Développement d'une blockchain en C#

Travail présenté à M.Rabat et M.Delisle dans le cadre du cours *PPRO 605*

Travail réalisé par REICHART Thibaut

3 Juin 2022

Sommaire :

Introduction :

Le projet sur lequel j'ai travaillé lors de ces deux derniers mois pour ce module de PPRO0605 est le développement d'une blockchain en C#. Tout d'abord, j'ai essayé d'y définir les limites et ce qu'on pouvait attendre d'un tel projet. Cependant, le terme de blockchain est largement équivoque il a donc fallu procéder de manière méthodique. Ce projet s'inscrit dans la suite des cours d'info 0603. C'est pourquoi, pour m'imprégner du sujet je me suis alors replongé dans ces cours afin d'avoir une base solide sur les principaux algorithmes de hachage. Une fois les outils en tête pour réaliser ce projet il me fallait un objectif précis et non l'idée abstraite d'une blockchain. Ainsi, la réunion avec M.Rabat au retour du présentiel m'a permis d'avoir un objectif final à vous présenter : le développement d'une blockchain en C# exportable comme API, pouvant être utilisé à des fins d'élections.

La blockchain se doit d'être « prête à l'emploi » pour différentes applications. De plus, elle doit avoir les qualités qu'on attend d'une telle structure. Des transactions qui sont sécurisés et de pair à pair, la possibilité de créer un environnement économique avec une crypto-monnaie, le minage de blocks pour implémenter le système de proof-of-staking ou working. Les travaux de M.Kremer(1) mettent en lumière les enjeux principaux d'une élection par blockchain. En effet, le vote doit être unique, immuable, anonyme et l'individu ne doit pas être influencé par autrui lors de son vote. La dernière de ces conditions est pour le moment invérifiable, pour un vote en ligne il est impossible d'usurper le rôle de l'isoloir. Néanmoins, cette blockchain remplit les 3 premières règles énoncés, on admettra alors que la mise en place de la 4ème règle revient à la charge des utilisateurs.

Le projet comporte deux types de blockchains : Une plus classique dite « crypto-blockchain » qui permet la mise en place d'une économie autour d'une cryptomonnaie. L'autre finalisant l'objectif fixé à savoir la blockchain pour un système d'élections.

Fonctionnement de l'application :

La crypto blockchain :

