PSC ECO006
SYSTÈME DE VOTE PAR BLOCKCHAIN

Rapport final

Auteurs

Solène Gomez, Mohamed Malhou,
Anass Ben Bouazza, David Admète

Promotion X2018

Tuteur
Julien Prat

Coordinateur
Guillaume Hollard

Remerciements

Nous souhaiterions tout d'abord remercier certaines personnes pour l'aide précieuse qu'elles nous ont apportée tout au long de la réalisation de notre projet, que ce soit une aide matérielle ou technique, ou même un soutien lors des phases délicates. Nous adressons donc nos sincères remerciements à :

— M. Guillaume HOLLARD, notre coordinateur au sein de département d'économie de l'Ecole Polytechnique, pour sa grande motivation à propos de ce sujet, qu'il nous a très vite transmise, et pour ses encouragements au cours de la conduite du projet ;

— M. Julien PRAT, notre tuteur, pour avoir contribué à fixer nos objectifs intermédiaires, pour nous avoir mis en contact avec les personnes qualifiées pour répondre à nos questions, et pour sa réactivité et son implication tout au long du projet ;

— Mme Myrto ARAPINIS, chercheuse à l'université d'Édimbourg, pour nous avoir partagé son protocole de vote sécurisé ainsi que sa preuve, pour avoir répondu à toutes nos questions techniques concernant sa compréhension, et pour sa grande disponibilité ;

— M. Ivaylo GENEV, chercheur de l'équipe d'Édimbourg, pour nous avoir transmis son travail sur l'implémentation du protocole E-cclesia, et pour avoir répondu à toutes nos questions sur les difficultés que nous rencontrions au fur et à mesure du projet ;

— M. Gilles MENTRE, fondateur d'Electis et ancien polytechnicien, pour nous avoir montré à quel point la fin de notre projet était au cœur de l'actualité démocratique, pour son temps et sa réactivité, ainsi que pour nous avoir permis de participer au premier vote test organisé dans une vingtaine d'universités dans le monde.

Résumé et mots clés

Dans le cadre de notre projet scientifique collectif, nous avons choisi de nous pencher sur un sujet complexe, éthiquement très débattu, et technologiquement novateur. Il s'agit de l'implémentation d'un système de vote par blockchain. Au cours de notre projet, nous nous sommes plongés au cœur de l'architecture d'une blockchain, et plus particulièrement sur le fonctionnement d'Ethereum, blockchain créée en 2015. Nous avons également analysé le protocole de vote E-cclesia, fiable, sécurisé, et répondant aux traditionnels critères des votes démocratiques à grande échelle. Notre projet a alors consisté à comprendre le protocole, puis à l'adapter pour l'implémenter sur la blockchain Ethereum, dans le but de développer une application permettant de tels votes sécurisés, décentralisés et à distance. [12]

Mots clés : Vote électronique sécurisé, Blockchain, Ethereum, Zero-knowledge, Protocole, Implémentation, Application, Time Lock Encryption

Abstract and keywords

For our Scientific Group Project, we decided to study a blockchain voting system. It is a complex subject with innovative technology, yet ethically discussed. During our project, we analysed the architecture and the functioning of Ethereum, a blockchain created in 2015. We also examined the voting protocol of E-cclesia, designed to be a secure, reliable voting system, compatible with the criteria of a large-scale democratic voting system. Our project consisted in implementing this voting protocol in the Ethereum blockchain, to develop an application allowing remote, decentralized and secure voting.

Keywords : Secure electroning voting, Blockchain, Ethereum, Zero-knowledge, Protocol, Implementation, Application, Time Lock Encryption

TABLE DES MATIÈRES

INTRODUCTION

Les systèmes de vote

Dans un monde de plus en plus démocratisé, et face à l'augmentation de la population, le vote est un outil crucial et compliqué à mettre en place à grande échelle. En effet, de nos jours, les grandes élections en Europe se font encore avec des bulletins papiers, ou plus rarement à l'aide des boitiers électroniques. Ceux-ci comptabilisent les votes, mais ne vérifient pas l'identité des votants, tâche toujours exclusivement manuelle (donc chronophage et soumise à de possibles erreurs). Les élections à plus petite échelle, au sein d'entreprises ou d'universités, peuvent déjà se faire électroniquement, mais ne sont pas sécurisées. Ces systèmes de vote centralisés (c'est-à-dire nécessitant une unité de centralisation qui récupère tous les votes, les comptabilise, puis annonce les résultats) posent plusieurs problèmes.

Tout d'abord, la mise en place de tels votes est extrêmement couteuse, en termes matériels, humains et financiers. Par ailleurs, on observe un manque de confiance croissant envers les unités de centralisation, dû à de nombreux abus possibles, à l'image des présomptions de trucage en Russie. A titre d'exemple, lors des dernières élections présidentielles en République Démocratique du Congo, le manque de confiance des citoyens devant les résultats qui tardaient à être publiés (à cause des délais dus au comptage manuel des voix) a provoqué des tensions et des émeutes qui se sont soldées par plusieurs décès. Ce manque de confiance remet en question les plus profonds fondements de la démocratie, et incite les citoyens à manquer à leur devoir de vote. L'abstention s'explique également par l'impossibilité pour certains citoyens de se rendre en personne à leur bureau de vote (éloignement, âge), et à la démarche assez chronophage qu'est celle de trouver une personne à qui faire une procuration, puis de se rendre en commissariat pour réaliser les démarches.

Face à ce manque de facilité pour voter, et à ce manque de confiance dans les institutions centralisant lesdits votes, les citoyens désertent les urnes depuis plusieurs années. Notre projet de système de vote par blockchain pourrait permettre de palier à ces inconvénients, et révolutionner le monde des scrutins.

La blockchain

La blockchain est une technologie relativement récente, au cœur des débats depuis 2008, année marquée par l'invention du Bitcoin. La technologie blockchain repose sur une succession de blocs, à l'image de Legos emboités. Chaque bloc contient un contrat, sur lequel est inscrit une clef finale. Celle-ci est témoin de chaque caractère du contrat courant et de la clef du contrat précédent le bloc dans la chaine. Ainsi, la caractéristique la plus importante de la blockchain, et qui en fait un sujet d'étude tout à fait prometteur pour implémenter un système de vote, est sa stricte immuabilité. En effet, pour modifier de façon indétectable un caractère sur un contrat, il faudrait mettre à jour sa propre clef, mais aussi l'ensemble des clefs des contrats suivants dans la chaine, ce qui est en pratique quasiment impossible. [7] La particularité d'Ethereum en comparaison avec la plupart des autres blockchains existantes est qu'elle dispose d'un langage Turing-complet, Solidity, permettant d'effectuer des algorithmes sur la blockchain en elle-même. Ces algorithmes sont ce que l'on appelle les contrats intelligents (smart contracts), et permettent de profiter du plein potentiel de l'architecture décentralisée de la blockchain.

D'autres problèmes peuvent également se trouver résolus avec cette technologie émergente. Elle pourrait permettre de vérifier et d'enregistrer chaque transaction (c'est-à-dire chaque vote lorsque on l'utilise comme un système démocratique), sans aucun risque qu'elle puisse ensuite être altérée par un possible fraudeur. De plus, grâce à un système d'authentification rigoureux, la technologie blockchain offrirait aux votants une solution pour soumettre leur choix sans crainte d'exposer leur identité ou leur vision politique. Une fois le vote soumis, le votant peut vérifier que son vote ait été enregistré, qu'il n'ait pas été modifié et qu'il ait été compté pour le résultat final. Aucune instance ne peut plus avoir accès au contenu du vote. Ainsi, il apparait que cette technologie transparente, décentralisée et résistante, offre une possible solution à un système de vote démocratique. Bien sûr, il faudrait prendre du temps pour éduquer les populations, leur expliquer le fonctionnement de cette technologie, afin qu'elles aient confiance en ce système de vote décentralisé, et donc sans tiers. Sans cette compréhension, un système de vote fondé sur cette technologie serait vu comme confus et peu sûr, incontrôlable s'il est piraté. Alors qu'en réalité, le seul point crucial pour rendre un tel système de vote sûr serait d'implémenter correctement un protocole de vote prouvé.

À Naples, en Italie, un vote fondé sur la technologie blockchain a été testé en 2017, et les résultats se sont avérés mitigés. En effet, les couts en hardware étaient importants, et le processus était plus long pour donner les résultats que les systèmes de vote traditionnels. Aux USA, une application de vote par blockchain, Voatz, a été utilisée pour 54 élections à moyenne échelle, mais elle s'est finalement révélée vulnérable, permettant à de potentiels hackeurs de voir et d'altérer les votes.

Nos motivations

C'est dans ce contexte où le besoin en transparence est clair, mais où la technologie blockchain est très ardemment discutée et controversée, que notre projet prend tout son sens. En développant un système de vote électronique, dont la sûreté est prouvée mathématiquement, et qui permet à l'utilisateur un usage à distance, rapide, simple et anonyme, notre équipe propose donc une avancée technologique considérable dans ce domaine. Notre objectif fut ainsi de créer une application inédite répondant à ce cahier des charges, pour ensuite la tester sur des universités, puis des entreprises, et pourquoi pas des pays entiers, résolvant les problèmes soulevés plus haut, tant sur le plan démocratique que sur le plan humain.

