Hes·so VALAIS WALLIS

Haute Ecole de Gestion
Hochschule für Wirtschaft

Filière Informatique de gestion
Studiengang Wirtschaftsinformatik

Travail de Bachelor 2024

Les gardiens de la blockchain

Comment intégrer la récupération sociale dans les wallets

Etudiante : Céline Vialard

Professeur : Jean-Luc Beuchat

HES-SO Valais-Wallis • rte de la Plaine 2 • CP 80 • 3960 Sierre
+41 58 606 93 11 • info.informatique@hevs.ch • www.hevs.ch/ig

swissuniversities

EQUAL-SALARY CERTIFIED

Valais excellence

Résumé

Cette thèse, intitulée *"Les gardiens de la blockchain : comment intégrer la récupération sociale dans les wallets"*, explore la conception d'un mécanisme pour la récupération de portefeuilles numériques en cas de vol ou de perte. Le système proposé repose sur de la récupération sociale, utilisant des entités de confiance appelées *gardiens*. Ce projet s'inscrit dans le contexte croissant de l'adoption des technologies de la blockchain et des portefeuilles numériques, en répondant aux défis liés à la sécurité et à la continuité d'accès aux actifs numériques.

La recherche s'est concentrée sur le développement de *smart contracts* en *Solidity* sur la plateforme *Ethereum*. Une analyse exhaustive des méthodes existantes de récupération de clés a été réalisée, suivie de la création d'un *smart contract* jouant le rôle de *wallet*. Ce *wallet* est conçu pour être récupérable en cas de perte de l'accès à la clé privée, grâce à l'intervention des *gardiens*. Les solutions proposées visent à garantir un haut niveau de sécurité et d'anonymat pour les utilisateurs lors des opérations de récupération de clés.

Mots clés : blockchain, portefeuille numérique, récupération sociale, gardiens, sécurité, anonymat, smart contract, solidity, ethereum, méthodes de récupération de clés

Remerciements

Je tiens à remercier chaleureusement les personnes suivantes pour l'aide qu'elles m'ont apporté durant mon travail de Bachelor :

— **Jean-Luc Beuchat**, enseignant responsable du travail, pour son aide, son suivi et ses conseils tout au long du travail.

— **Martin Simon**, collaborateur dans le projet principal, pour sa relecture.

— **Julien Le Bras**, mon mari, pour son écoute et sa patience afin d'améliorer mes réflexions.

— **Tanya Amaya**, bibliothécaire, pour sa relecture et l'amélioration de la bibliographie.

Table des matières

Table des figures

1 Introduction

1.1 Contexte

Ce travail de Bachelor s'inscrit dans le cadre d'un projet de recherche intitulé *"Smart Wallet Architecture Guaranteeing Anonymity During Social Key Recovery Operations"*. Dans un contexte où les technologies de la blockchain et des portefeuilles numériques prennent une place de plus en plus prépondérante, l'objectif principal de cette recherche est de concevoir un portefeuille numérique récupérable. Ce portefeuille doit pouvoir être reconstruit de manière sécurisée et anonyme en cas de vol ou de perte, répondant ainsi à une problématique cruciale de sécurité et de continuité d'accès aux actifs numériques.

Avec l'accroissement des recherches et des applications visant à créer des identités électroniques sécurisées, il devient impératif de développer des solutions de récupération de clés efficaces et respectueuses de l'anonymat des utilisateurs. Ce projet s'inscrit dans cette dynamique en cherchant à garantir que les opérations de récupération de clés puissent être menées sans compromettre la confidentialité des utilisateurs.

1.2 Objectifs

Le principal objectif de cette thèse est de développer un portefeuille numérique capable de réaliser des transactions financières tout en permettant la modification de la clé privée via un mécanisme de *social recovery*. Ce mécanisme repose sur l'utilisation de gardiens, qui sont des entités ou des personnes de confiance désignées pour aider à la récupération de la clé privée en cas de besoin. Cela implique la création d'un système sécurisé et anonyme pour la récupération des clés, assurant ainsi la continuité de l'accès aux actifs numériques sans compromettre la sécurité ou la confidentialité des utilisateurs.

1.3 Sujet traité

Cette recherche se concentre sur la création de *smart contracts* en *Solidity* sur la plateforme *Ethereum*. L'objectif est d'explorer et de développer des solutions innovantes pour la gestion et la récupération des clés privées, en s'appuyant sur les capacités de programmation de la blockchain Ethereum. Les *smart contracts* permettront d'automatiser les processus de récupération de clés et de garantir leur sécurité et leur anonymat.

1.4 Méthodologie

Pour mener à bien cette recherche, une compréhension approfondie des *smart contracts* est cruciale, ainsi qu'une exploration des diverses méthodes de récupération de clés existantes. Une revue exhaustive de l'état de l'art sera réalisée, couvrant les solutions actuelles et les recherches en cours dans le domaine de la récupération de clés et de la gestion des portefeuilles numériques. Cette revue sera complétée par une formation continue sur les technologies associées, afin de rester à jour avec les dernières avancées dans ce domaine en constante évolution.

1.5 Environnement de travail

L'environnement de recherche inclura l'utilisation de différents portefeuilles numériques avec des clés privées pour effectuer une variété de tests pratiques. L'utilisation de Visual Studio Code sera nécessaire pour créer et exécuter les scriptes afin de déployer des *smart contracts* sur la plateforme Ethereum. Cette configuration technique permettra de développer, tester et valider les solutions proposées dans un cadre pratique et sécurisé. Les tests couvriront divers scénarios de récupération de clés, garantissant que les solutions développées sont robustes, sécurisées et respectueuses de l'anonymat des utilisateurs.

2 Présentation des concepts et des technologies

2.1 Blockchain

La blockchain est un registre de données décentralisé. Sa force réside dans l'impossibilité de modifier les données facilement, ce qui garantit leur intégrité et leur sécurité. Dans ce travail, il est crucial de bien comprendre la gestion des adresses. (HELBIG, 2019)

Il existe deux types d'adresses : les adresses de portefeuille (*wallet addresses*) et les adresses de contrat (*contract addresses*).

Les adresses de portefeuille sont associées à une clé privée qui permet de signer les transactions effectuées. Ces adresses sont utilisées par les portefeuilles pour effectuer des transactions sur la blockchain (SERAH, 2023 ; ALCHEMY, 2023a).

Les adresses de contrat, quant à elles, sont associées à des contrats intelligents (*smart contracts*). Elles n'ont pas de clé privée, car elles représentent du code qui s'exécute de manière autonome sur la blockchain (SERAH, 2023 ; ALCHEMY, 2023a).

