Déploiement d'un contrat	Durant la signature d'un contrat, il est impératif de choisir de façon arbitraire l'ordre dans lequel le contrat est signé par les différents pairs. Après la signature de toutes les parties, on doit vérifier que le contrat est légitime.
Durée limite de la signature du contrat	Dans l'état actuel du code, un contrat attend à l'infini qu'il soit signé, ce qui pose un problème sémantique, ainsi qu'un problèmede sécurité. Pour pallier à cela, il faut ajouter une date d'expiration à chaque contrat, qui a été acceptée par tous les partis.
Signature de contrats avec N pairs	La multi signatures permet à l'utilisateur de réaliser une transaction avec plus d'un utilisateur. C'est approprié pour gérer les transferts dans les petits groupes ou organisations, ou pour des échanges multiples ; où plusieurs échanges à deux pairs sont réalisés à la suite afin de réaliser notre échange.

Alléger la synchronisation avec la blockchain	Par défaut, afin de travailler avec la blockchain, il faut posséder le fichier dans son intégralité. A terme, un utilisateur utilisant la méthode blockchain, doit pouvoir avoir un client léger.
Implémenter notre solution dans SXP	Une fois la méthode Ethereum fonctionnelle, une implémentation dans SXP doit être réalisée de manière transparente.
Bonus : Anonymiser les pairs durant la signature d'un contrat	Lors de la procédure de signature entre tous les clients, l'identité des personnes ayant signées précédemment ne peut pas être connue par l'utilisateur.

Spécifications

Charges 1 et 2 : Gestion de compte Ethereum et Deploiement d'un contrat

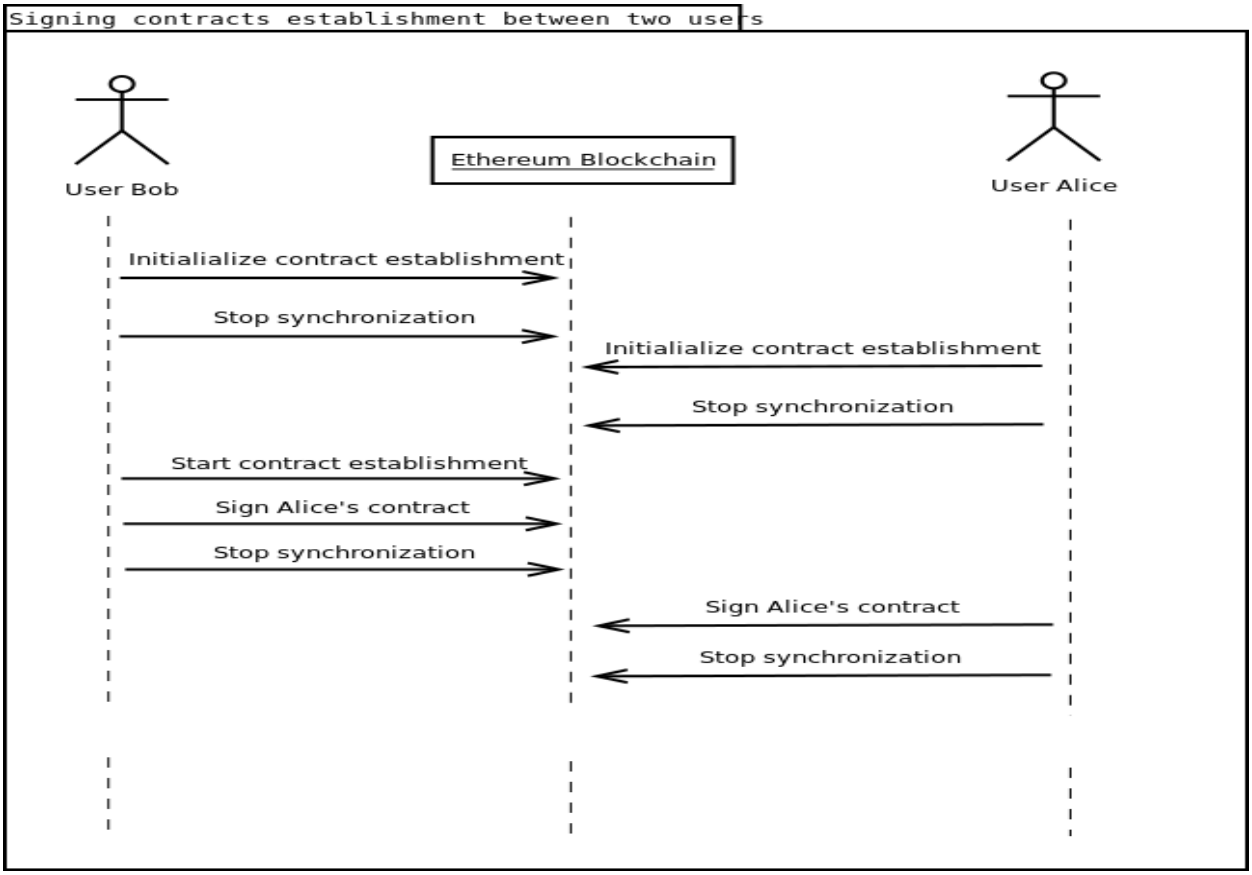
Actuellement, la classe de test BlockChainEstablisherTest peut signer les contrats entre deux utilisateurs en procédant en deux étapes :