Défis à relever

Nous avons utilisé la technologie blockchain, et plus particulièrement la blockchain Ethereum pour implémenter un protocole de vote sécurisé et décentralisé, devant répondre à plusieurs critères usuels et déterminants en matière de vote. Ceux-ci sont :

- La correction : tous les votants, et uniquement eux, doivent voter au plus une fois ;
- La vérifiabilité : chacun doit pouvoir vérifier que les votes ne sont pas contrefaits ;
- L'anonymat : personne ne peut retrouver pour qui a voté un électeur ;
- L'impossibilité de preuve : un votant ne peut prouver le contenu de son vote
- La rapidité : la durée du vote doit pouvoir être déterminée, et le système doit fournir les résultats à la date de fin prévue, sans jamais être en avance

— La simplicité : le procédé de vote doit être simple d'utilisation
— L'usage à distance : les votants ne doivent pas avoir à se déplacer pour voter

Le projet était certes ambitieux et complexe, mais la portée que pouvait prendre nos travaux a probablement été la source de motivation la plus féconde tout au long de sa réalisation.

I
CADRE DU PROJET

I.1 Travail en collaboration

Notre projet, dans le cadre du PSC, s'inscrit dans un travail de bien plus grande ampleur. En effet, nous avons avancé en collaboration étroite avec d'autres acteurs du vote par blockchain. Tout d'abord, la chercheuse Myrto Arapinis ainsi que son équipe de l'université d'Édimbourg. Myrto nous a partagé son protocole de vote sécurisé, ainsi que sa preuve mathématique. Tout notre travail d'implémentation s'est fondé sur la compréhension de cette preuve, et l'adaptation du protocole à la blockchain Ethereum. Par ailleurs, deux membres de l'équipe de Myrto, Ivaylo Genev et Jonathan Levi, nous ont amplement aidé concernant toutes les questions de programmation et de compatibilité entre les différentes plateformes utilisées. Les deux chercheurs nous ont transmis le travail qu'ils avaient déjà entamé sur ce protocole, et nous ont permis de gagner un temps précieux puisqu'ils avaient déjà étudié bon nombre de questions relatives à la sécurité de ce système de vote.

Nous avons également travaillé en lien étroit avec l'équipe de Gilles Mentré, ancien polytechnicien et fondateur d'Electis. Notre tuteur, Julien Prat, également en collaboration avec Electis, nous a mis en contact. Le groupe code et expérimente un système de vote similaire au notre, sur une autre blockchain, Tezos, et est intéressé pour récupérer notre travail lorsqu'il sera terminé. Ces échanges réguliers nous donc ont montré les aboutissements concrets qui pouvaient être ceux de notre projet, et l'existence de forts liens avec la communauté scientifique actuelle nous a permis de toucher une dimension encore plus large du travail de groupe. Ainsi, ce PSC s'inscrit dans un cadre bien plus large, ou de nombreux acteurs sont en collaboration afin de faire avancer les possibilités de scrutin sur la blockchain, et de les rendre opérationnels au plus vite.

I.2 Définition des objectifs

Dans ce contexte, nous avons pu définir notre projet de manière claire, en nous fixant des objectifs progressifs :

- Réussir à interagir avec la blockchain Ethereum
- Comprendre le protocole de vote sécurisé E-cclesia
- Adapter le protocole et la partie de code déjà disponible à la blockchain Ethereum
- Implémenter le reste du protocole, et en particulier la Zero-knowledge proof (II.2)
- Développer une application de vote à distance opérationnelle

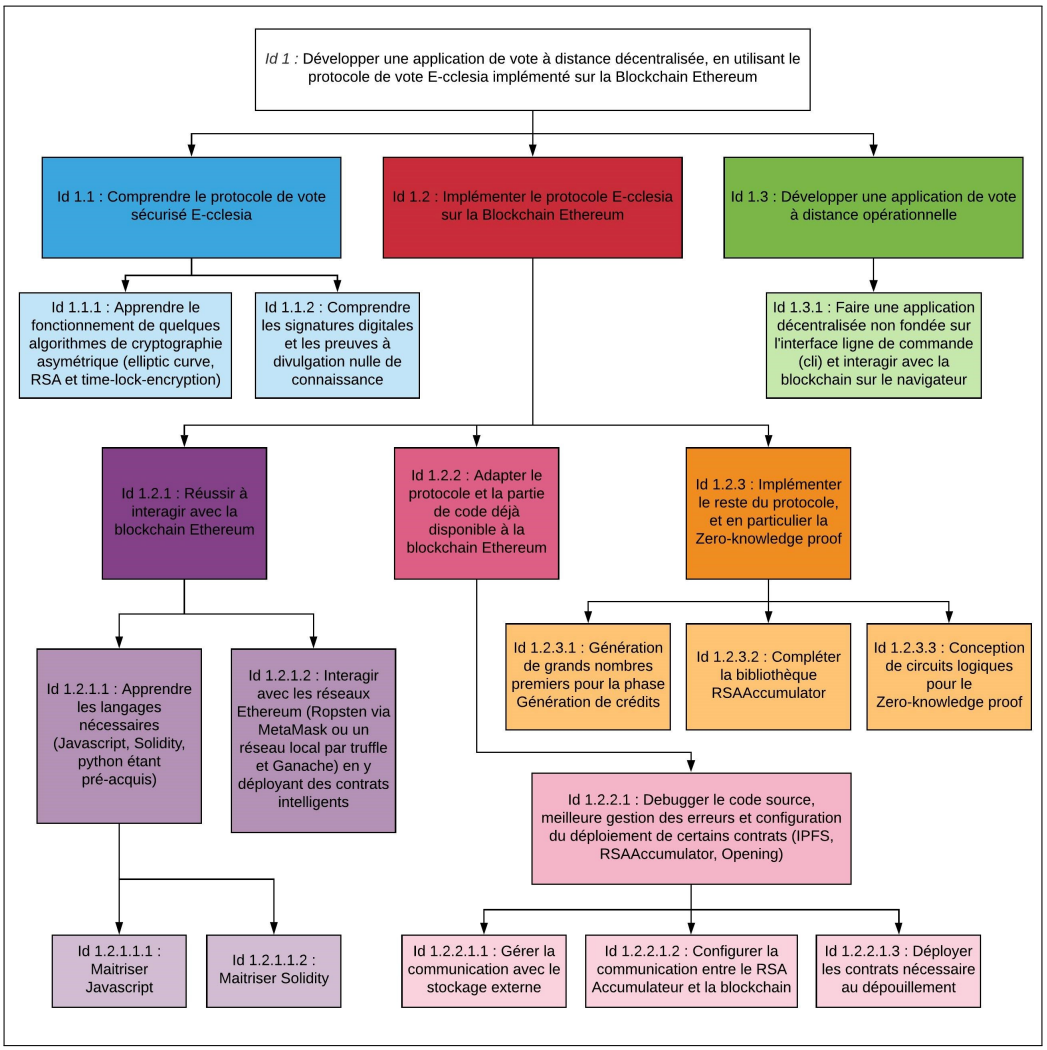


Figure 1 – Définition des objectifs de notre PSC

II
PRÉ-REQUIS TECHNIQUES

Avant de rentrer dans le détail du protocole de vote, il convient de préciser plusieurs composantes majeures du projet, fonctionnant parallèlement au protocole, et sans lesquelles l'application de vote ne pourrait fonctionner. C'est pourquoi, avant de rentrer dans le détail du protocole, nous présentons la mise en place du Time Lock Encryption (TLE), de la Zero-Knowledge proof, ainsi que l'initialisation de la communication avec la blockchain Ethereum.

II.1 Time Lock Encryption

Cette partie du protocole est née de la nécessité de garder les votes secrets jusqu'au dépouillement, afin de ne pas influencer la partie de la population n'ayant pas encore soumis son vote. L'application doit donc disposer d'un système permettant d'assurer que les votes ne puissent être ouverts avant la fermeture du dispositif.

La solution proposée par E-cclesia est le Time Lock Encryption. Ce dernier permet de chiffrer les votes lorsqu'ils sont soumis à l'application. Pour ouvrir le vote, il est alors nécessaire de déchiffrer le message associé, ce qui ne peut se faire que par l'intermédiaire de la résolution d'un problème par l'ordinateur. Ainsi, en déterminant un problème d'une complexité suffisante pour que celui-ci ne puisse être résolu avant la fin du vote, nous parvenons à sécuriser l'ouverture de celui-ci.

L'algorithme est similaire au célèbre algorithme de cryptage RSA, et fonctionne donc en deux parties : le chiffrement et le déchiffrement. On ne considérera pour l'instant que deux protagonistes : le votant, du côté de l'application de vote, qui envoie le vote et l'autorité de vote, du côté de la blockchain, qui est en charge de l'organisation et ouvre le vote au moment du dépouillement. Nous désignerons, par souci de simplicité, le votant lorsqu'il s'agit de la partie qui envoie le vote. De plus, nous noterons T le temps

(en secondes) pendant lequel le vote doit être tenu secret, et S le nombre de fois qu'un ordinateur peut mettre un nombre au carré modulo un entier en une seconde. S dépend évidemment de l'ordinateur en question, nous reviendrons sur ce point ultérieurement.