2.2 Wallet

Un *wallet*, ou portefeuille électronique, est un logiciel permettant d'effectuer des transactions sur la blockchain. Il permet aux utilisateurs de stocker, recevoir et envoyer des cryptomonnaies ainsi que d'autres actifs numériques. Le portefeuille électronique contient une clé privée, qui est une information cruciale nécessaire pour signer les transactions et prouver la propriété des actifs. La sécurité de cette clé privée est essentielle : si elle est perdue ou compromise, l'accès aux fonds et aux données stockées dans le portefeuille est également perdu, entraînant une perte irréversible des actifs numériques (HELBIG, 2019).

Il existe deux types principaux de portefeuilles : les portefeuilles chauds (*hot wallets*) et les stockages froids (*cold storages*).

Portefeuille chaud : Un portefeuille chaud est connecté à Internet, ce qui permet aux utilisateurs d'effectuer des transactions directement sur la blockchain de manière rapide et pratique. Cependant, cette accessibilité accrue présente un risque plus élevé de compromission, car la clé privée est exposée aux menaces en ligne telles que les attaques de pirates informatiques (HELBIG, 2019).

Stockage froid : Un stockage froid stocke la clé privée hors ligne, minimisant ainsi le risque d'exposition à Internet et aux cyberattaques. Les portefeuilles froids comprennent des dispositifs matériels (*hardware wallets*), des portefeuilles papier (*paper wallets*) et d'autres formes de stockage physique. Bien que ce type de portefeuille offre une sécurité supérieure, il est moins pratique pour les transactions fréquentes, car la clé privée doit être accessible physiquement pour signer une transaction (HELBIG, 2019).

2.3 Smart Contract

Un *smart contract* est un morceau de code déployé sur la blockchain qui peut être exécuté de manière autonome. Il est généralement utilisé grâce à un portefeuille standard ou un autre *smart contract*. Il est possible d'écrire des *smart contracts*, par exemple, en Solidity ou en Rust (IBM, 2021 ; ETHEREUM, 2024c ; HAFZA, 2024).

Solidity est un langage qui a eu sa première version en 2014 qui est influencé par Python, C++, et JavaScript. C'est un langage spécialisé pour écrire des *smart contracts*, largement utilisé sur la blockchain Ethereum (WIKIPÉDIA, 2024b).

Rust, quant à lui, est un langage plus ancien (2006), inspiré de C mais avec une gestion avancée de la mémoire. Il est également utilisé pour créer des *smart contracts*, notamment sur des blockchains comme Solana et Secret Network (WIKIPÉDIA, 2024c).

2.4 Les gardiens

Les gardiens sont des personnes, des dispositifs matériels ou des entités de confiance désignés par l'utilisateur pour l'assister dans la récupération de l'accès à ses données en cas de besoin. Pour garantir une sécurité optimale, il est recommandé que les gardiens choisis ne se connaissent pas entre eux. En effet, la récupération des accès nécessite la collaboration de plusieurs gardiens, ce qui complique une éventuelle conspiration si ces derniers ne sont pas en contact (FLAYDEM, 2023b).

Il est crucial de désigner plusieurs gardiens, car c'est leur action concertée qui permet au détenteur de récupérer ses accès. Si les gardiens se connaissaient, ils pourraient potentiellement conspirer contre le propriétaire principal et usurper ses droits. Pour cette raison, il est essentiel que les gardiens restent anonymes et conservent leur anonymat tout au long du processus. Il faut bien que leur anonymat soit conservé pour éviter les attaque de type *Social engineering*. Il s'agit d'une attaque qui cible les gardiens en se faisant passé pour le propriétaire et demande le changement de propriétaire (FLAYDEM, 2023b).

3 État de l'art

3.1 Introduction

Deux principales méthodes ont été identifiées pour ce travail. Premièrement, l'utilisation de portefeuilles intelligents (*smart wallets*) qui gèrent les actifs d'un compte et autorisent certaines adresses à interagir avec eux. Deuxièmement, la création d'un portefeuille basé sur le *Calcul Multipartite Sécurisé* (MPC), permettant de stocker plusieurs fragments de clé différents et de signer les transactions sans jamais reconstruire à un endroit la clé principal.

3.2 Wallet MPC

Les *wallets* basés sur le *Calcul Multipartite Sécurisé* (MPC) visent à diviser la clé privée en plusieurs fragments ou parts. Ces fragments sont distribués entre différentes entités, de sorte que la clé privée complète ne peut être reconstruite. Cette méthode permet d'effectuer des transactions sans qu'aucune entité individuelle ne possède l'intégralité de la clé privée, augmentant ainsi la sécurité. La figure 3.1 illustre la fragmentation de la clé. Il est aussi possible de directement la fragmenté sans jamais connaître la clé principale. Cette illustration ne montre qu'un cas d'utilisation (FLAYDEM, 2023a ; FIREBLOCKS, 2024).

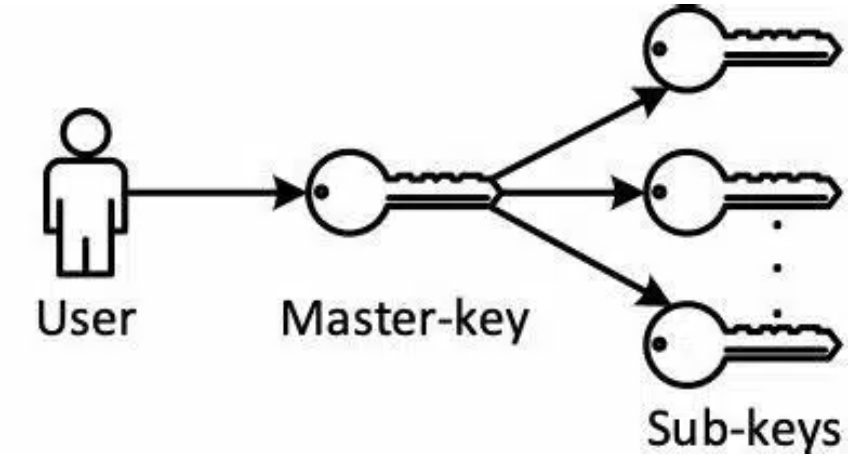

FIGURE 3.1 – *Schéma de la fragmentation de clé dans un wallet MPC*

Source : FLAYDEM*, 2023a*

L'objectif principal de cette approche est de permettre d'utiliser les clés afin de réaliser des transactions sans que les différentes entités impliquées ne puissent accéder individuellement à la clé complète. Cela garantit que même si certaines entités sont compromises, la clé privée reste protégée (FLAYDEM, 2023a ; FIREBLOCKS, 2024).

3.3 Utilisation des *wallets* basés sur MPC

Actuellement, plusieurs plateformes exploitent le concept du *wallet* MPC (Multi-Party Computation). Par exemple, Zengo, Coinbase, et Dfns utilisent chacune cette technologie pour offrir des solutions de gestion de portefeuilles numériques plus sécurisées.