- L'initialisation des deux contrats à partir des comptes Ethereum et des clauses de l'échange.
- La synchronisation avec la blockchain entre les deux contrats.

La première partie d'initialisation peut être exprimé par l'algorithme suivant :

```
procedure initialization (two Ethereum accounts) :
    Having two agreed users
    Init two users and set for each an Ethereum key
    Init two contract entities with user id et an contract clauses
    Create an uris map, with PK in key and url in value
    Create two blockchain contract, with with a contract entity and a set of
parties
    Create two block chain establisher with a user in uris map
```

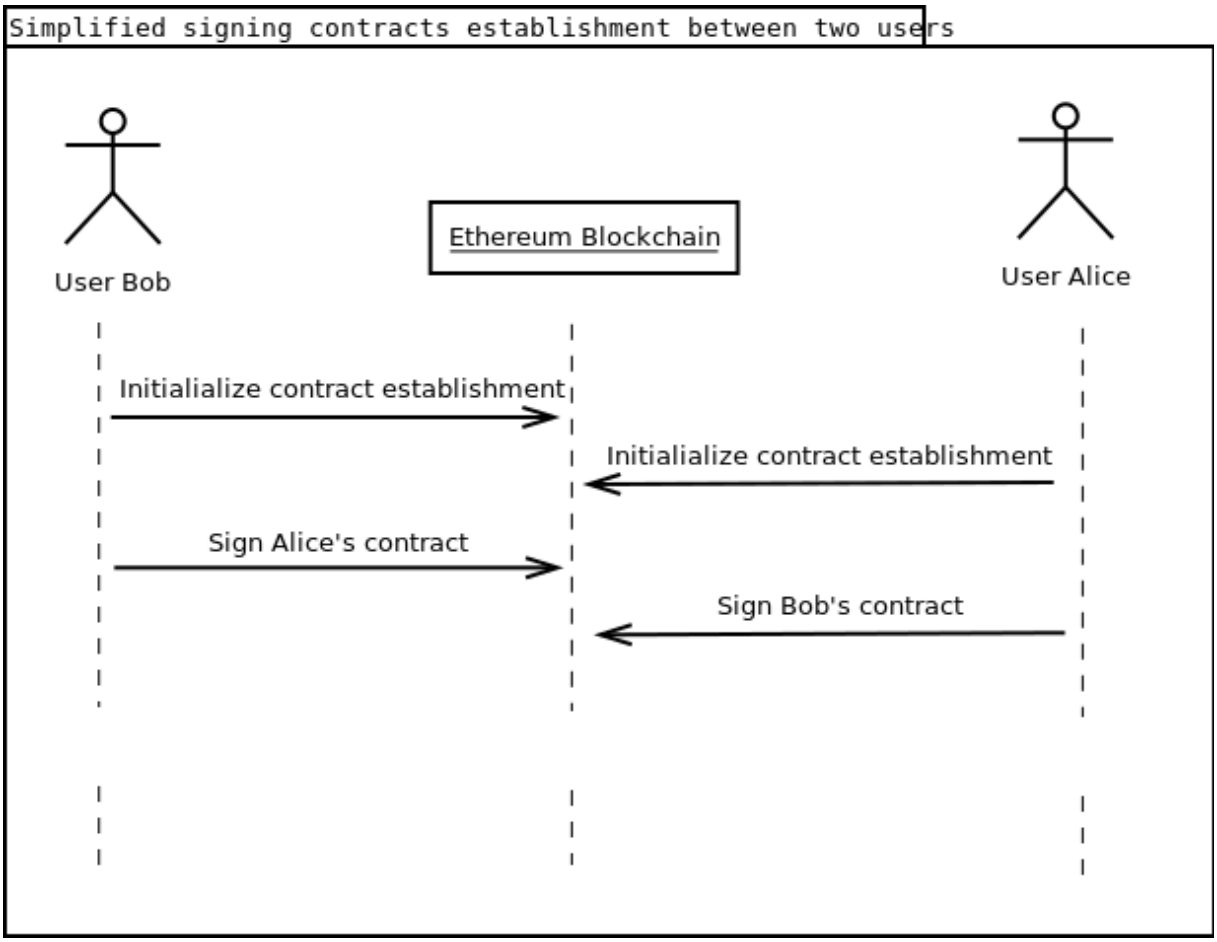
La deuxième partie, de synchronisation, peut être représenté avec le diagramme de séquences suivant :