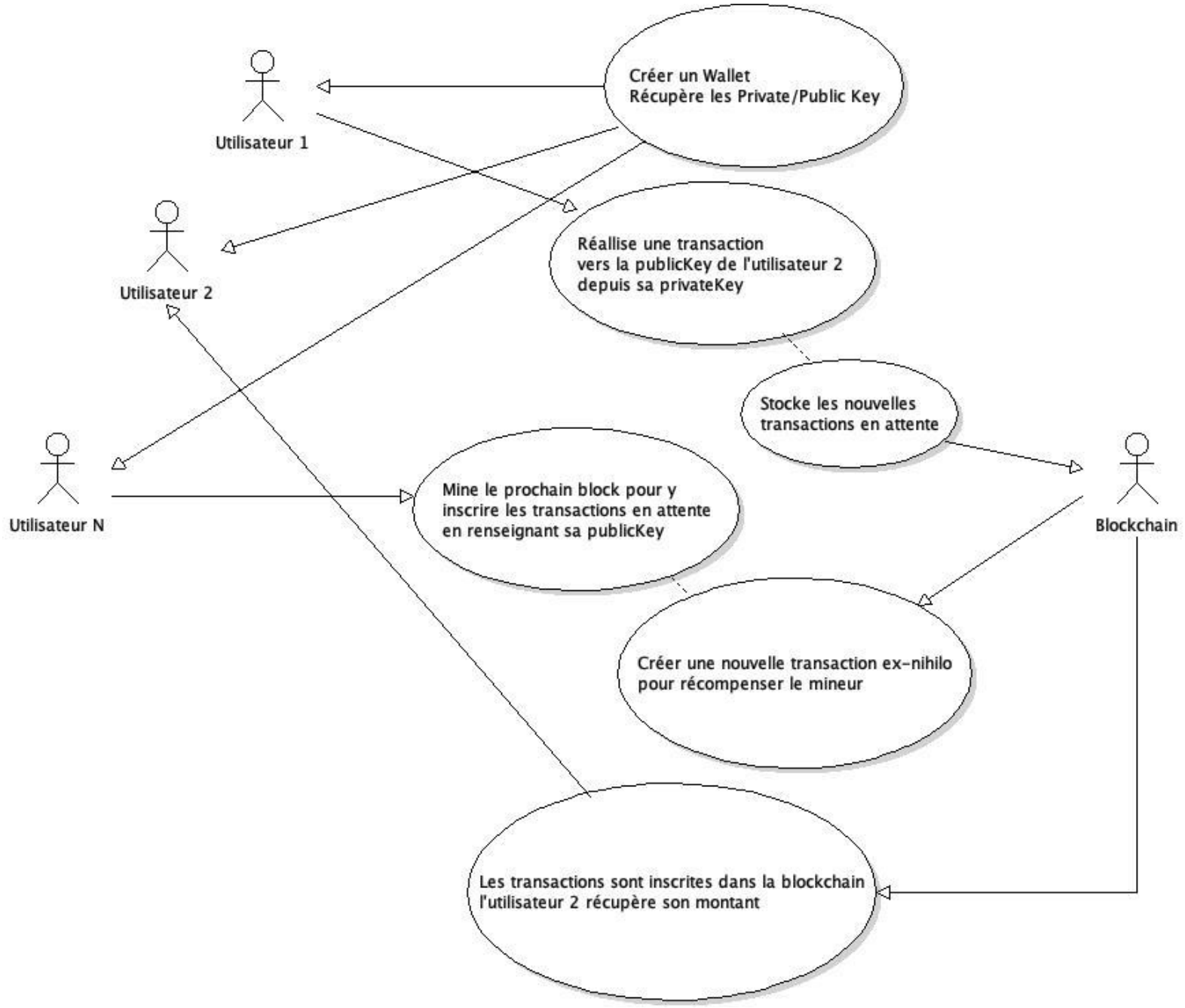

Figure 1 : Diagramme d'utilisation de la cryptoblockchain

Ce diagramme schématise l'utilisation de la cryptoblockchain par un utilisateur tiers. Comme on peut le voir, chaque utilisateur doit créer un Wallet afin d'utiliser la blockchain. C'est via le certificat contenu dans ce Wallet que les transactions vont pouvoir être signées et donc sécurisées. Le Wallet fonctionne comme un compte en ligne « classique » la privateKey faisant office de mot de passe et la publicKey de nom de compte. C'est ce pour quoi il est demandé à l'utilisateur de renseigner la privateKey de son Wallet pour pouvoir dépenser la cryptomonnaie qu'il a accumulé.

À l'inverse pour recevoir il n'a besoin que de sa publicKey. De plus, n'importe quel utilisateur peut consulter la blockchain, il verra alors les transactions effectuées avec les adresses(publicKey) de chaque Wallet. Ainsi, seul l'utilisateur ayant créée le Wallet peut donc (s'il ne l'a pas communiqué) dépenser sa cryptomonnaie.

Enfin, pour compléter une transaction il est nécessaire qu'un block la contenant soit ajouté à la blockchain. En effet, une fois la transaction créée elle est stockée dans une liste d'attente pour transaction au sein de la blockchain mais c'est seulement une fois que le prochain block sera miné que la transaction deviendra valide et que l'utilisateur bénéficiaire de la transaction pourra récupérer le montant. De ce fait un utilisateur peut en renseignant sa clé publique (donc son adresse) miner le prochain block c'est-à-dire l'ajouter à la chaine déjà en place via le proof-of-working en échange d'une récompense. Chaque utilisateur peut également consulter la balance de son compte, en renseignant son adresse à la blockchain. Cette dernière consultera alors toutes les transactions dont l'adresse a fait part pour afficher le montant. Il est donc important de noter qu'il peut y avoir un décalage si une transaction concernant l'adresse en question n'a pas encore été ajouté à la blockchain par le minage.