- Le protocole de chiffrement est le suivant :

i. Le votant choisit p et q deux entiers premiers, calcule $n = pq$ et $\phi(n) = (p-1)(q-1)$
ii. Le votant calcule également $t = TS$
iii. Le votant utilise un algorithme de cryptage simple quelconque, où K est la clé permettant de chiffrer et de déchiffrer le message. Il crypte le vote avec K, ce dernier devient donc C_v
iv. Le votant choisit un entier $a \in [\![1, n]\!]$ uniformément. Alors, on note $C_k = K + a^e \bmod n$ où $e = 2^t \bmod \phi(n)$
v. Le votant renvoie finalement les le n-uplet (n, a, t, C_k, C_v)

Le but du déchiffrement est de permettre à l'autorité de vote de retrouver le vote v à partir de (n, a, t, C_k, C_v).

- Le protocole de déchiffrement est le suivant :

i. L'autorité de vote calcule $b = a^{2^t} \bmod n$
ii. L'autorité de vote calcule $K = C_k - b \bmod n$
iii. L'autorité de vote calcule finalement v à partir de C_v et de K

Pour déterminer K à partir de (n, a, t, C_k, C_v), il convient de calculer $b = a2^t \bmod n$. En effet, n étant un entier de grande taille, il n'est pas possible de le factoriser pour obtenir p et q dans un temps raisonnable, ce qui rend impossible de simplement refaire le calcul utilisé pour chiffrer en sens inverse. C'est là qu'intervient la vitesse de calcul : le calcul de b ne peut pas être effectué par plusieurs machines en parallèle, et il prend donc T secondes à se terminer. [3]

Néanmoins, au vu des fortes différences de vitesse de calcul des ordinateurs, il est difficile de prévoir avec précision les problèmes permettant au vote de se dérouler convenablement. En effet, un problème trop simple

serait résolu avant le dépouillement, ce qui mettrait en péril le caractère secret des bulletins, alors qu'un problème trop complexe ne pourrait être résolu à l'heure du dépouillement, ce qui rendrait impossible l'accès aux résultats.

Expliquons donc notre façon de procéder. Supposons que l'autorité de vote dispose d'un ordinateur muni d'une bonne capacité de calcul, semblable à celle des meilleurs calculateurs en vente sur le marché. Nous implémentons un canal auxiliaire permettant aux votants de délivrer leur clé K au moment du dépouillement. Ainsi, l'autorité de vote n'aura finalement besoin de déchiffrer qu'un nombre limité de votes cryptés – ceux des votants ayant refusé de transmettre leur clé K – ce qui limitera la probabilité que le déchiffrement dépasse considérablement le temps T. En effet, les votes dont nous n'aurons pas récupéré la clé K pourront être déchiffrés parallèlement, sur plusieurs ordinateur, alors que deux ordinateurs différents ne peuvent pas être utilisés simultanément pour augmenter la vitesse de décryptage d'un unique vote. Ceci réduit donc les problèmes causés par la méconnaissance des capacités de calculs des ordinateurs mis en jeu.

Du point de vue de l'implémentation, contrairement à la plupart des autres parties du protocole, celle-ci est codée en Python, dont l'efficacité des bibliothèques de cryptographie (e.g. cryptography.hazmat.primitives.asymmetric) et de résolution de problèmes (timelockpuzzle.algorithms) a pu être vérifiée.

```python
def decrypt(n: int, a: int, t: int, enc_key: int, enc_message: int) -> bytes:
    # Successive squaring to find b
    # We assume this cannot be parallelized
    b = a % n
    for i in range(t):
        b = b**2 % n
    dec_key = (enc_key - b) % n

    # Retrieve key, decrypt message
    key_bytes = int.to_bytes(dec_key, length=64, byteorder=sys.byteorder)
    cipher_suite = Fernet(key_bytes)
    return cipher_suite.decrypt(enc_message)
```

FIGURE 2 – Algorithme de décryptage du TLE, implémenté en PYTHON

II.2 Zero-knowledge

Comme nous l'avons détaillé au début du rapport, le système de vote doit vérifier plusieurs propriétés. En particulier, nous nous intéressons ici aux conditions suivantes :

- Chaque vote doit effectivement provenir d'un des votants déclarés officiellement
- Le vote ne peut être modifié entre son déploiement et le dépouillement
- L'autorité de vote ne doit pas pouvoir identifier le vote d'un votant donné

Dans un premier temps, notons que ces conditions semblent presque contradictoires. En effet, il paraît difficile de ne pas connaître le vote d'un participant tout en pouvant vérifier que le vote de ce dernier n'a pas été corrompu entre le vote et le dépouillement. C'est en fait le principe de la Zero-knowledge proof. Elle désigne les protocoles permettant de s'assurer de l'authenticité d'une assertion (ici : "le vote n'a pas été modifié") sans divulguer aucune information sur le contenu du vote en question.

Dans le cadre de notre système de vote, plusieurs Zero-knowledge proofs sont utilisées, notamment pour vérifier que le vote n'est pas corrompu, et sont également mises en parallèles avec d'autres algorithmes, tels que les accumulateurs, pour vérifier que le votant associé à un vote appartient bien à l'ensemble des votants officiels, sans donner d'information sur son vote ni sur le reste des votants.

En particulier, on choisit d'utiliser des zk-SNARKs (Zero-knowledge Succinct Non-interactive Arguments of Knowledge) dans notre implémentation : là où une Zero-knowledge proof rudimentaire peut faire appel à des interactions entre le fournisseur de preuve (qui veut prouver qu'il connaît une information sans la révéler pour autant) et le vérificateur (qui veut savoir si le fournisseur de preuve connaît cette information), une zk-SNARK ne fait pas rentrer en contact ces deux agents.

Expliquons brièvement le principe de la zk-SNARK, sans toutefois rentrer dans le détail de la théorie sur les fonctions holomorphes, qui permet de prouver le bon fonctionnement de l'algorithme. Dans un premier temps, l'assertion à prouver est transformée en un problème d'égalité entre deux produits de polynômes : $t(x)h(x) = w(x)v(x)$. Ainsi, le fournisseur de

preuve veut montrer au vérificateur que cette égalité est valide. Le vérificateur choisit ensuite aléatoirement un point s, afin de ramener le problème à la vérification de $t(s)h(s) = w(s)v(s)$. Ensuite, une fonction de chiffrement E est utilisée. Celle-ci doit vérifier plusieurs propriétés, habituellement associées aux fonctions holomorphes, c'est à dire dérivables en tout point de $\mathbb{C}$, sans toutefois faire partie de cet ensemble. Les propriétés de E permettent au fournisseur de preuve de calculer $E(t(s))$, $E(h(s))$, $E(w(s))$, $E(v(s))$ en connaissant uniquement $E(s)$, et non s. Enfin, le fournisseur de preuve multiplie les valeurs de $E(t(s))$, $E(h(s))$, $E(w(s))$, $E(v(s))$ par une constante k choisie aléatoirement, et les fournit au vérificateur. Celui-ci peut effectivement vérifier que l'égalité de départ est correcte, sans toutefois avoir possibilité de calculer $t(s)$, $h(s)$, $w(s)$ ou $v(s)$. Remarquons que les deux agents ne sont à aucun moment rentrés directement en contact.

L'implémentation d'une Zero-knowledge proof sur Javascript utilise les paquets circom et snarkjs de Node.js, et se fait comme suit :

— On conçoit un circuit traduisant les informations et les contraintes de la preuve. Par exemple, si l'on souhaite prouver que l'on connaît deux nombres a, b et c tels que $ab - c = d$, le circuit traduit que l'on doit fournir en entrée a, b et c, qu'il existe une sortie d, et qu'il faut que $ab - c = d$. Circom permet d'implémenter ces contraintes facilement : on trouvera ci-dessous l'implémentation d'un tel circuit.
— On génère avec snarkjs un contrat SOLIDITY traduisant le circuit précédent ainsi que le protocole de la Zero-knowledge proof.
— On deploie ce contrat sur la blockchain.

Nous choisissons d'utiliser ces paquets car, contrairement aux autres implémentations disponibles, snarkjs permet d'obtenir directement un contrat déployable sur la blockchain. Alors en faisant appel à ce contrat, le fournisseur de preuve peut prouver qu'il connaît a, b et c vérifiant les contraintes, et le vérificateur peut vérifier que le fournisseur a fourni cette preuve.

Nous ne sommes malheureusement pas encore parvenus à intégrer la Zero-knowledge proof à l'implémentation du protocole au jour de la rédaction de ce rapport, mais nous continuerons à travailler dessus jusqu'à la soutenance. En effet, le concept de zk-SNARK reste aujourd'hui encore à la pointe de la recherche cryptographique et son implémentation pratique pose encore des problèmes : le paquet snarkjs que nous utilisons n'a été mis à disposition du public qu'en août 2018 et présente encore des bugs, notamment au niveau de son compilateur. En particulier, les collectifs et start-ups à l'origine de telles implémentations eux-mêmes préviennent que

```
template Exemple() {
    signal private input a;
    signal private input b;
    signal private input c;
    signal output d;

    d <== a * b - c;
}

component main = Exemple();
```

FIGURE 3 – Exemple de circuit

leurs implémentations sont encore expérimentales.

Une possibilité à explorer pour la suite serait de se détourner de snarkjs et d'utiliser la librairie C++ libsnark, quitte à devoir par nous-mêmes implémenter les contrats traduisant la Zero-knowledge proof.

II.3 INITIALISATION

Un des enjeux majeurs du projet est de parvenir à interagir avec la blockchain ETHEREUM. Ainsi, nos différents programmes doivent pouvoir être interprétés par la blockchain, être capables de recueillir des informations auprès d'elle, et de la modifier.