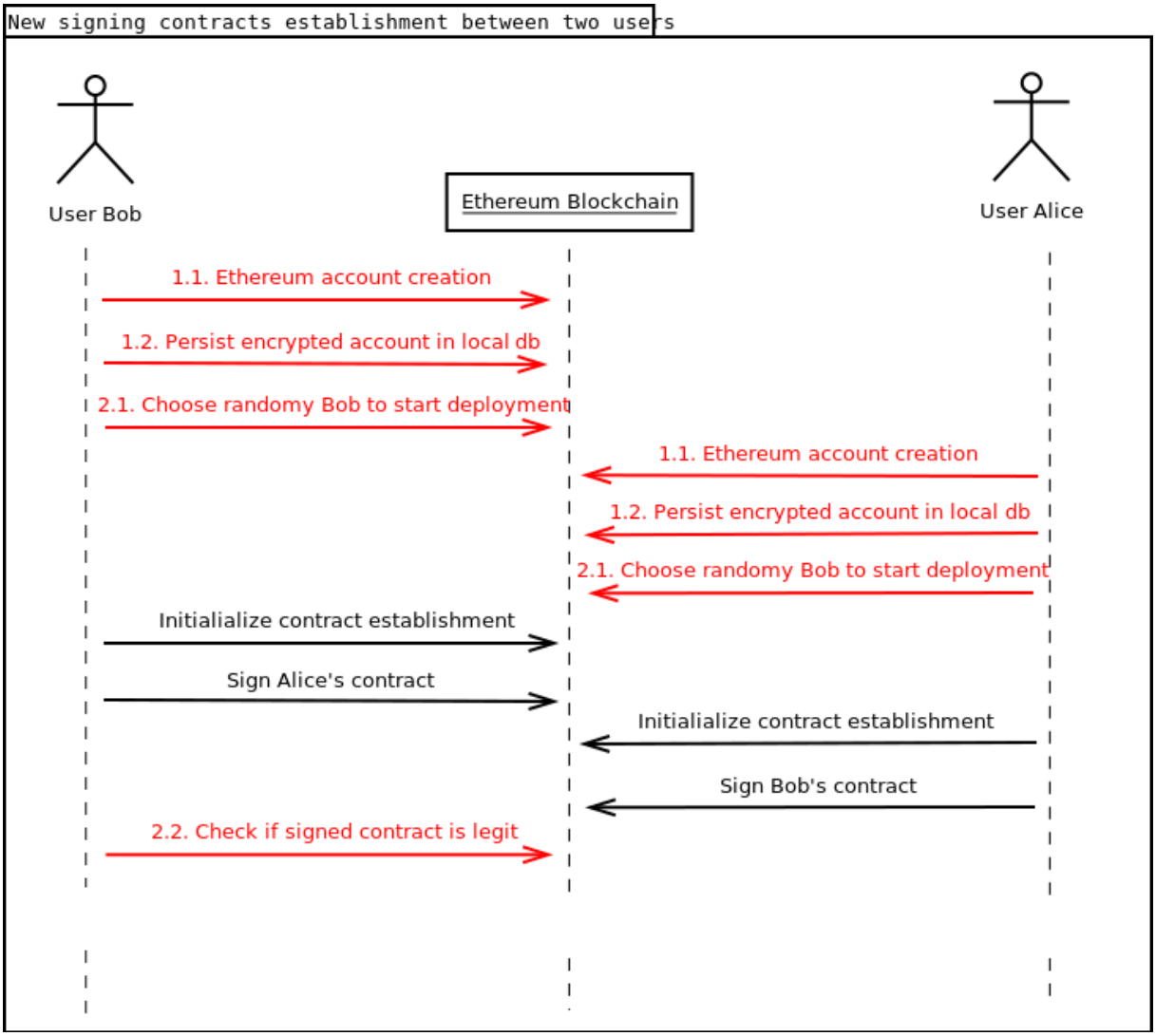
Remarques:

- Pour des raisons pratiques, les signatures des deux contrats sont réalisés dans le même fichier de test, et donc dans la même machine. Cependant, celà réduit la possibilité du nombre de pairs à 2.
- Etant donné que la signature de l'échange est effectué dans la même machine, il est nécessaire d'endormir le programme entre chaque requête. Celà n'est pas représenté sur le diagramme.

Ce diagramme peut être simplifié de la manière suivante :



En lui appliquant notre cahier des charges, le procéder est maintenant représenté par le diagramme de séquences suivant:



Les charges 1 et 2 ainsi représentées, nous savons à présent où dans le projet elles se situent dans le projet.

Après la création d'un compte Ethereum, les informations relatives à ce compte seront chiffrées dans la base de données en utilisant la clé publique générée par le mot de passe Utilisateur de SXP.

Pour choisir de manière arbitraire un client dans la liste des pairs du contrat, chaque pair calcule un nombre aléatoire. On fait la somme de ces nombres ; puis, on applique un modulo sur le nombre de pairs du contrat.

Charge 3 : Durée limite de la signature du contrat

Afin de mesurer la date d'expiration d'un contrat, utiliser une date en *timestamp* n'est pas une option envisageable, parce que cela poserait des problèmes de synchronisation entre les différents fuseaux horaires. Une solution est de compter le nombre de blocs écrits dans la blockchain depuis le début du déploiement du contrat.