Zengo met en avant la sécurité de son service, affirmant que son système est inviolable grâce au MPC. Cependant, il convient de noter qu'aucune solution n'est proposée pour changer l'accès en cas de compromission de la clé principale. De son côté, Coinbase permet d'intégrer leur *wallet* dans des applications et, à l'instar de Zengo, assure que les clés sont inviolables grâce au MPC. Quant à Dfns, elle propose une API facile à utiliser, avec la possibilité de tester gratuitement. Ce logiciel utilise la signature à seuil (*Threshold Signature Scheme*, TSS) comme méthode de MPC (ZENGO, 2024 ; COINBASE, 2024a ; DFNS, 2024).

En résumé, les *wallets* basés sur le MPC offrent une méthode avancée et sécurisée pour la gestion des clés privées, répartissant ainsi le risque et améliorant la résilience contre les compromissions potentielles.

3.4 NFT

Les *Non-Fungible Tokens* (NFT) sont des jetons numériques uniques et non interchangeables, principalement utilisés comme certificats d'authenticité dans divers domaines. Sur la blockchain Ethereum, les NFT sont régis par les protocoles `ERC-721` et `ERC-1155`, qui définissent les normes pour les *smart contracts* associés à ces actifs (WIKIPÉDIA, 2024a ; WARDZALA, 2023 ; LEROUX, 2023).

Depuis leur apparition, les NFT ont évolué pour s'adapter à des usages de plus en plus variés. Par exemple, les **NFT dynamiques** ont récemment fait leur apparition. Ces jetons peuvent changer de caractéristiques en fonction de conditions prédéfinies, comme des événements externes ou des interactions spécifiques, ouvrant ainsi de nouvelles possibilités d'application, notamment dans les jeux vidéo ou les objets de collection évolutifs (WIKIPÉDIA, 2024a ; WARDZALA, 2023 ; LEROUX, 2023).

Parmi les principales applications actuelles des NFT (WIKIPÉDIA, 2024a ; WARDZALA, 2023 ; LEROUX, 2023), on peut citer :

— **Art numérique :** Les NFT permettent de monétiser des œuvres d'art numériques en certifiant leur authenticité et leur propriété. Ils offrent ainsi aux artistes une nouvelle manière de vendre et de protéger leurs créations.

— **Immobilier virtuel :** Dans les métavers, les NFT servent à acheter et vendre des propriétés virtuelles, offrant ainsi un moyen sécurisé de gérer les transactions immobilières dans des environnements numériques.

— **Licences et brevets :** Les NFT peuvent représenter des licences ou des brevets, simplifiant leur transmission et leur gestion tout en assurant une traçabilité accrue.

— **Jeux vidéo :** Dans l'univers des jeux vidéo, les NFT permettent de posséder des objets virtuels ou des éléments de jeu, conférant aux joueurs une véritable propriété de leurs biens numériques.

L'un des principaux avantages des NFT réside dans leur caractère unique et leur capacité à conserver leur valeur dans le temps. Cependant, leur transférabilité peut poser des défis pour certaines applications, comme la certification de données personnelles, où l'immuabilité et la non-transférabilité sont essentielles (WIKIPÉDIA, 2024a ; WARDZALA, 2023 ; LEROUX, 2023).

3.5 Soulbound Token

Les *Soulbound Tokens* (SBT) sont des *NFT* non transmissibles, ce qui signifie qu'ils ne peuvent pas être transférés d'un portefeuille à un autre. Cette caractéristique unique les rend particulièrement adaptés à certaines applications où l'incapacité de transmettre les tokens est essentielle pour maintenir l'intégrité des informations associées (BLOCKGAMES, 2023 ; LEDGER, 2023 ; LEARNCRYPTO, 2024).

Parmi les principales utilisations des SBT (BLOCKGAMES, 2023 ; LEDGER, 2023 ; LEARNCRYPTO, 2024), on trouve :

— **Certificats de compétence :** Les SBT peuvent être utilisés pour lier des certificats de compétence à un portefeuille personnel. Par exemple, une université pourrait émettre un SBT à un diplômé pour attester de ses qualifications. Ce certificat numérique resterait toujours attaché au portefeuille du diplômé, garantissant son authenticité et empêchant toute falsification.

— **Dossiers médicaux :** Les SBT peuvent également être utilisés pour lier des dossiers médicaux à un portefeuille personnel. Cela permettrait aux patients de conserver un historique médical complet et sécurisé. Les professionnels de la santé pourraient accéder à ces informations en toute confiance, sachant qu'elles sont authentiques et inaltérables.

— **Autres données personnelles :** Les SBT peuvent lier toute autre donnée personnelle nécessitant une attestation de validité et de non-transférabilité, telles que des preuves d'identité, des diplômes professionnels ou des licences.

— **Gaming :** Les SBT peuvent être gagnés en accomplissant des haut-faits. Ces badges appartiennent donc au joueur ce qui peut lui permettre des avantages en jeu sur différents jeux.

Un autre avantage significatif des SBT est leur système de récupération par gardien. En cas de perte d'accès au portefeuille, le propriétaire peut désigner des gardiens, tels que des amis de confiance, des membres de la famille ou des institutions, pour aider à récupérer l'accès. Ce système renforce la sécurité et la résilience des portefeuilles utilisant des SBT, car il permet de restaurer l'accès sans compromettre la sécurité des informations sensibles (BLOCKGAMES, 2023 ; LEDGER, 2023 ; LEARNCRYPTO, 2024).

3.6 Smart Wallet

Un *smart wallet* est un portefeuille décentralisé basé sur des *smart contracts*.

Dans notre contexte, les *smart wallets* sont particulièrement innovants car ils permettent de déployer du code sur une adresse non associée à une clé privée. Ce code, intégré dans un *smart contract*, peut interagir directement avec la blockchain, par exemple pour recevoir ou envoyer des fonds. Une caractéristique distinctive de ces portefeuilles est la possibilité de définir par code une adresse propriétaire distincte, autorisée à effectuer certaines actions prédéfinies (SANKRIT K, 2024).

L'aspect le plus novateur réside dans la capacité de modifier dynamiquement le propriétaire de l'adresse, permettant ainsi à une nouvelle adresse de gérer les fonds détenus par le *smart contract*. Cette fonctionnalité assure un haut niveau de flexibilité tout en maintenant la sécurité et la décentralisation du portefeuille (SANKRIT K, 2024).

En pratique, ces portefeuilles peuvent exécuter des transactions financières en suivant les instructions programmées dans le *smart contract*. Par exemple, ils peuvent accorder des permissions spécifiques à certaines adresses pour réaliser des transactions ou modifier les droits d'accès. La possibilité de transférer la gestion des fonds à une autre adresse est une fonctionnalité clé, assurant une continuité sécurisée et décentralisée (SANKRIT K, 2024).