Avant de présenter le processus d'initialisation, présentons une des particularités de la blockchain ETHEREUM, le gas, qui la distingue de sa célèbre homologue Bitcoin. Rappelons que chaque usager d'ETHEREUM possède un compte contenant une certaine somme en Ether (noté ETH), la monnaie courante de cette blockchain. Sur ETHEREUM, chaque opération effectuée est « payante » au sens où, à la suite de chaque transfert de données, le compte qui a lancé l'opération est débité d'une certaine somme, que l'on appelle le gas. L'intérêt principal du gas est de s'assurer que la blockchain

ne soit pas cible d'une surcharge – volontaire ou non – de calculs : le gas rend par exemple impossible l'exécution d'une boucle infinie. La somme débitée dépend du temps que l'usager souhaite laisser à la blockchain pour qu'elle réalise l'opération, et de la quantité de données transférées. Il se mesure généralement en wei, ce qui représente 10^{-18} Ether. A titre d'exemple, en mars 2020, un Ether correspond à environ une centaine d'euros, et le gas déployé pour le transfert d'un ether représente environ un dix-millième d'ether, soit environ un centime d'euro. Le coût des opérations effectuées sur Ethereum est donc à prendre en compte lorsque l'on organise un vote. A l'échelle du Royaume Uni, les élections générales de 2017 auraient coûté de l'ordre de 50 millions de livres si elles avaient été organisées avec Ecclésia, une somme considérable donc, qui reste néanmoins inférieure aux 140 millions de livres qui ont été effectivement dépensées [6].

Afin de ne pas induire de coût additionnel, nous testons donc nos programmes sur des réseaux de test, tels que le réseau Ropsten Testnet, ou encore sur des blockchains virtuelles. Celles-ci permettent de simuler la blockchain Ethereum avec un haut niveau de réalisme, incluant le calcul du gas nécessaire à chaque transaction, la seule différence étant qu'il est possible de demander à ce qu'un compte soit réapprovisionné, par l'intermédiaire d'une fonction appelée « faucet ».Notre choix s'est porté sur Ganache, une plateforme permettant de simuler des blockchains, dont le système de gestion des erreurs est plus efficace que celui de ses concurrents.

L'initialisation consiste à mettre en place les canaux d'échange entre l'application et la blockchain. Nous communiquons avec celle-ci par l'intermédiaire de « contrats » que nous envoyons à la blockchain. Ceux-ci permettent par exemple d'ajouter des blocs dans la blockchain. Nous avons utilisé la plateforme Truffle pour compiler l'ensemble des fichiers, afin qu'ils soient lisibles par Ethereum. Ainsi, nous sommes capables d'échanger des informations avec Ethereum, et de visualiser les transferts effectués grâce à l'interface de Ganache.

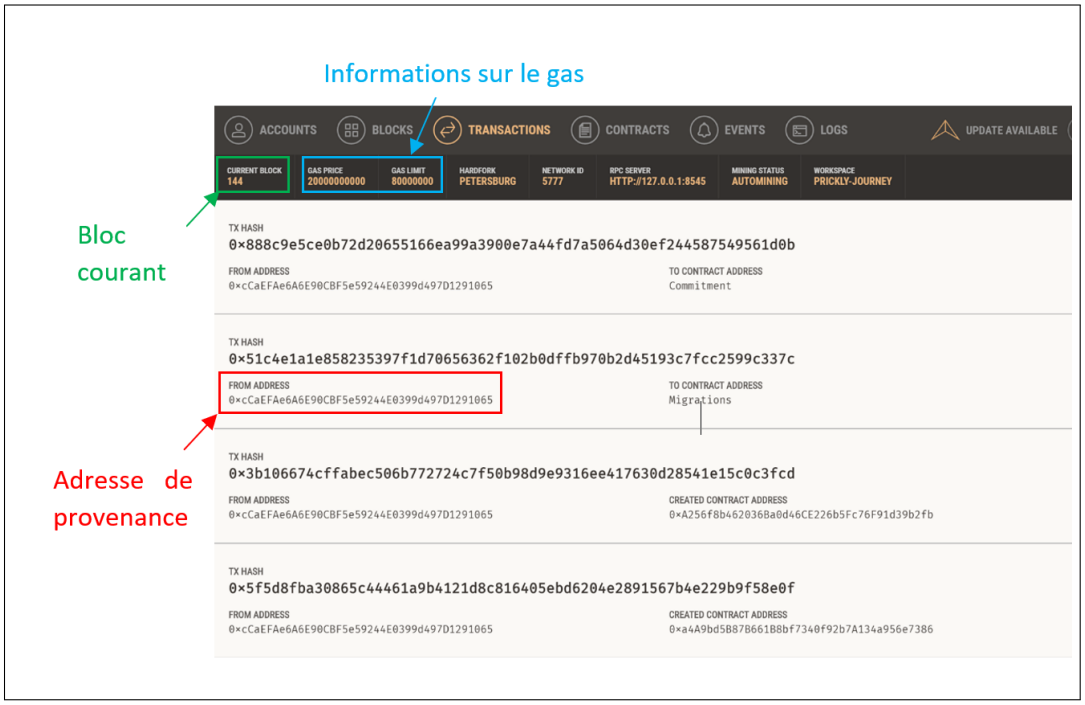


FIGURE 4 – Interface de communication de GANACHE – Transactions

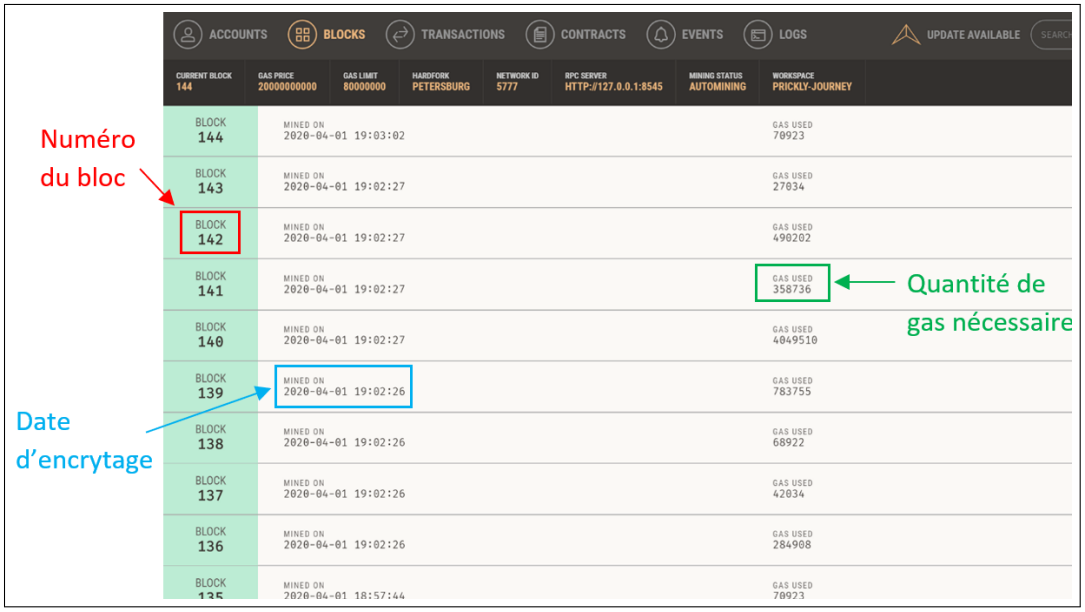


FIGURE 5 – Interface de communication de GANACHE – Blocs

III
LE PROTOCOLE

Le vote tel que le prévoit le protocole s'articule en quatre parties :
— L'enregistrement
— L'accréditation
— Le vote
— Le dépouillement

Chacune de ces étapes représente une phase différente du protocole. Nous avons ainsi décidé de travailler selon ces axes de manière à obtenir un code modulaire afin de faciliter la correction d'erreurs, tout en vérifiant l'adaptabilité des différents modules. Il est alors naturel de présenter notre travail selon les différents angles que nous avons pu adopter tout au long du projet, en exposant pour chaque partie nos avancées par rapport au projet initial, ainsi que l'organisation que nous avons choisie pour les aborder. L'annexe 1 résume sous la forme d'un schéma le fonctionnement de notre système de vote, en reprenant les phases principales du protocole, tout en le mettant en lien avec les différents systèmes sollicités.

III.1 Enregistrement

La première opération que l'autorité de vote doit effectuer n'est autre que l'enregistrement des votants. Chaque individu ayant le droit de vote se voit attribuer un identifiant, une clé publique, et une clé privée, qui lui permettront de voter anonymement. Détaillons donc le procédé correspondant. A ce stade, il s'agit d'initier la communication avec la blockchain. Pour ce faire, l'autorité de vote n'a qu'à initier le protocole de vote en publiant les contrats initiaux sur la blockchain Ethereum. Dans notre cas, c'est l'application qui s'en charge directement, en ordonnant l'envoi des contrats par le mécanisme décrit dans la phase d'initialisation. A l'issue de cet envoi, l'application reçoit de la part d'Ethereum l'adresse à laquelle se situent les contrats déployés. Celle-ci se présente sous la forme des adresses des différents blocs contenant les contrats, qui ne sont pas nécessairement des blocs consécutifs.

L'application rentre ensuite dans une phase d'échange, durant laquelle les votants peuvent interagir avec les contrats. Les canaux d'échanges sont alors provisoirement ouverts, afin de permettre aux votants d'accéder à leurs paramètres.