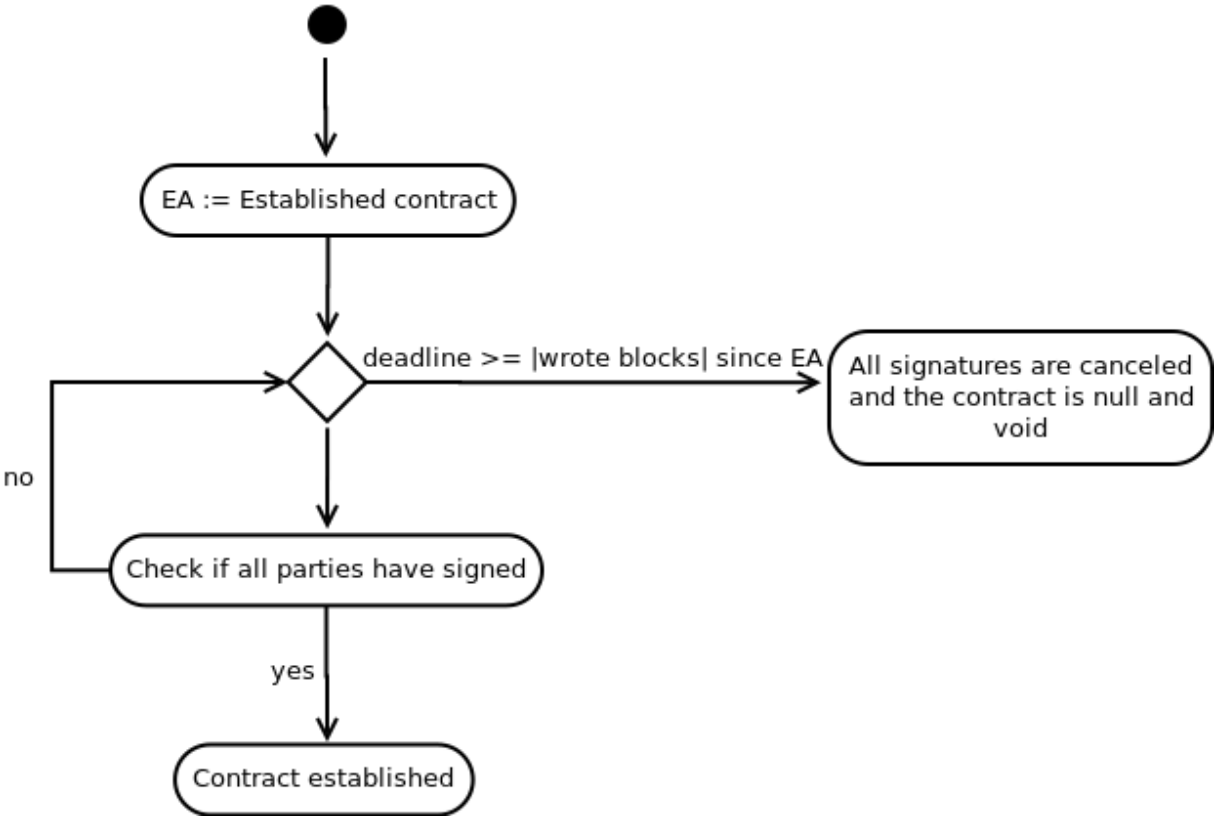
À la création d'un contrat, ont créé une variable pour la signature du contrat, représentant un nombre de blocs qu'on initialise avec un nombre calculé à partir du nombre de blocs lus depuis le dépoilement de notre contrat. Cette variable pourra être modifiée au cours du temps. Sa valeur est égale au nombre de blocs multiplié par la durée d'expiration de la signature du contrat. Par exemple, si 20 blocs ont été écrit en une heure dans la blockchain pour une durée d'expiration d'une heure, on comptera 20 blocs durant la signature du contrat.

Ceci peut être exprimé par l'extrait de code en Solidity suivant :

```
if (c.deadline <= block.number) {
        uint j = 0;
        uint n = c.numFunders;
        c.beneficiary = 0;
        c.fundingGoal = 0;
        c.numFunders = 0;
        c.deadline = 0;
        c.amount = 0;
        while (j <= n){
            c.funders[j].addr.send(c.funders[j].amount);
            c.funders[j].addr = 0;
            c.funders[j].amount = 0;
            j++;
        }
        return true;
    }
    return false;
}
```

Source : https://dappsforbeginners.wordpress.com/tutorials/contracts-that-send-transactions/