3.7 Utilisation des *Smart Wallets*

Actuellement, plusieurs plateformes permettent déjà de créer et de gérer des portefeuilles via des *smart contracts*. Par exemple , on retrouve Safe (anciennement Gnosis Safe), Coinbase et Soul Wallet

Safe permet de créer un *smart wallet* pour effectuer des transactions avec signature multiple. Cependant, un inconvénient de ce portefeuille réside dans le fait que les frais de transaction sont relativement élevés par rapport à d'autres solutions. Pour les développeurs, Safe offre une section dédiée leur permettant de concevoir des applications sur des comptes décentralisés. Le portefeuille intègre également le module *Safe RecoveryHub*, qui facilite la récupération du portefeuille. En plus de la récupération sociale, diverses méthodes alternatives peuvent être mises en place. Il est par ailleurs possible de créer des applications personnalisées basées sur cette technologie (MIGUEL, 2024 ; SAFE ECOSYSTEM FOUNDATION, 2024 ; SAFE, 2023).

Coinbase est principalement connue pour sa plateforme d'échange, ainsi que pour son portefeuille standard. Cependant, ils ont récemment introduit une nouveauté : le *smart wallet*, qui sera intégré dans le future à Coinbase Wallet. Ce nouveau développement simplifie l'utilisation des portefeuilles pour le grand public tout en réduisant les coûts de transaction, qui pouvaient varier en fonction de la charge sur le réseau. Ce *smart wallet* utilise une combinaison d'authentification biométrique et de techniques de sécurité avancées pour accéder au portefeuille. Le projet implémente également le standard ERC-4337. Ce développement se présente sous la forme d'une bibliothèque disponible pour les développeurs souhaitant intégrer les *smart wallets* dans leurs applications (RABBITCOIN, 2024 ; PORCELLI, 2024 ; COINBASE, 2024c).

Soul Wallet est un *smart wallet* open-source disponible sur GitHub, qui prend en charge la récupération sociale. Cette solution ne propose pas de librairie pour les développeurs. Mais il aborde l'un des défis majeurs des portefeuilles décentralisés en implémentant égallement le protocole ERC-4337, qui sera détaillé au chapitre 6.3. Bien que Soul Wallet soit actuellement limité à la blockchain *Ethereum* et à certaines solutions de seconde couche, il présente un avantage : la possibilité de régler les frais de transaction en tokens ERC-20, avec des coûts inférieurs à ceux des transactions standard (STRUCK, 2023 ; SOUL WALLET, 2024).

Dans les trois exemples que nous avons étudiés, il existe différentes manières d'utiliser les *smart contracts* pour créer des portefeuilles décentralisés. En effet, certains n'implémentent pas la récupération sociale mais utilisent d'autres technologies intéressantes permettant d'accéder aux portefeuilles. D'autres proposent diverses méthodes de récupération, offrant ainsi à l'utilisateur la possibilité de choisir celle qui lui convient le mieux. Dans le cas de Safe, on constate que les frais de transaction sont élevés, alors que pour les deux autres solutions, ces frais sont faibles, voire inexistants. Malheureusement, aucun d'entre eux ne mentionne l'anonymat des gardiens ou les moyens de récupération.

Pour mieux comparer les outils existants, il est essentiel de comprendre d'abord le fonctionnement de base des *smart wallets*.

3.8 Smart wallet vs MPC wallet

Dans ce travail, nous analyserons principalement les *smart wallets*. Étant donné que l'objectif est de pouvoir modifier la clé privée en cas de vol ou de perte, les *smart wallets* semblent plus adaptés. En effet, bien que les *MPC wallets* rendent le vol de la clé plus difficile, la rotation de clé est coûteuse et pas possible en cas de perte des accès, alors que les *smart wallets* peuvent le faire en toute circonstance.

3.9 Choix de la blockchain

Il existe différentes blockchains sur lesquelles il est possible de développer un *smart contract*. Le choix de la blockchain peut influencer le langage de programmation utilisé, ainsi que la méthode de développement et les caractéristiques du contrat. Plusieurs blockchains ont été envisagées pour ce projet pour leur compatibilité avec le *smart contract*, telles que Ethereum, Polygon, Solana et Secret Network.

3.9.1 Ethereum

Ethereum est une blockchain de référence, largement reconnue pour la gestion de la cryptomonnaie Ether (ETH). En tant que l'une des technologies blockchain les plus connues et développées, Ethereum se distingue par sa capacité à créer et exécuter des *smart contracts* ainsi que des applications décentralisées (*dApps*) (ETHEREUM, 2024b : ETHEREUM, 2024a).

L'une des forces majeures d'Ethereum réside dans son vaste écosystème, qui comprend de nombreuses bibliothèques, des outils de développement variés, et une documentation abondante. Cette richesse de ressources rend le développement et l'intégration d'applications sur Ethereum particulièrement accessibles, attirant ainsi une large communauté de développeurs et de projets innovants (ETHEREUM, 2024b : ETHEREUM, 2024a).

De plus Ethereum, c'est beaucoup amélioré ces dernières années concernant l'impact énergétique. En effet, depuis qu'ils ont passé du proof of work en proof of stack la consommation en énergie a grandement diminué (ALLISON, 2023).

3.9.2 Polygon

Polygon est une solution de seconde couche seconde couche construite sur Ethereum, conçue pour faciliter le déploiement d'applications décentralisées (*dApps*). Les tokens natifs utilisés sur cette blockchain sont appelés MATIC (COINBASE, 2024b ; POLYGON, 2024).

L'un des principaux objectifs de *Polygon* est de réduire les coûts de transaction et d'améliorer la vitesse des opérations par rapport à Ethereum, tout en maintenant une compatibilité totale avec l'écosystème Ethereum. En outre, *Polygon* met en avant son engagement écologique en adoptant des pratiques visant à réduire son empreinte carbone, ce qui en fait une option attrayante pour les projets soucieux de leur impact environnemental (COINBASE, 2024b ; POLYGON, 2024).

3.9.3 Solana

Solana est une blockchain de haute performance, spécialisée dans la gestion d'un réseau de données pour de nombreuses applications décentralisées. Elle se distingue par son expertise dans le développement de *smart contracts*, appelés "programs" dans son environnement.

Une caractéristique notable de *Solana* est la possibilité d'effectuer des déploiements continus, similaires à ceux réalisés dans des environnements de développement d'applications classiques, offrant ainsi une flexibilité accrue pour les développeurs (PICARDO, 2023 ; SOLANA, 2024).

L'objectif principal de *Solana* est de maximiser la performance. Grâce à une architecture optimisée, elle permet de traiter un nombre massif de transactions tout en maintenant des temps de latence très faibles. Cela rend *Solana* particulièrement adaptée pour les applications nécessitant une grande vitesse de traitement et une évolutivité sans compromis sur les délais de validation des transactions (PICARDO, 2023).

3.9.4 Secret Network

Secret Network est une blockchain spécifiquement conçue pour offrir un haut niveau de confidentialité. Contrairement à de nombreuses autres blockchains où les données et transactions sont généralement visibles par tous, *Secret Network* assure que même les *smart contracts* restent confidentiels. Cela signifie que les informations sensibles et les transactions privées sont robustement protégées, ce qui en fait une solution idéale pour les applications nécessitant un haut degré de confidentialité (STAN, 2022 ; SECRET NETWORK, 2024).