Blockchain pour un système d’élections :

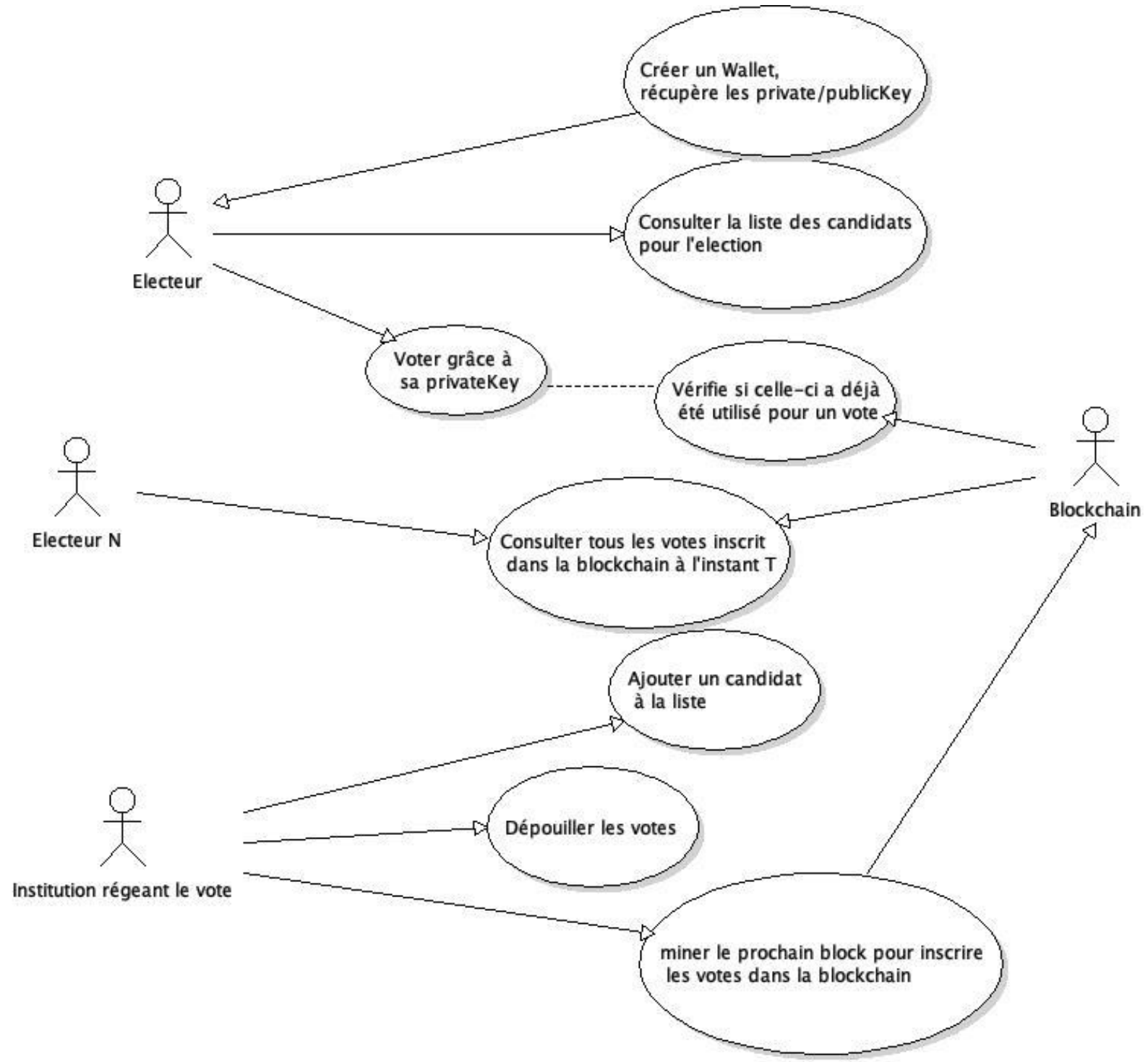

Figure 2 : Diagramme d'utilisation de la blockchain pour système d'élections

Pour cette partie il était important de respecter les 3 principes qui ont été énumérés plus tôt à savoir : l'unicité, l'anonymat et l'immuabilité du vote. Pour pouvoir voter un électeur doit créer un Wallet, on peut imaginer que ce soit l'institution organisant l'élection qui s'en charge afin de bien créer un seul Wallet par électeur. Une fois ces deux clés (privées et publiques) récupérées l'électeur peut consulter la liste des candidats. Cette liste est modifiable, l'institution peut y ajouter des candidats.

Pour garantir l'unicité du vote il est demandé à l'électeur de saisir sa clé privée lors de la création du bulletin de vote. La blockchain se charge alors de vérifier s'il n'y a aucune clé publique correspondant à celle associée à la clé privée saisie dans la blockchain et dans les transactions en attente. Ainsi, chaque Wallet et par extension chaque électeur ne peut saisir qu'un seul bulletin de vote.

Afin de préserver l'anonymat de chaque vote, sur le bulletin il est seulement mentionné la clé publique de l'électeur et le nom du candidat (ou autre). On mentionne la clé publique et non privée pour ne pas laisser des individus malveillants usurper l'identité de l'électeur lors d'une prochaine élection. Chaque électeur peut alors uniquement connaître son vote.

Cette blockchain fonctionne aussi par proof-of-working qui permet de garantir l'immuabilité des votes. Le minage s'effectue alors sans adresse pour être récompensé, on suppose que l'institution qui régit le vote n'a pas d'intention lucrative. De plus, il est possible de vérifier à tout instant l'intégrité de la blockchain si jamais l'une des transactions devait être modifiées, sa signature et même le calcul du hashage seraient alors faux. Enfin chaque utilisateur peut la consulter et donc vérifier son propre vote grâce à sa clé publique.

Analyse technique du projet :

Pour implémenter ce projet, il a été nécessaire de le découper en différentes classes se complétant les unes les autres. Comme le montre le diagramme UML (figure 3). Le squelette du projet est composé d'une chaine et des blocks la composant, on retrouve également les transactions qui créent à elles 3 le principe de blockchain. La classe Wallet permet notamment la signature via des algorithmes de chiffrement par courbes elliptiques et la classe Data de mettre en place le côté API du projet. En effet, on retrouve dans les classes Transaction, Chain et Data 2 constructeurs à paramètres différents selon la blockchain que l'on souhaite utilisé même principe pour la fonction ajoutBlockAttente() dans Chain, qui est surchargée pour s'adapter aux besoins de chaque type de blockchain.