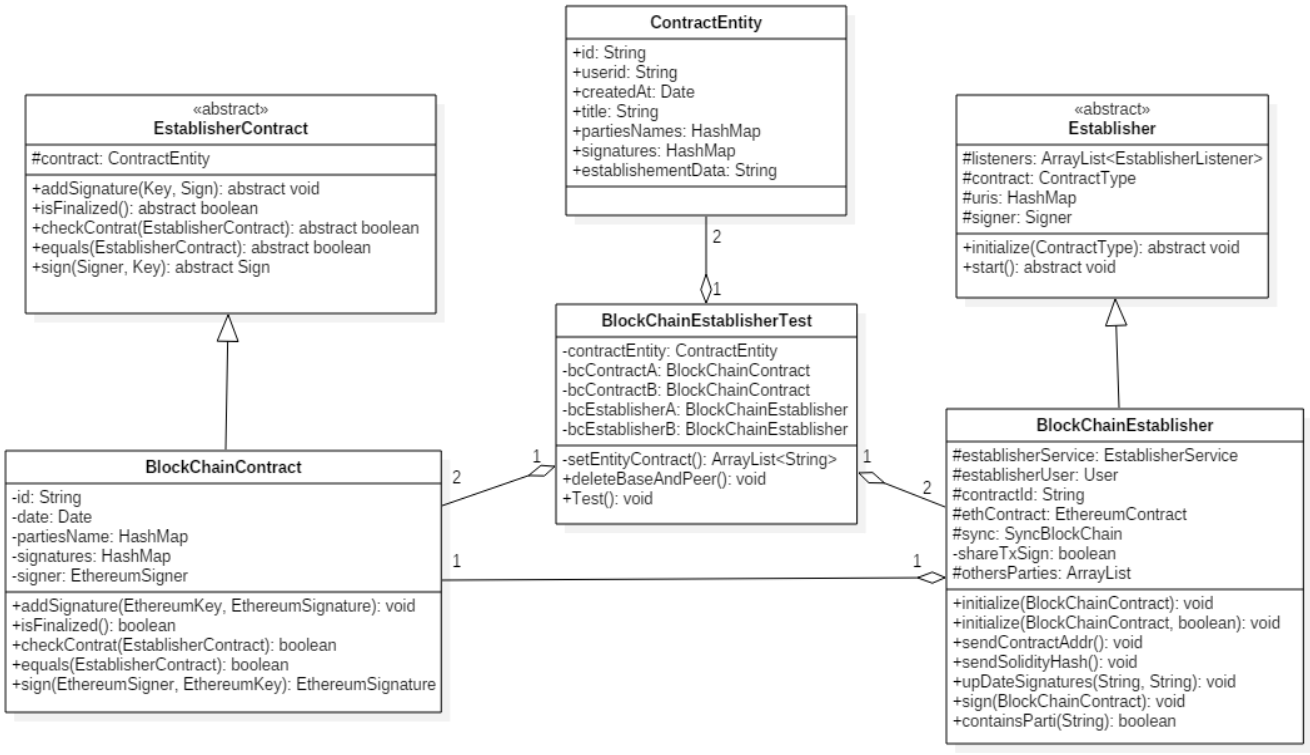
La date d'expiration d'un contrat en attente d'être signé, utilisant la méthode du comptage de blocs, peut être représentée par le diagramme d'activité suivant :



Le contrat est établie ;
Si la *deadline* (nombre de blocs max représentant le temps max souhaité) n'est pas respecté : toutes les signatures sont annulées et le contrat devient nul et non avenu ;
Sinon si tous les parties ont signés le contrat : le contrat est correctement établi, et appliqué ;
Sinon boucler.

Charge 4 : Signature de contrat avec N pairs

Voici l'architecture des classes utilisées pour tester l'établissement des signatures des contrats entre tous les partis. La classe de test BlockChainEstablisherTest en est le pilier (cf. les diagrammes de séquences plus haut).



Durant notre futur développement, nous comptant utiliser ces classes, en dérivant notre code à partir de la classe BlockChainEstablisherTest, dont son contenu nous sert d'exemple.

Remarque :
Les multiplicités sont toutes à 2 exclusif, étant donné que la classe de test ne nous permet que ce nombre de signataires pour l'instant.
Nous allons donc utiliser les mêmes classes comme base, et ainsi changer les multiplicités à 2..*.