Secret Network est également compatible avec d'autres blockchains, telles qu'Ethereum, facilitant ainsi l'intégration et l'interopérabilité avec des systèmes existants. De plus, un certain nombre d'applications décentralisées (*dApps*) ont déjà été développées sur cette plateforme, tirant parti de ses capacités uniques en matière de confidentialité pour offrir des services sécurisés et privés (STAN, 2022 ; SECRET NETWORK, 2024).

3.9.5 Comparaison

Pour évaluer les différentes blockchains, nous utiliserons la grille suivante (DEBELLOIR, 2021 ; FLAYDEM, 2023a ; ARTHUR, 2024 ; CHAINSPECT, 2024b ; CHAINSPECT, 2024a) :

	Ethereum	Polygon	Solana	Secret Network
Langage	Solidity	Solidity	Rust	Rust
Confidentialité des smart contracts	Faible [1]	Moyenne	Faible	Élevée
Coût des transactions	Élevé	Faible	Très faible	Moyen
Complexité des smart contracts	Moyenne	Moyenne	Élevée	Moyenne
Communauté et support	Excellente	Bonne	Bonne	Bonne
Soutien et documentation	Excellente	Bonne	Bonne	Bonne
Utilisation énergétique	Moyenne	Très faible	Très faible	Faible [2]
Risque de congestion	Élevé	Moyen	Faible	Non disponible

[1] Ethereum n'a pas de confidentialité native, mais des solutions comme zk-SNARKs et des extensions comme Aztec ou Nightfall peuvent ajouter cette fonctionnalité.

[2] La mesure dépend de la blockchain de base utilisée

TABLE 3.1 – *Grille d'évaluation des blockchains pour un smart contract*

Cette grille permet de comparer les différentes blockchains analysées. Il en ressort que, pour un premier déploiement de *smart contract*, Ethereum est une option préférée dans le cadre de ce travail. En effet, c'est l'option qui offre le plus de documentation et de soutien, facilitant ainsi le déploiement et l'utilisation du *smart contract*.

De plus, étant donné qu'il s'agit d'un travail visant à comprendre le fonctionnement et le potentiel des *smart contracts*, Ethereum est plus accessible pour une exploration approfondie en raison de sa popularité et de son adoption généralisée.

4 | Mise en oeuvre

Toute la procédure du chapitre se base sur différentes procédures/documentations trouvées lors des recherches (HALPERN, 2021a ; HALPERN, 2021b ; de BALASY, 2023 ; TUSHARMAHAJAN, 2023 ; ALCHEMY, 2023b).

4.1 Principe du projet

Le but de ce développement est illustré dans le schéma suivant :

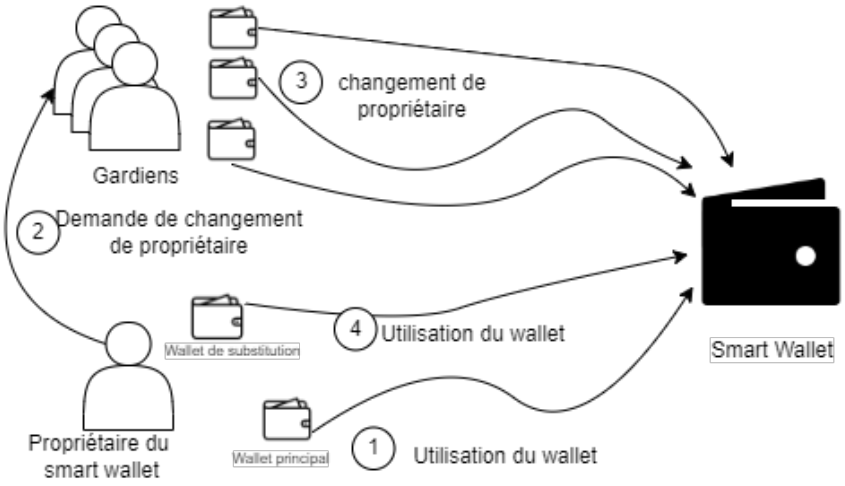

FIGURE 4.1 – *Schéma explicatif du fonctionnement du smart wallet*

Source de l'auteur

1. Le propriétaire utilise son *smart wallet* via son portefeuille principal.
2. Si le propriétaire perd son portefeuille principal ou se le fait voler, il peut bloquer l'accès à son *smart wallet* et demander à ses gardiens de transférer la propriété vers un portefeuille de substitution.
3. Chaque gardien soumet une demande de changement de propriétaire au *smart wallet*. Une fois qu'un nombre suffisant de demandes est validé, le transfert de propriété est effectué.
4. Le propriétaire peut alors accéder à nouveau à son *smart wallet* à l'aide du portefeuille de substitution.

Il est essentiel que les gardiens restent anonymes pour prévenir tout risque de collusion ou d'ingénierie sociale, comme illustré dans la Figure 4.2.

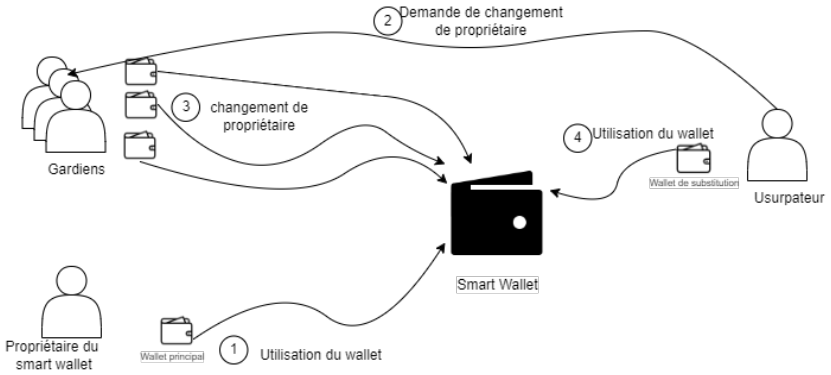

FIGURE 4.2 – *Schéma explicatif du smart wallet en cas d'ingénierie sociale*

Source de l'auteur

1. Le propriétaire utilise son *smart wallet* via son portefeuille principal.
2. Un usurpateur demande aux gardiens de transférer la propriété du *smart wallet* à l'adresse de son propre portefeuille.
3. Chaque gardien soumet une demande de changement de propriétaire au *smart wallet*. La transaction est validée lorsque le nombre requis de demandes est atteint.
4. L'usurpateur obtient alors l'accès au *smart wallet* via son propre portefeuille.

On peut voir que si l'usurpateur fait bien son travail les gardiens peuvent facilement le prendre pour le vrai propriétaire.