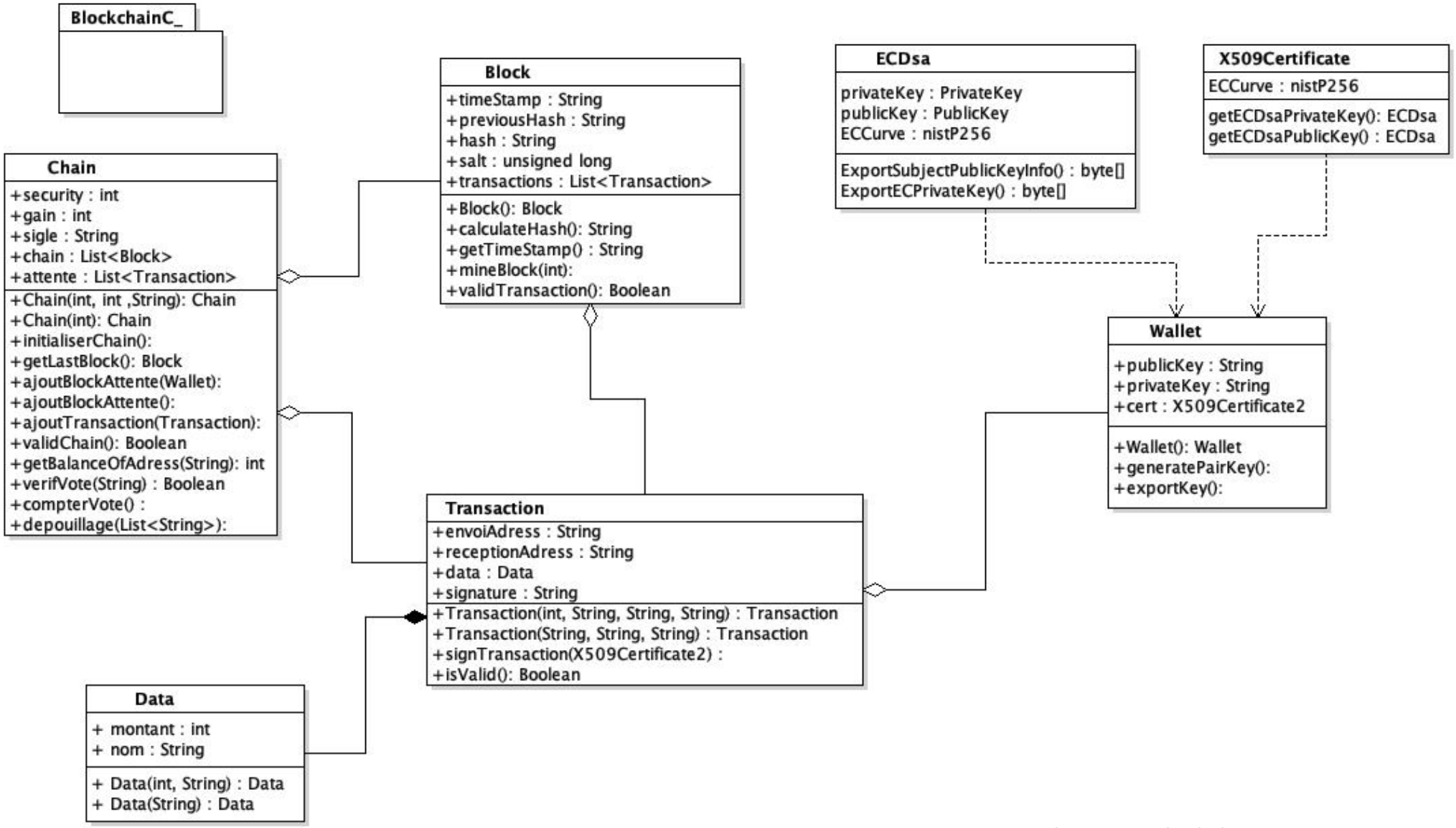

Figure 3 : UML du projet Blockchain

Fonction de hachage des Blocks et vérification :

Chaque Block qui compose la Blockchain doit calculer sa valeur de hash. Comme on peut le voir sur le diagramme UML (fig.3), la classe Block comporte deux String à savoir previousHash et hash. Ces valeurs sont les liens qui lient les blocks entre eux afin de créer la Blockchain. En effet, à l'exception du block dit genèse, chaque Block comporte en son sein la valeur de hashage du block qui le précèdent ce qui permet une authentification rapide et efficace de chaque block.