L'application déploie alors un contrat permettant de mettre en place le système de signature électronique sécurisée. Parallèlement, chaque votant se voit assigner une clé publique en fournissant son identité, indépendamment de la blockchain. La transmission de cette clé n'est pas étudiée dans le projet E-CCLESIA, puisque le choix de transmission revient à l'autorité de vote. La transmission pourrait par exemple s'effectuer en ligne sur un site sécurisé, en lien avec l'application de vote, ou même physiquement. En effet, le votant est susceptible d'être amené à voter de nouveau. La structure de données stockant la correspondance entre chaque identifiant et chaque clé publique est également stockée sur ETHEREUM par l'intermédiaire d'un contrat.

A la fin de l'enregistrement, les canaux permettant aux votants d'interagir avec la blockchain sont fermés, et l'application est prête pour passer à la phase de génération des accréditations.

```
1  const BN = require("bn.js");
2  const crypto = require("crypto");
3
4  const accumulatorUtils = require("./gen/utils/accumulator");
5
6  const Ecclesia = artifacts.require("./Ecclesia.sol");
7  const Registration = artifacts.require("./Registration.sol");
8  const CredentialGeneration = artifacts.require("./CredentialGeneration.sol");
9  const Commitment = artifacts.require("./Commitment.sol");
10
11 // Deployment Arguments
12 // Registration
13 const currentTime = parseInt((new Date().getTime() / 1000).toFixed(0));
14 const RegistrationArgs = [
15     currentTime + 10,
16     currentTime + 12,
17     "secp256k1",
18     "SHA1"
19 ];
```

FIGURE 6 – Initialisation de l'enregistrement – SOLIDITY

III.2 Génération des accréditations

A l'issue de l'enregistrement, nous entrons dans la deuxième phase du protocole : la génération des accréditations. Le but de celle-ci est avant tout d'assurer que les seules personnes pouvant voter sont effectivement membres de la liste officielle des électeurs, et que ces derniers ne peuvent le faire qu'une seule fois.

• L'accumulateur

La difficulté de cette partie du protocole réside dans la nécessité de vérifier que le votant appartient bien à la liste, sans pour autant révéler les autres membres de la liste. Ainsi, nous utilisons un accumulateur, qui permet de s'assurer qu'il n'y a pas de fraude. Il s'agit d'une primitive cryptographique – en d'autres termes un algorithme – qui permet de prouver qu'un élément appartient à un ensemble donné, sans en dévoiler les autres éléments. Il va ainsi de paire avec la Zero-knowledge proof, que nous avons introduit précédemment.

L'accumulateur, également appelé RSA accumulator, fonctionne comme suit : chaque votant est représenté par un entier premier, qui rentre en compte dans le calcul de l'accumulateur dès qu'il vote. [4] Plus précisément, à l'instant initial, l'accumulateur A n'est autre qu'un un nombre tiré aléatoirement : le générateur. De plus, on tire N un entier produit de deux grands nombres premiers. Si un votant, dont la clé est notée w, entame la procédure de vote, l'accumulateur devient $A' = A^w \bmod N$. Ainsi, à l'issue de l'étape, l'accumulateur est devenu $g^{\Pi} \bmod N$, où Π est le produit des clés des votants. La preuve sans divulgation consiste alors à montrer, si l'on veut prouver que w est bien dans la liste des votants, qu'il existe un x tel que $(g^w)^x = A$.

Néanmoins, les entiers calculés pouvant atteindre de très grandes tailles, une procédure plus astucieuse est implémentée dans notre application. En effet, le plus grand entier naturel dans Solidity correspond au type uint256 ($2^{256}-1$), ce qui n'est pas suffisant pour l'implémentation de l'accumulateur. Ainsi nous utilisons l'implémentation qui consiste à représenter les entiers par une liste d'objets de type uint256 [11]. On y trouve les différentes opérations (addition modulaire, substitution, exponentiation...). L'exponentiation modulaire peut être calculée sur Ethereum à l'aide d'une précompilation, qui est incluse dans toutes les implémentations client à l'adresse

0x05 depuis la mise à jour majeure d'Ethereum, appelée fork de Byzance , qui a notamment permis de renforcer sa sécurité ainsi que son adaptabilité. Les précompilations peuvent être appelées à partir d'un contrat intelligent Solidity à l'aide du code d'assemblage. Ce dernier désigne le langage de plus bas niveau qui représente le langage machine sous une forme lisible par un humain.

Nous nous heurtons à un problème : la multiplication n'est pas est implémentée. Nous nous en sommes approchés en utilisant la relation astucieuse : $4ab = (a+b)^2 - (a-b)^2$, mais aucune fonction ne permet la division par 4. Nous avons alors fait recours à l'assembleur Solidity afin de contourner le problème et de l'implémenter "à la main". Nous décrivons dans l'annexe 2 ce procédé. Ainsi, l'accumulateur peut effectivement être implémenté.

• À propos des nombres premiers

Nous aurons ainsi remarqué que la génération de nombres premiers est fondamentale pour le codage de nos différents algorithmes, comme il en est coutume en cryptographie.

Comme nous l'avons avancé dans le rapport intermédiaire, la question de la méthode de génération des nombres premiers a été problématique : nous avons dû choisir entre des méthodes fournissant des résultats sûrs mais chères en termes de temps, et des méthodes probabilistes, plus efficaces pour l'obtention de grands nombres premiers. Nous nous sommes finalement tournés vers l'une de ces dernières. Le fonctionnement est le suivant : nous appliquons un test de primalité, dans notre cas, le test de Miller Rabin, jusqu'à obtenir un nombre qui le valide, dont on affirmera alors qu'il est premier.

Néanmoins, le test de Miller-Rabin étant probabiliste, nous ne pouvons que donner une probabilité, aussi proche de 1 soit-elle, que le nombre testé positivement soit effectivement premier. Plus précisément, le test ne peut pas donner de faux-négatif (s'il indique que le nombre n'est pas premier, il a forcément raison), et se trompe avec une probabilité qui décroit exponentiellement avec le nombre d'essais lorsque le nombre n'est pas premier.

Ayant conscience que nous ne sommes pas des développeurs chevronnés, et ayant la volonté de produire une implémentation aussi cryptographiquement sécurisée que possible, nous avons fait le choix de ne pas coder nous-mêmes l'algorithme décrit ci-dessus, mais d'avoir recours à une im-

plémentation reconnue fiable par la communauté. La difficulté première fut donc de nous documenter et de peser le pour et le contre des bibliothèques disponibles et offrant cette possibilité. Notre premier choix se porta sur la bibliothèque bigint-crypto-utils de Javascript, étant open-source et disposant d'une fonction renvoyant directement un nombre premier probable aléatoire de taille arbitraire : nous pouvions alors directement lire le code de l'algorithme pour comprendre son fonctionnement et ses défauts. Celui-ci vérifie que les 250 plus petits nombres premiers ne sont pas des diviseurs du nombre à tester, puis applique 16 fois l'algorithme de Miller Rabin avec des entiers choisis aléatoirement. Ainsi, avec des entiers premiers à tester de taille usuelle, la probabilité d'un faux positif est de l'ordre de un sur un milliard. Néanmoins, le créateur de cette librairie explique que son utilisation d'opérations en temps non constant pouvait nuire à l'utilisation en pratique de sa librairie. Nous avons donc finalement opté pour le module crypto de Node.js, dont nous n'avons pu lire le code trop long et détaillé, mais bien plus utilisé par la communauté.

• Interactions avec la blockchain

Le début de la phase de génération des accréditations est marqué par l'ouverture de la communication avec la blockchain, qui accepte désormais les messages de l'application.

Les votants se voient alors attribuer deux clés : une accréditation de vote, et un secret, ce dernier contenant un numéro de série et une graine, chacun choisi aléatoirement. Une liste des accréditations de votes est également conservée et mise à jour à chaque vote, afin d'éviter qu'un votant ne vote plusieurs fois. Celle-ci n'est pas conservée sur la blockchain pour des raisons de coût spatial, mais un hash, c'est à dire un entier image de cette liste par une fonction, y est stocké. Si la liste est modifiée, son hash changera considérablement, et il ne correspondra plus à celui stocké sur la blockchain.

Avant d'envoyer son accréditation de vote à la blockchain, le votant le signe avec une signature digitale sécurisée du type EU CMA (Existential Unforgeability under a Chosen Message Attack). Ce type de signature, inventé en 1983 par Goldwasser, Micali et Rivest, permet de réduire la probabilité pour qu'un individu puisse générer un vote à la place d'un votant tout en se faisant passer pour lui. [14]
Après le déploiement des accréditations de vote, les canaux de communication avec la blockchain se referment, et il n'est plus possible de communiquer avec elle.