Charge 5 : Alléger la synchronisation avec la blockchain

Une des clés de la sécurité de la blockchain est que chaque client à la même copie de la blockchain. Cependant, n'importe quelle machine ne peut télécharger un fichier de plusieurs GB. Une solution consite à utiliser le protocole client léger ou Light client protocol.

L'idée est de donner à un utilisateur léger, comme un smartphone, un ordinateur de bureau, etc. la possibilité de voir l'état d'un aspect d'Ethereum ou de vérifier l'execution d'une transaction.

Le principe consiste à utiliser la structure Ethereum Patricia Merkle tree pour déterminer l'entête d'un bloc pour trouver qui est le nœud où l'information que le client cherche est localisé.

- Dans le cas de la vérification de l'état d'un compte à une date donnée, on peut obtenir ses informations : solde, index, sa localisation etc... :

  - Solution proposée : Le client télécharge les entêtes des nœuds de l'arbre récursivement jusqu'à obtenir la valeur désirée.

- Le client veut vérifier la confirmation d'une transaction :

  - Solution proposée : Le client demande au réseau l'index et le numéro de bloc de la transaction et récursivement il télécharge les nœuds et vérifie si la transaction est disponible.

- Le client veut valider collectivement un bloc :

  - Solution proposée : Chaque client C[i] choisit une transaction T à l'adresse i, T[i] avec reçu correspondant, R[i], et procède ainsi :
    - Initie l'état de la racine à R[i].medstate et R[i-1].gas_used, si i=0
    - Le client ajoute la transaction choisi
    - Vérifie que le résultat donne bien la valeur dans la racine et le *gas* consommé correspond
    - Vérifie que les logs correspondent

- Le client veut regarder des événements qui ont été référencés :

  - Solution proposée :
    - Le client demande toutes les entêtes de blocs et filtre avec celui qu'il cherche,
    - Après l'avoir trouvé il peut télécharger la liste de toutes les transactions du bloc
    - Et enfin faire correspondre le log de la transaction et voir si elle est correcte

Source : https://github.com/ethereum/wiki/wiki/Light-client-protocol

Charge 7 : Anonymiser les pairs durant la signature d'un contrat

Afin d'anonymiser les pairs durant la signature du contrat, on peut utiliser un Solidity Contract utilisant un système de vote. Ainsi, chaque pair remarque que quelqu'un a signé sans pouvoir connaître l'identité de cette personne.