4.2 Création de compte

La première étape consiste à créer des comptes avec une clé privée. Pour cela, l'extension MetaMask sera installée. Pour les différents tests à effectuer, quatre comptes seront utilisés : deux comptes qui se transmettront les droits sur le *wallet* et deux autres qui joueront le rôle de gardiens.

4.3 Collecte d'ETH de test

Pour déployer le *smart contract*, il est nécessaire de disposer d'ETH, utilisé pour payer le *gas* requis par les transactions sur la blockchain. Ethereum propose des réseaux de test, comme *Sepolia*, qui permettent d'obtenir des ETH gratuits pour les tests. Au cours de cette recherche, il a été nécessaire de récupérer quotidiennement des ETH sur deux sites spécifiques, appelés *faucets*. Ces *faucets* distribuent des ETH de test gratuitement, bien que certains nécessitent un compte avec des ETH réels pour y accéder. Les demandes de distribution peuvent être effectuées toutes les 24 heures.

Les *faucets* utilisés, car ne nécessitant pas de compte disposant d'ETH réels, sont :

— Sepolia Faucet by Google Cloud Web3

— Sepolia Faucet Powered by Automata

4.4 Configuration du projet

4.4.1 Création du Projet sur Alchemy

Avant de commencer à programmer notre smart contract, il est nécessaire de créer un compte sur Alchemy pour créer une application. Cette plateforme sera utilisée pour publier et gérer le smart contract. Une fois l'application créée, l'onglet *network* fournit une clé API qui sera utilisée dans les configurations.

4.4.2 Installation

Ensuite, nous commençons par créer le projet. Ce projet sera réalisé en JavaScript. Pour cela, initialisez un projet npm avec la commande suivante :

```
npm init --yes
```

Ce projet utilise Hardhat comme environnement de développement. Pour l'installer, utilisez :

```
npm install --save-dev hardhat
```

Grâce à Hardhat, la création de projet est simplifiée :

```
npx hardhat
```

Parmi les options proposées, choisissez un projet vide. Une fois le projet créé, ajoutez les dossiers suivants : `contracts` et `scripts`, qui contiendront respectivement les smart contracts en Solidity et les scripts d'interaction avec le smart contract.

Ensuite, configurez l'environnement en installant `dotenv` pour la gestion des variables d'environnement :

```
npm install dotenv --save
```

Dans le fichier `.env`, ajoutez les lignes suivantes :

```
API_URL = "https://eth-goerli.alchemyapi.io/v2/[your-api-key]"
PRIVATE_KEY = "[your-metamask-private-key]"

# Clé privée account 1
PRIVATE_KEY_A1 = ""
# Clé privée account 2
PRIVATE_KEY_A2 = ""

```

```
# Clé privée Guardian 1
PRIVATE_KEY_G1 = ""
# Clé privée Guardian 2
PRIVATE_KEY_G2 = ""

MIXER_ADDRESS="0x230f1A8f005D6a62f96eCb03f90CEc3C6DA83dE7"
```

4.4.3 Librairie `@nomicfoundation/hardhat-ethers`

Ensuite, la librairie `@nomicfoundation/hardhat-ethers` sera utilisée pour les interactions avec la blockchain et pour compiler et déployer les smart contracts :

```
npm install --save-dev @nomicfoundation/hardhat-ethers
```

Il faut maintenant configurer `hardhat.config.cjs` comme suit :

```
/** @type import('hardhat/config').HardhatUserConfig */
require('dotenv').config();
require("@nomicfoundation/hardhat-ethers");

const { API_URL, PRIVATE_KEY } = process.env;

module.exports = {
  solidity: "0.8.24",
  defaultNetwork: "sepolia",
   networks: {
      hardhat: {},
      sepolia: {
         url: API_URL,
         accounts: [`0x${PRIVATE_KEY}`]
      }
   },
};
```

Enfin, pour avoir une meilleure idée de la structure du projet, en voici un résumé :

```
SmartContract
├── contracts
│   ├── SmartWallet.sol
│   └── SmartWalletMixer.sol
├── scripts
│   ├── app.js
│   ├── deploy.js
│   ├── interact.js
│   └── menu.js
└── artefacts
    └── contracts
        └── SmartWallet.sol
            └── SmartWallet.dbg.json
```

```
        SmartWallet.json
      SmartWalletMixer.sol
        SmartWalletMixer.dbg.json
        SmartWalletMixer.json
  node_modules
  cache
  .env
  hardhat.config.cjs
  package.json
```

4.5 Écriture du Smart Contract

L'écriture du *smart contract* utilise le langage Solidity. Il est impératif de commencer le document en spécifiant la version de Solidity utilisée.

```
3 pragma solidity >=0.8.24;
```

Pour créer le *smart wallet*, deux *smart contracts* sont utilisés : le premier est le *smart wallet* lui-même et le deuxième agit comme créateur de *smart contracts* et a également un rôle de *mixer* pour les gardiens demandant un changement de propriétaire, afin de conserver l'anonymat.

Pour mieux comprendre les actions possibles et les entités auxquelles elles s'appliquent, voici un diagramme de séquence.

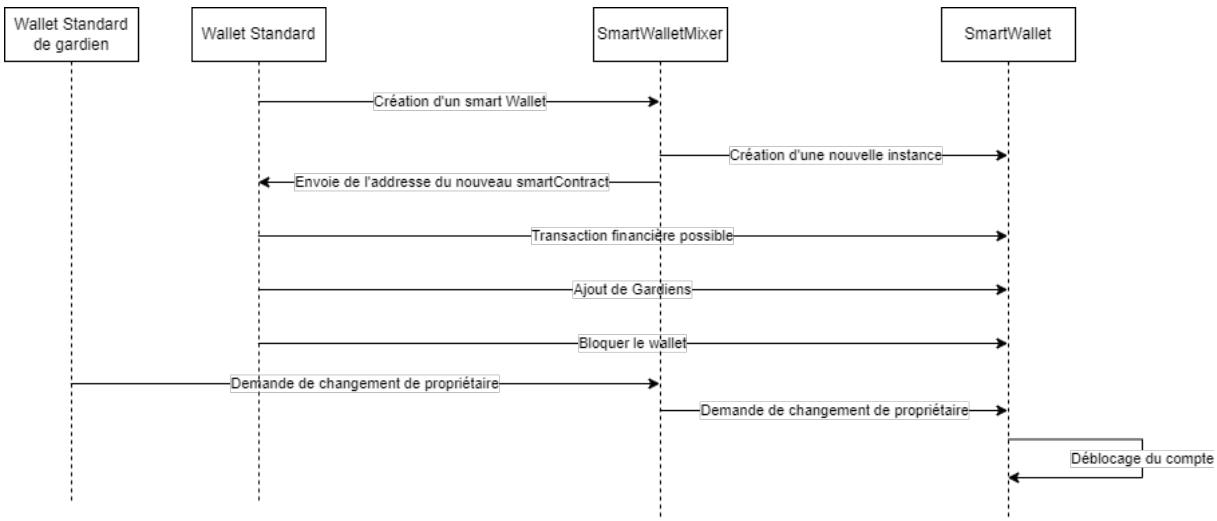

FIGURE 4.3 – *Diagramme de séquence fonctionnalité des smart contract*

Source de l'auteur

Le *smart wallet* contient les transactions de base pour recevoir de l'argent, envoyer de l'argent et consulter le montant disponible. Pour l'envoi d'argent, un modificateur vérifie que c'est bien le propriétaire qui effectue la requête et que le contrat n'est pas bloqué. Le propriétaire peut bloquer le contrat en attendant le changement de son adresse :

```
34      modifier onlyOwner() {
35          require(msg.sender == owner, "Only owner can call this function");
36          require(!isBlocked, "This smart wallet is blocked");
37          _;
38      }
```