Au niveau de l'implémentation, nous avons été amené à rajouter un fichier de configuration au projet, afin de rassembler les informations qui dépendront de l'utilisateur :

```
.env
1  HOST=localhost
2  PORT=8545
3  FROMADDRESS=0xcCaEFAe6A6E90CBF5e59244E0399d497D1291065
4  PRIVATEKEY=0x678dcb2a1ac50c372e46f9f4102e397ed3c207b8b6dcd3c885d89028f1b007aa
5  PYTHONPATH=C://ProgramData//Anaconda3//python.exe
6  SCRIPTPATH=timelock/timelockpuzzle
```

FIGURE 7 – Fichier de configuration – Génération des accréditations

Nous avons alors déclaré une commande `require`(), ce qui permet d'utiliser le code générique suivant :

```
22      const credGenContract = await new ContractLibrary(
23          process.env.HOST,
24          process.env.PORT,
25          process.env.FROMADDRESS,
26          process.env.PRIVATEKEY
27      ).connectToCredentialGeneration();
```

FIGURE 8 – Appel des variables – Génération des accréditations

Ceci nous a permis d'uniformiser notre code afin qu'il soit utilisable sur tout support sur lequel l'application de vote sera installée.

III.3 VOTE

A l'issue de la génération des accréditations, la phase de vote en lui même peut démarrer. C'est dans cette phase que l'ensemble des éléments étudiés précédemment prennent leur sens. Les différents systèmes de sécurité, tels

que le Time Lock Encryption, la Zero-knowledge proof, et l'accumulateur sont mis en action.

Les votants décident de l'entité pour laquelle ils souhaitent voter. A la soumission de leur vote, plusieurs processus sont entraînés :

— Le Time Lock Encryption permet de chiffrer le vote. L'application côté votant garde un exemplaire de la clé de déchiffrement, et la révèlera à l'autorité de vote à l'heure du dépouillement si le votant l'accepte. Sinon, elle sera calculée par l'autorité de vote, qui commence le calcul dès la soumission du bulletin.
— L'accumulateur permet de vérifier que le vote à comptabiliser est effectivement associé au votant, et que ce dernier n'est pas un usurpateur. Celui-ci vérifie, comme nous l'avons vu précédemment, que le votant fait partie de la liste des électeurs. La liste tenue à jour, dont le hash est présent sur la blockchain, permet également de vérifier que le vote n'est soumis qu'une seule fois.
— Une signature Zero-Knowledge est calculée, et transmise avec le bulletin à la blockchain.

Une fois que la blockchain reçoit le bulletin signé, la signature Zero-Knowledge est vérifiée. Elle permet de contrôler l'authenticité du vote : celui-ci ne doit pas avoir été corrompu. Si celle-ci est correcte, le vote est enregistré, l'accumulateur est mis à jour, tout comme la liste des votants.

Remarquons qu'à ce stade, le vote est sur la blockchain, mais il ne peut être lu par l'autorité de vote, ni par quiconque. En effet, il est crypté par le Time Lock Encryption, et le temps minimum pour le décrypter ne permettra pas de connaître son contenu avant le dépouillement. De plus, la clé liant le votant à son bulletin étant inconnue du public ou de l'autorité de vote – c'est le principe de la Zero-Knowledge proof – personne ne pourra remonter au votant une fois le bulletin ouvert.

III.4 Dépouillement

Le dépouillement a lieu à l'issue de la phase de vote. L'heure du dépouillement a été annoncée précédemment par l'autorité de vote. Comme nous l'avons expliqué dans la partie dédiée au Time Lock Encryption, lors du dépouillement, les votants ont la possibilité de révéler leur clé de déchiffrement, afin d'éviter à l'autorité de vote d'avoir à résoudre les problèmes

Time Lock de chaque bulletin. Toutefois, pour ceux qui refusent de transmettre la clé, quelle que soit la raison de ce refus, l'autorité de vote termine la résolution des problèmes associés (rappelons que celle-ci a commencé dès la réception du bulletin).

La liste des résultats est tenue à jour au fil du dépouillement. À chaque ajout d'une voix, un hash de cette liste est déployé sur la blockchain, afin de vérifier que la liste n'est pas modifiée.

Dans la pratique, il est possible d'obtenir des valeurs relativement précises pour la capacité de calcul d'un ordinateur donné. Ainsi, en travaillant sur un ordinateur récent, pour une durée de 3 minutes nous avons pu créer un problème Time Lock résolu en 179 secondes, ce qui correspond à un écart relatif de $0,5\%$, ce qui reste tout à fait respectable dans le cadre d'une élection.

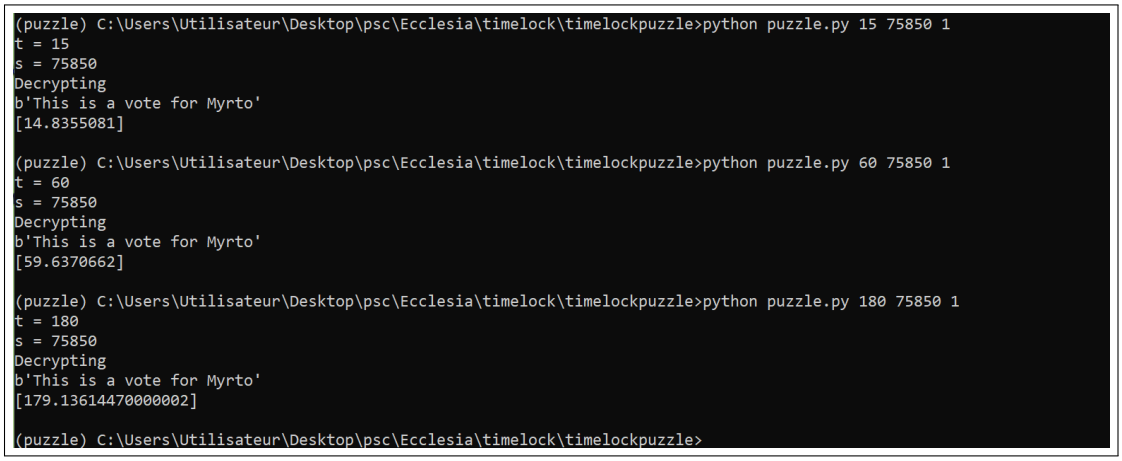


FIGURE 9 – Résolution du Time Lock Encryption - Ligne de commande

Revenons sur l'implémentation que nous proposons pour la phase de dépouillement. Celle-ci peut être composée en deux parties. Tout d'abord, les votants prennent l'initiative de révéler anonymement leurs votes ainsi que leurs nombres premiers secrets pour le time-lock. Afin de rendre cela possible, nous avons rajouté au code existant une commande reveal, ainsi que des fichiers permettant de communiquer avec les contrats SOLIDITY commitment.sol et Opening.sol. Cette commande prend les informations nécessaires de la machine locale (nous avons stocké ces informations dans le fichier myVote.json au moment du vote) en passant toujours par le stockage décentralisé IPFS, et révèle finalement le hash obtenu.

```
cli.action.start(`  Storing your vote parameters on IPFS`);
instance = await new StorageLibrary().connectToStorage();
const writtenValues = await instance.write('voteP', vPrimes);
const retrievedValues = await instance.read(writtenValues.hash);

if (vPrimes !== retrievedValues) {
    throw `vote parameters could not be stored on IPFS`;
}
cli.action.stop()
// store IPFS hash on contract
cli.action.start('  Submitting vote parameters hash to contract');
const feedback = await OpeningContract.reveal(writtenValues.hash);
```

FIGURE 10 – Révélation des votes par les votants

La deuxième partie commence quand le délai pour révéler ses informations est passé. L'autorité procède alors au décryptage des votes restants, puis à la mise à jour sur la blockchain du vote, en laissant au public de vérifier que la clé décryptée est bien la clé utilisée par le votant.

```
const [n, a,t, enc_key, enc_vote] = cvote.split(' ');
const pythonLibrary = new PythonLibrary(
    process.env.PYTHONPATH,
    process.env.SCRIPTPATH
).connectToPython();
const decrypted = await pythonLibrary.decrypt(n, a, t, enc_key, enc_vote);
console.log(decrypted);
```

FIGURE 11 – Déchiffrement des votes non-révélés

Il est à noter que tout le monde peut accomplir ces calculs, mais que la mise à jour des votes dans la blockchain est réservée à l'autorité.

```
require(msg.sender == electionAuthority,"this functionality is only for EA");

if (votes[_dishonestVoter].size == 0) {
    votes[_dishonestVoter] = IPFS.Multihash(_digest, _hashFunction, _size);
}
```

FIGURE 12 – Seule l'autorité de vote peut mettre à jour les votes

IV
BILAN DE NOTRE TRAVAIL

IV.1 À propos de la gestion du travail

L'un de nos objectifs premiers en choisissant ce PSC était de mettre à l'épreuve nos capacités de travail en groupe. En effet, il nous semblait que ce projet, à la dimension internationale et faisant intervenir de nombreux acteurs, nous donnerait un avant-goût de ce qui pourrait nous attendre dans le futur en milieu professionnel ou académique. Il y a en outre eu deux dimensions à notre travail collectif : la communication avec les agents extérieurs, et notre organisation interne.

Nous avons déjà souligné dans le rapport intermédiaire les difficultés et les défis qui naissaient du nombre conséquent d'acteurs ayant intervenu dans notre projet. Plus récemment, la fondation Electis a pris un rôle plus important en nous contactant afin que nous participions à la première campagne de vote de l'implémentation d'E-cclesia par des développeurs professionnels sur la blockchain Tezos.