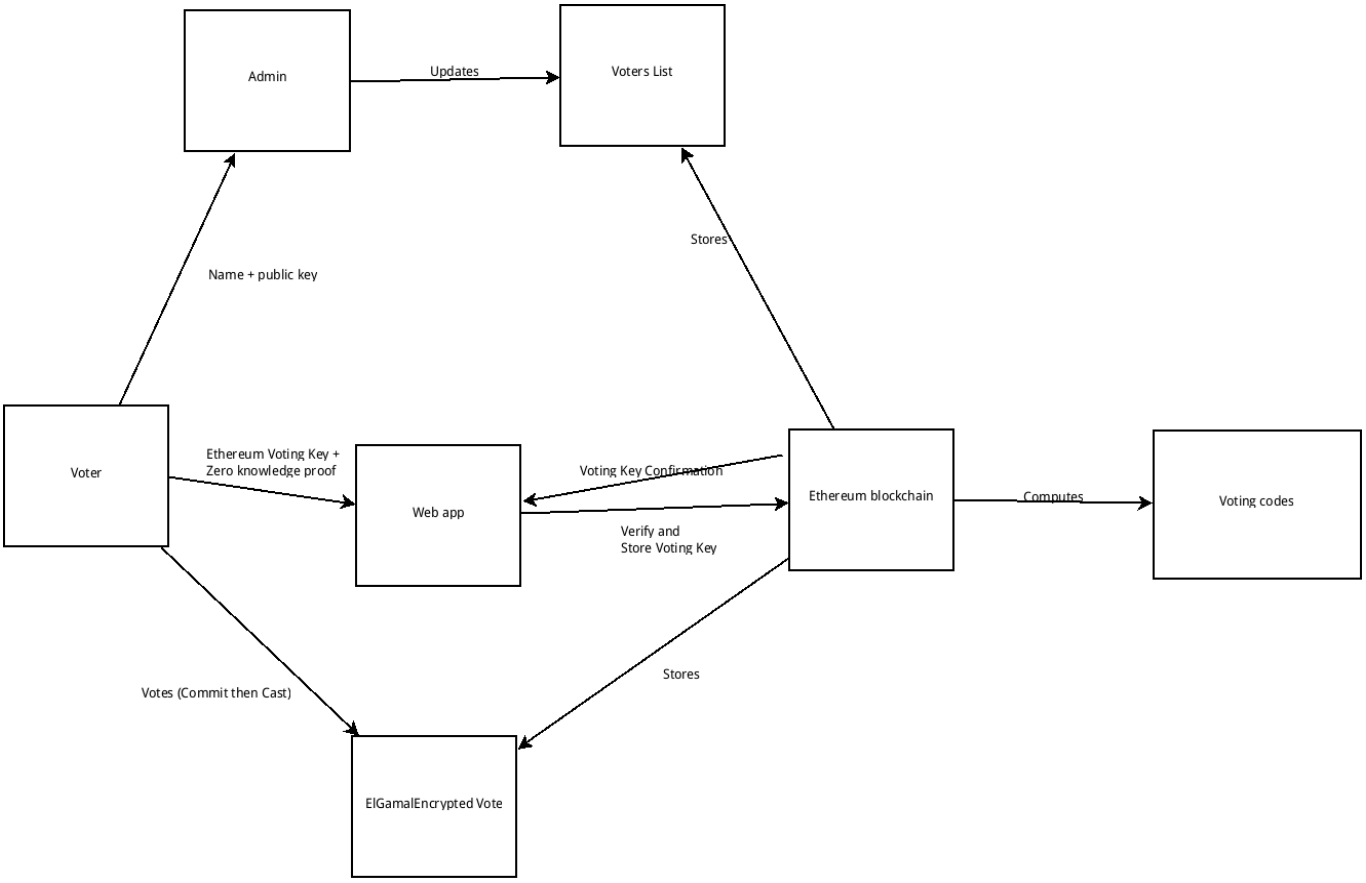
En 2009, F. Hao, P.Y.A. Ryan et P. Zielinski on amélioré l'algorithme de vote anonyme par tours, présenté dans le papier « Anonymous voting by two-round public discussion » : http://homepages.cs.ncl.ac.uk/feng.hao/files/OpenVote_IET.pdf

Une implémentation de cet algorithme a été réalisé par Patrick McCorry et est disponible ici : https://github.com/stonecoldpat/anonymousvoting

Le principe en est le suivant : Le « Open Vote Network (OV-net) » est un protocol de vote décentralisé en deux tours, contenant les fonctionnalités suivantes :

- Toute la communication est publique : pas de secret de session entre les voteurs n'est requis.
- Le système peut compter les votes lui-même : pas d'authorité nécessaire est requise pour procéder au dépouillement.
- La protection privé du votant est maximum.
- Le système est sans conflit - tout le monde peut vérifier si tous les électeurs agissent selon le protocole, ce qui garantit que le résultat est publiquement vérifiable.

Le workflow des votes peut être réprésenté ainsi :



Où :

- Admin est l'utilisateur responsable de l'envoie à Ethereum d'une liste blanche de voteurs éligibles.
- Voter est un utilisateur éligible pouvant voter
- Voters list est la liste des utilisateurs pouvant participer au vote
- Web app est un client JS recevant les résultats du vote
- Ethereum blockchain la base de données décentralisée
- ElGamalEncryptedVote représente un vote chiffré à l'avance afin que le votant ne puisse changer d'avis après coup.