Pour calculer cet hash on récupère tous les attributs stockés dans le Block dont la valeur du hash du Block précédent qu'on transforme alors en String (via un Json pour les listes d'objets tel que la liste de Transaction. C'est ce String qu'on va alors hacher via la fonction SHA256().

En plus de lier les blocks entre eux, cet hash permet surtout d'assurer l'intégrité de chaque block et transaction. Sa nature de calcul rend chaque hash unique, imaginons qu'un utilisateur malveillant veuille modifier l'un des contenus d'une de ces transactions. Lors de la vérification de l'intégrité de la blockchain, on regarde dans un premier temps si le previousHash correspond bien au hash du Block précédent mais surtout la blockchain recalcule alors chaque hash propre à chaque Block et le compare avec celui qui y est inscrit, en modifiant une partie de la transaction utilisée pour le calcul de hachage la blockchain identifierait alors une falsification des données. Même si l'utilisateur malveillant venait à lui-même recalculer le hash et le modifier en conséquence il faudrait le faire pour chaque block qui suit la dans la blockchain. Ce qui semble en théorie possible d'où l'implémentation du proof-of-working.

Sécurisation de la blockchain par le Proof-of-Working et les transactions multiples :

S'il on s'en tient à la première partie falsifier la blockchain semble en théorie possible, avec une transaction par block il suffirait alors de modifier le block que l'on veuille, recalculer le hash pour chaque block qui suit et le tour est joué. Afin de palier à ces problèmes de sécurité on complexifie le calcul avec le proof-of-working et on implémente le fait qu'un seul block puisse accepter plusieurs transactions pour l'ergonomie et des raisons de réalisme (un block pour une transaction rend la blockchain inutilisable et trop coûteuse).

Le proof-of-working (POW) est un concept qui complexifie et donc rallonge les calculs même pour les ordinateurs les plus puissants. Pour le comprendre on va se pencher sur les attributs security de Chain et salt de Block. Le int security est défini lors de la construction, il correspond au nombre de « 0 » attendu en début de chaîne pour l'attribut hash. Cela semble contradictoire car comme énoncé précédemment chaque hash est unique selon ces propres composants. C'est là que l'attribut salt devient important. Lors du minage du Block, le Block va calculer son hash tant qu'il ne correspond pas aux attentes définis par la variable security. Et afin de ne pas calculer tout le temps la même valeur on incrémente la valeur salt à chaque calcul. À titre d'exemple, pour une security de 5 il faut en moyenne 16^5 soit 1 millions de calcul pour obtenir les 5 « 0 » en début de chaîne. Ce qui rend la falsification d'une blockchain dès lors quasi impossible pour un très grand nombre de « 0 ».

Plusieurs transactions en un seul Block est très pratique cela permet une économie d'énergie et rend la blockchain réaliste. Il suffit d'ajouter les Transactions dans une liste que l'on utilisera pour le calcul du hash. Mais la particularité des transactions c'est de permettre de renforcer encore la sécurité de la blockchain via les signatures par courbes elliptiques.

Signature des transactions par courbes elliptiques :

Le dernier niveau de sécurité est la signature de chaque transaction par une courbe elliptiques (EC). La courbe choisie est une EC nistP256. La signature se fait grâce à un certificat de type X509Certificate2 dans la classe Wallet. Ce certificat se génère par la fonction CertificateRequest() qui prend en paramètres une ECDsa c'est-à-dire un algorithme de signature digital par courbe elliptique. Via le certificat on peut alors récupérer les private/public Key de notre ECDsa qui vont nous servir à signer et authentifier les transactions.

Pour mettre en place ces signatures on récupère via le certificat la clé privée de l'émetteur de la transaction c'est via sa clé privé qu'on va signer la transaction grâce à la fonction signData() de la classe ECDsa, le résultat est stocké dans la variable prévue à cet effet. Encore une fois le résultat d'une même signature sera toujours le même et c'est cette propriété qui nous permet la sécurisation de chaque transaction.