De plus, nous avons progressé de manière conséquente dans l'organisation interne de notre groupe. En effet, au début de notre PSC, en septembre, nous envisagions de tous acquérir les mêmes compétences afin d'avoir un socle commun sur lequel fonder notre travail futur, et de telle sorte que le travail effectué par l'un d'entre nous puisse être repris par un autre. C'est pourquoi nous avions décidé initialement de tous apprendre simultanément les principes de la blockchain ainsi que le langage Solidity permettant de coder sur Ethereum. Cependant, nous nous sommes rapidement aperçus, avec l'aide de notre tuteur Julien Prat, que cela n'était pas possible, et que tous progresser de manière parallèle n'était pas envisageable à l'échelle du PSC. Alors nous avons décidé de diviser notre groupe en deux : Solène et David acquerront les bases cryptographiques nécessaires à la compréhension du protocole E-cclesia, Anass et Mohamed apprendront à coder en Solidity et en Javascript, de telle sorte que le nombre de compétences à acquérir à la fin soit réduit et que l'on puisse commencer le travail concret immédiatement après cette étape.

Cette organisation a survécu jusqu'à la fin du PSC, prouvant ainsi sa pertinence, mais faisant également naître une autre difficulté évoquée dans le rapport intermédiaire : comment transmettre entre deux équipes aux compétences radicalement différentes les consignes et les notions nécessaires à la bonne progression du projet ? Cette question, restée sans réponse claire au moment de la rédaction dudit rapport, a depuis progressivement mis en évidence qu'il était tout simplement nécessaire que chaque groupe assimile une partie des compétences de l'autre groupe. Cependant, certaines habitudes, comme des réunions PSC régulières dans une salle vacante des logements de la section équitation, et des outils informatiques d'organisation et de partage des données (Trello, Dropbox, GitHub) ont permis de surmonter ces difficultés, d'autant plus accentuées par la crise du coronavirus et la fermeture de l'école début mars.

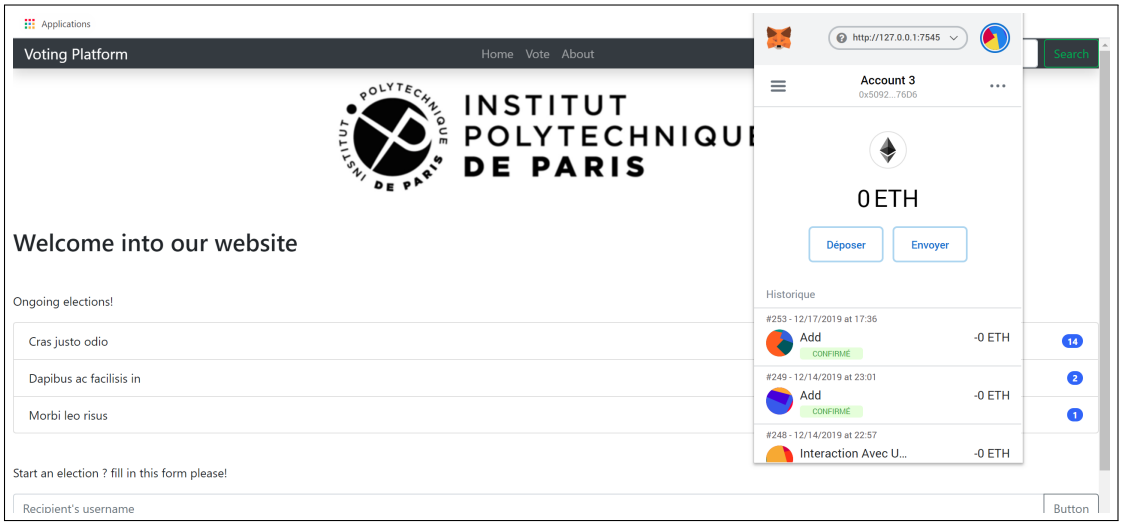
IV.2 Notre application de vote

Nous avons donc été amené, durant notre projet, à réaliser une application de vote décentralisée, interagissant avec la blockchain Ethereum par le biais de contrats intelligents.

Bien que certaines composantes de l'application ne soient pas encore fonctionnelles, telles que la Zero-knowledge proof qui n'a pas encore été entièrement implémentée, l'application permet tout de même de réaliser le vote décentralisé. Si notre système de vote est opérationnel, il est toutefois nécessaire de passer par la ligne de commande pour la lancer, l'interface n'ayant pas encore été développée.

Grâce à la commande `>truffle migrate -reset`, nous déployons l'application décentralisée. Cette opération s'élève à un coût de 0.1175688 ETH.

Nous lançons alors l'application locale http ://localhost :3000/ par la commande `>yarn run start`. Remarquons que nous avons utilisé Node.js pour le développement backend.



En se rendant à la page "Vote", nous pouvons effectuer un vote pour tester le fonctionnement des différentes interfaces de programmation, ainsi que la liaison avec la blockchain.

```
12  const ipfsClient = require('ipfs-http-client')
13  const ipfs = ipfsClient({ host: 'ipfs.infura.io', port: 5001, protocol: 'https' })
```

```
17  async componentWillMount() {
18    await this.loadWeb3()
19    await this.loadBlockchainData()
20  }
```

```
22  async loadWeb3() {
23    if (window.ethereum) {
24      window.web3 = new Web3(window.ethereum)
25      await window.ethereum.enable()
26    }
27    else if (window.web3) {
28      window.web3 = new Web3(window.web3.currentProvider)
29    }
30    else {
31      window.alert('Non-Ethereum browser detected. You should consider trying MetaMask!')
32    }
33  }
```

La transaction est passée dès la confirmation de la part du votant. Nous avons la certitude que le vote est stocké sur le stockage externe, et que le hash de la liste des votes a été déployée sur la blockchain.

Le vote pourra ensuite se dérouler comme selon le protocole, et la révélation des résultats aura lieu après le dépouillement, qui nécessitera ou non que l'autorité de vote déchiffre des bulletins.

À la lumière de nos avancées dans le projets nous pouvons maintenant conclure sur la réalisation des objectifs que nous nous étions fixés :

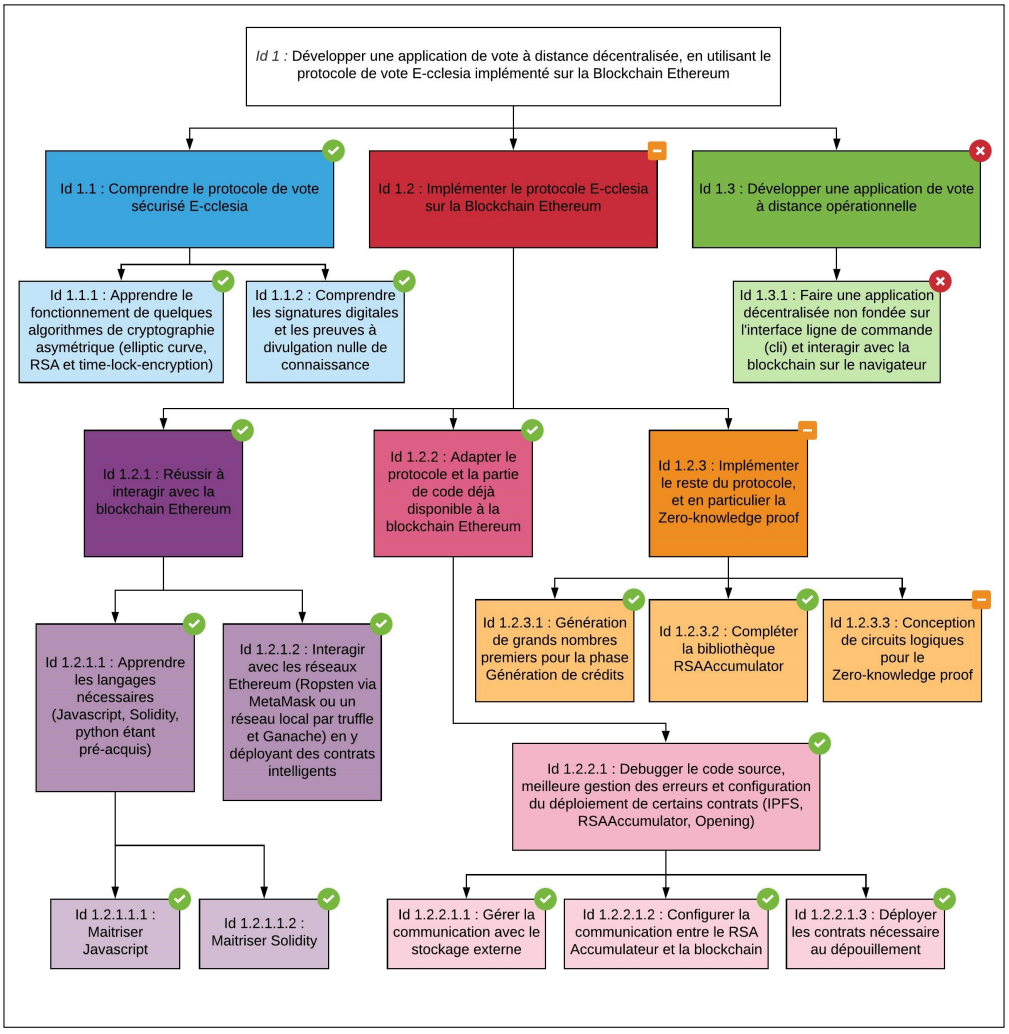


FIGURE 13 – Validation des objectifs au rendu du rapport

CONCLUSION

Ce PSC fut donc l'opportunité pour nous de retenir de nombreuses leçons précieuses. En effet, jamais nous n'avions eu la chance de travailler sur un sujet scientifique aussi complexe, dans une telle autonomie, et qui est aujourd'hui encore un sujet de recherche. Nous avons donc appris à nous renseigner, à explorer l'état de l'art et en extraire une bibliographie. Il nous fut nécessaire de travailler en équipe et de savoir demander conseils et explications à nos encadrants. Nous avons également acquis de nombreux savoirs et savoir-faire : langages de programmation Solidity ou Javascript, notions de blockchain et de cryptographie, autant de compétences qui pourraient nous servir à l'avenir. Mais la leçon principale du PSC est certainement celle du travail en groupe, de l'organisation, de la communication et de la coordination.