Ensuite, il y a les transactions permettant de gérer les gardiens et le changement de propriétaire. Pour la gestion des gardiens, une variable contient une liste d'adresses qui sont les gardiens et la transaction reçoit une adresse pour ajouter le gardien à la liste. Lors de la création du *smart contract*, une variable par défaut demande que deux gardiens fassent la même requête de changement de propriétaire.

Pour gérer ces requêtes, la méthode s'inspire du *multisig*. Une liste de transactions est une structure comme suit :

```
15      struct Transaction {
16          address newOwner;
17          bool executed;
18      }
```

Un *mapping* (qui fonctionne comme un *hash map*) est utilisé avec l'identifiant de la transaction et l'adresse du gardien :

```
28      mapping (uint=>mapping (address=>bool)) isConfirmed;
```

Lorsqu'une demande est faite, il est vérifié si une demande a déjà été faite et le *hash map* `isConfirmed` est alimenté. Sinon, la transaction est créée et le *hash map* est alimenté. Enfin, il est vérifié si le nombre de demandes a atteint la limite initialisée lors de la création du *smart contract*. Si c'est le cas, l'adresse du propriétaire est changée et le contrat est débloqué :

```
77      function updateKey(address _guardian,address newKey) external onlyMixer{
78          require(newKey != address(0), "New key cannot be the zero address");
79          require(isValueInGuardiens(_guardian), "Only guardians can call this function");
80
81          Transaction memory t;
82          bool found = false;
83          uint transactionId=transactions.length;
84          for (uint i=0;i<transactions.length;i++){
85              if(transactions[i].newOwner == newKey){
86                  t=transactions[i];
87                  found = true;
88                  transactionId = i;
89                  break;
90              }
```

```
91          }
92          // If the transaction does not exist, create a new one
93          if (!found) {
94             t = Transaction({
95                   newOwner: newKey,
96                   executed: false
97             });
98
99             transactions.push(t);
100         }
101         if(isConfirmed[transactionId][_guardien]){
102            return;
103         }
104         isConfirmed[transactionId][_guardien] = true;
105         if(isTransactionConfirmed(transactionId)){
106            emit KeyUpdated(owner, newKey);
107            owner = newKey;
108            isBlocked = false;
109         }
110      }
```

Un modificateur `OnlyMixer` est appliqué à cette transaction. Cela permet aux gardiens de passer par le *mixer* lorsqu'ils souhaitent effectuer la transaction.

Dans le *mixer*, la demande est simplement transmise au *smart wallet* pour exécuter la requête, sans apparaître dans le scan de la blockchain. En effet, comme l'état de la blockchain n'est pas modifié, il ne s'agit pas d'une transaction mais uniquement d'un calcul :

```
18    function updateKey(SmartWallet walletAddress, address newKey) external{
19       require(newKey != address(0), "New key cannot be the zero address");
20       walletAddress.updateKey(msg.sender,newKey);
21    }
```

Ce *mixer* est également utilisé pour créer les *smart wallets*. En utilisant ce procédé, les frais de transaction lors de la création des *smart wallets* sont considérablement réduits. Pour créer le *smart contract*, il suffit d'utiliser `new`. Un événement est utilisé pour connaître l'adresse du nouveau *smart contract*, permettant ainsi à une application abonnée à cet événement de connaître l'adresse du *wallet* créé afin de pouvoir l'utiliser :

```
9     event SmartContractAddress(SmartWallet walletAddress);
10
11    function createWallet() public
12       returns (SmartWallet walletAddress)
13       {
14          walletAddress = new SmartWallet(msg.sender);
15          emit SmartContractAddress(walletAddress);
16       }
```

Le *smart contract* peut maintenant être compilé afin de se préparer au déploiement :

```
npx hardhat compile
```

4.6 Interaction avec le Smart Contract

Pour interagir avec le *smart contract*, nous utilisons la bibliothèque `@nomicfoundation/hardhat-ethers`, installée préalablement.

Il faut créer un script de déploiement appelé `deploy.js` et un script de gestion de *wallet* appelé `app.js`.

Pour accélérer les tests des applications, nous pouvons ajouter deux lignes dans le fichier `package.json` pour faciliter l'exécution des commandes de démarrage de l'application cliente et de déploiement du contrat.

```
"scripts": {
    "start": "npx hardhat run --network sepolia scripts/app.js" ,
    "deploy": "npx hardhat run --network sepolia scripts/deploy.js"
  }
```

Le fichier *package.json* devrait ressembler à ceci :

```
{
  "name": "firstcode",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "npx hardhat run --network sepolia scripts/app.js" ,
    "deploy": "npx hardhat run --network sepolia scripts/deploy.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@nomicfoundation/hardhat-ethers": "^3.0.6",
    "ethers": "^6.12.1",
    "hardhat": "^2.22.4"
  },
  "dependencies": {
    "dotenv": "^16.4.5"
  }
}
```

4.6.1 Déploiement

Pour déployer le *smart contract*, le script suivant permet de récupérer le contenu du contrat compilé et de le déployer sur une adresse. Ce script utilise les configurations de Hardhat avec le fichier de configuration et le fichier `.env`.

```
async function main() {
    const SmartWalletMixer = await ethers.getContractFactory("SmartWalletMixer");

    // Start deployment, returning a promise that resolves to a contract object
    const smart_Wallet = await SmartWalletMixer.deploy();
    console.log("Contract deployed to address:", smart_Wallet.address);
 }

 main()
   .then(() => process.exit(0))
   .catch(error => {
     console.error(error);
     process.exit(1);
   });
```

Il suffit ensuite d'exécuter la commande suivante afin de déployer le *smart contract* du *mixer*.

```
npm run deploy
```

4.6.2 Utilisation

Dans `app.js`, il y a la création du `main` qui va ensuite appeler les différents menus en fonction de la progression dans l'application.

Dans `menu.js`, les différents menus possibles pour l'utilisateur sont créés et appellent les méthodes de la classe `SmartContractClient`.

Pour voir comment effectuer les différentes transactions sur le *smart contract*, il faut regarder le fichier `interact.js`.