Pour la vérification on ne peut pas utiliser la clé privée qui a permis de signer la transaction puisque seule la clé publique apparaît dans la transaction (on rappelle que lors de l'envoi on affiche la clé publique et non la clé privée de l'utilisateur). Cependant, en créant un objet ECDsa() et en y important la clé publique on peut alors vérifier si une signature a bien été signé par la clé privée associé à la clé publique. C'est le point clé de ce chiffrement, on peut utiliser une clé privée que l'on garde secrète mais la clé publique que l'on peut partager et qui ne dévoile pas la clé privée permet de confirmer si c'est bien cette clé privée qui a signé pour ces données précises. Comme dans le cas du calcul du hash du Block on reprends les données contenues dans la transaction et on les compare avec les signatures et la clé publique, ce qui nous indique si une falsification a été réalisé ou non. On comprend alors que sans une puissance de calcul phénoménale il est impensable de pouvoir falsifier une seule transaction, et encore moins un Block.

Gestion du projet :

Deux mois pour réaliser ce projet me semblait de premier abord plutôt large, à la suite des deux semaines de recherches j'ai plutôt bien enchaînés toutes les étapes de mon code. Cependant, la signature par courbes elliptiques a été non seulement le plus long mais aussi le plus complexe à mettre en place.

On peut le voir sur le journal que j'ai rempli tout le long du projet ou même sur le diagramme de Gantt (fig.4). Notamment à cause du manque de connaissances personnelles, j'avais jusqu'alors seulement effleuré le sujet. Mais aussi dû au fait du manque de ressources pratiques que j'ai pu trouver dans mes recherches. En effet, j'avais jusque-là toujours réussi à voir des exemples pratiques des idées que je voulais implémenter, pas forcément en C# mais cela donnait tout de même un bon aperçu de la structure du code. Enfin, les seuls objets API existants sur le sujet n'étaient pas compatibles avec le système d'exploitation de ma machine.

Par ailleurs, les Wallets qui n'étaient pas prévu dans la roadmap de base sont un ajout de taille. Ils permettent une véritable personnification de l'application et la rende beaucoup plus réaliste.
Tout n'est pas parfait, il manque notamment la possibilité de créer cette blockchain via un proof-of-stake que j'aurais aimé développer afin de proposer des blockchains différentes tant sur le contenu que sur la forme. De plus, un système de base de données peut être souhaitable à l'avenir pour les Wallets, cela éviterait alors les actions fastidieuses de copier-coller les clés privés et publiques pour simuler les transactions.

Figure 4 : Diagramme de Gantt du projet

Conclusion :

C'est avec fierté que je présente ce projet que je pense avoir réussi à mener à bien au bout de ces 8 semaines. Je ne pensais pas réussir à aller aussi loin surtout en tant que néophyte en C# dans l'implémentation des fonctionnalités notamment les courbes elliptiques et dans leur explication dans ce rapport.

Je garde une petite déception pour le proof-of-staking ou encore la base de données qui auraient pu peaufiner ce projet. Ce qui me fait réaliser qu'un projet ne se finit jamais et ne se déroule jamais comme on le souhaite même lorsqu'on est seul.

Il s'agit de mon premier projet informatique seul et je pense avoir beaucoup appris sur la rigueur de travail et la gestion d'un emploi du temps. Je ressors à titre personnel grandi de cette expérience, j'ai su ne pas me décourager lorsque je rencontrais une difficulté notamment les EC qui me semblaient impossible surtout après les erreurs pour plateformes incompatibles. Et j'ai pu apprendre un tout nouveau langage que je ne connaissais pas et approfondir mes connaissances sur un sujet qui me passionne.

Référence :

(1) V*erifying privacy-type properties of electronic voting protocols*, rédigé par Delaune Stephanie, Kremer Steve et Ryan Mark en 2007. (En ressource sur le git).