Nous nous étions donnés pour objectif de ressortir de notre PSC avec un système de vote par blockchain fonctionnel. Bien que cet objectif soit finalement réalisé, la solution apportée ne satisfait pas toutes les contraintes et sophistications que nous percevions comme nécessaires. Cependant, une équipe de développeurs professionnels a travaillé en parallèle sur le même protocole afin de l'implémenter sur la blockchain Tezos : un premier vote test de cette implémentation, auquel nous prendrons part, aura lieu le 30 avril et fera participer de nombreuses universités internationales, comme Stanford, le King's College de Londres ou l'Université d'Édimbourg. Nous avons donc hâte de voir les étapes suivantes des projets encouragés par la fondation Electis.

RÉFÉRENCES

[1] Alan T. Sherman, Russell A. Fink, Richard Carback, and David Chaum, *Scantegrity iii : automatic trustworthy receipts, highlighting over/under votes, and full voter verifiability*, (2011).

[2] Ben Adida, *Web-based open-audit voting*, IEEE Transactions on Information Forensics and Security (2008).

[3] Gwern Branwen, *Time-lock encryption - gwern.net*, 2011.

[4] Buterin, *Rsa accumulators for plasma cash history reduction*, 2018.

[5] Chris Dannen, *Introducing Ethereum and Solidity : Foundations of Cryptocurrency and Blockchain Programming for Beginners*, (2017).

[6] Chris Skidmore, *Cabinet office consolidated fund standing service*, (2017).

[7] David Gerard, *Attack of the 50 Foot Blockchain, Bitcoin, Blockchain, Ethereum and Smart Contracts*, Chapter 10 : Smart contracts, stupid humans.

[8] Ivaylo Genev, *Zerovote implementation : Electronic voting on the ethereum blockchain*, Undergraduate Honors Thesis (2018).

[9] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi, *How to build time-lock encryption. Designs, Codes and Cryptography*, (2018), 86(11) :2549–2586.

[10] Jonathan Levi, *E-cclesia Implementation using Time-Lock Encryption*, (2019).

[11] Matter Labs, *Implementation of rsa accumulator for plasma history reduction.*

[12] Lenka Marekova, *Zerovote : Self-tallying e-voting protocol in the uc framework*, Undergraduate Honors Thesis (2018).

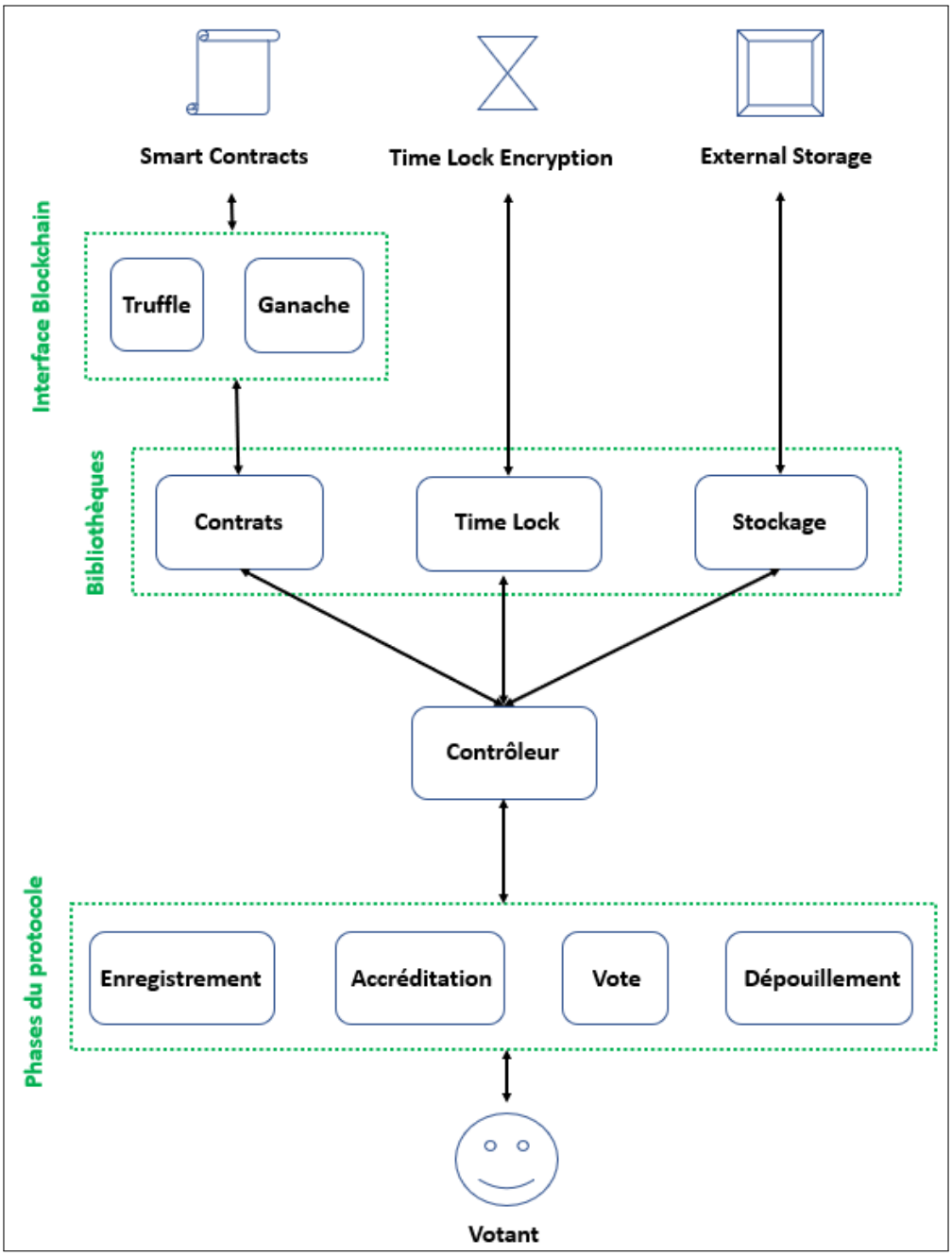
[13] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia, *Prêt à voter : a voter-verifiable voting system*, IEEE Transactions on Information Forensics and Security (2009).

[14] Ran Canetti, *Introduction to Cryptography*, Data Signatures **8** (2008).

[15] Ronald L Rivest, AdiShamir, and David A Wagner, *Time-lock puzzles and timed-release crypto. Technical report*, (1996).

Annexe 1

Schéma fonctionnel du système de vote



Annexe 2
Implémentation de la division modulaire sur Solidity

Afin d'implémenter la division par 4 sur SOLIDITY, il nous a fallu en apprendre davantage sur sa gestion de la mémoire interne. SOLIDITY SOLIDITY gère la mémoire d'une manière très simple : un «pointeur de mémoire libre» est stocké à la position $0x40$ en mémoire. On y stocke la longueur du dividende, qui est $0x20\times$(nombre de membres du dividende).

```
458         assembly {
459             memoryPointer := mload(0x40)
460             mstore(memoryPointer,length)
461         }
```

Nous rajoutons ensuite chaque membre du dividende à la mémoire (remarquons que 0x20 est la représentation hexadécimale de 32) :

```
463         uint256 limb = 0;
464         for (uint256 i = 0; i < NlengthIn32ByteLimbs; i++) {
465             limb = dividend[i];
466             dataLength += 0x20;
467             assembly {
468                 mstore(add(memoryPointer, dataLength), limb)  // cycle over dividend
469             }
470         }
471         assembly {
472             result_ptr := add(memoryPointer, dataLength)
473         }
```

Ensuite, nous pouvons procéder à la division par 4, qui correspond au décalage de deux bits à droite :

```
475         for(uint256 j = NlengthIn32ByteLimbs-1; j>=0;j--){      //for each word:
476             assembly{
477                 word_shifted := mload(result_ptr)               //get next word
478                 switch eq(j,0)                                  //if i==0:
479                 case 1 { mask := 0 }                            // handles msword: no mask needed.
480                 default { mask := mload(sub(result_ptr,0x20)) } // else get mask.
481             }
482             word_shifted >>= value;                             //right shift current by value
483             mask <<= mask_shift;                                // left shift next significant word by mask_shift
484             assembly{
485                 mstore(result_ptr, or(word_shifted,mask))
486             }                                                   // store OR'd mask and shifted value in-place
487             result_ptr -= 32;                                   // point to next value.
```

Finalement, afin de déterminer le quotient, nous rassemblons le résultat dans une liste :

```
500         limb = 0;
501         dataLength = 0x20; // skip length;
502         for (uint256 ii = 0; ii < NlengthIn32ByteLimbs; ii++) {
503             assembly {
504                 limb := mload(add(result_ptr, dataLength))
505             }
506             result[ii] = limb;
507             dataLength += 0x20;
508         }
509         return result;
```

INDEX