On crée l'accès aux contrats grâce aux lignes suivantes :

```
  constructor(privateKey){
    // Signer
    this.signer = new ethers.Wallet(privateKey, alchemyProvider);
    // Contract
    this.SmartWalletMixer = new ethers.Contract(MIXER_ADDRESS, mixer.abi, this.signer);
  }

  setContractAddress(contractAddress){
    this.SmartWalletContract = new ethers.Contract(contractAddress, contract.abi,
    ↪  this.signer);
    console.log("smart contract is defined");
  }
```

On utilise une transaction comme suit :

```
37      const tx = await this.SmartWalletMixer.createWallet();
38      const txReceipt = await tx.wait();
```

Et on s'abonne à un événement comme suivant :

```
33      this.SmartWalletMixer.on('SmartContractAddress',(walletAddress)=>{
34        console.log("The address from event is: " + walletAddress);
35        this.setContractAddress(walletAddress);
36      });
```

5 | Proof of concept

5.1 Utilisation du client

Lors du lancement de l'application, un menu propose de choisir un compte. Ce menu pourrait être amélioré en permettant l'ajout de comptes supplémentaires, enregistrés dans une base de données sécurisée. Pour ce travail, une version simple a été proposée, avec quatre comptes existants dont les clés privées sont stockées dans le fichier `.env`.

Une fois le compte choisi, il est possible de :

— Créer un *smart wallet* (requête effectuée sur le *mixer*).
— Utiliser un *smart wallet* déjà créé en fournissant une adresse.

Ensuite, un menu permet d'interagir avec le *wallet*. Les options disponibles sont :

— Consulter le solde.
— Retirer un montant du *smart wallet*.
— Ajouter des fonds depuis notre compte vers le *smart wallet*.

La section de récupération permet de :

— Ajouter des gardiens en entrant leur adresse.
— Bloquer le compte en cas de vol de la clé.
— Demander un changement de propriétaire en fournissant une nouvelle adresse (requête effectuée sur le *mixer* afin de garantir l'anonymat des gardiens).

Dans cette version du *smart wallet*, seuls deux gardiens sont nécessaires pour effectuer le changement de propriétaire.

5.2 Tests effectués

Afin de détecter les éventuelles failles du projet, une série de tests rigoureux ont été effectués. Ces tests, dont les résultats sont présentés ci-dessous, ont tous été menés avec succès, démontrant le fonctionnement du système. De plus, ces tests ont permis de faire des constatations intéressantes sur le fonctionnement et la sécurité du projet. Toutes les preuves des tests sont en annexes de ce document.

Numéro	Quoi	Compte	Résultat attendu
1	Création d'un smart wallet	Compte 1	Succès
2	Ajout d'argent sur le smart wallet	Compte 1	Succès
3	Retrait d'argent sur le smart wallet créé	Compte 1	Succès
4	Ajout des deux gardiens	Compte 1	Succès
5	Retrait d'argent sur le smart wallet	Compte 2	Erreur
6	Ajout du compte 2 en tant que gardien	Compte 2	Erreur
7	Bloquer le smart wallet	Compte 2	Erreur
8	Bloquer le smart wallet	Compte 1	Succès
9	Retrait d'argent sur le smart wallet	Compte 1	Erreur
10	Demande de changement de propriétaire pour le compte 2	Compte 1	Erreur
11	Demande de changement de propriétaire pour le compte 2	Compte 2	Erreur
12	Demande de changement de propriétaire pour le compte 2	Gardien 1	Succès
13	Demande de changement de propriétaire pour le compte 2	Gardien 2	Validation, compte changé
14	Retrait d'argent sur le smart wallet	Compte 2	Succès
15	Retrait d'argent sur le smart wallet	Compte 1	Erreur

TABLE 5.1 – *Liste des tests*

Grâce à ce protocole, nous avons vérifié qu'un compte autre que le compte propriétaire ne peut ni récupérer de l'argent ni ajouter un gardien. De plus, seuls les gardiens peuvent faire des demandes de changement de propriétaire. Une fois le nombre requis de demandes atteint, le changement est effectué et le compte est débloqué.

Nous avons également démontré que le changement de gardien se fait de manière invisible lors de la lecture de la blockchain, garantissant ainsi l'anonymat des opérations. Dans les figure qui vont suivre on pourra constater que la transaction arrive sur le *mixer* mais qu'aucune transaction interne ou externe ne sera visible sur le *smart wallet*.

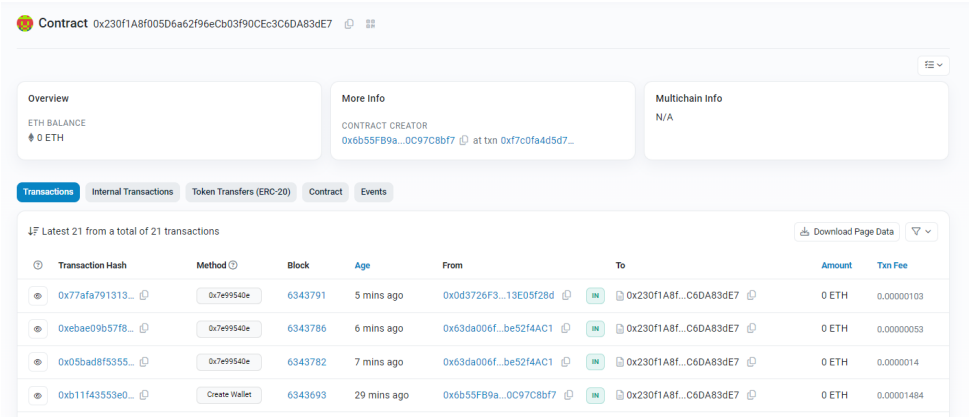

FIGURE 5.1 – *Transactions externes des gardiens depuis le mixer*

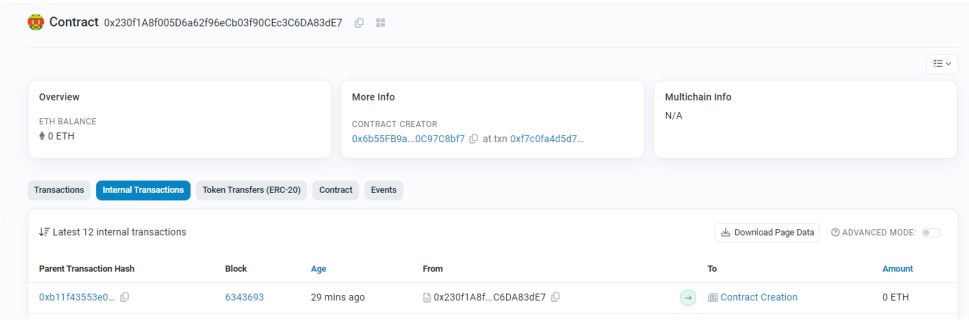

FIGURE 5.2 – *Transactions internes des gardiens depuis le mixer*

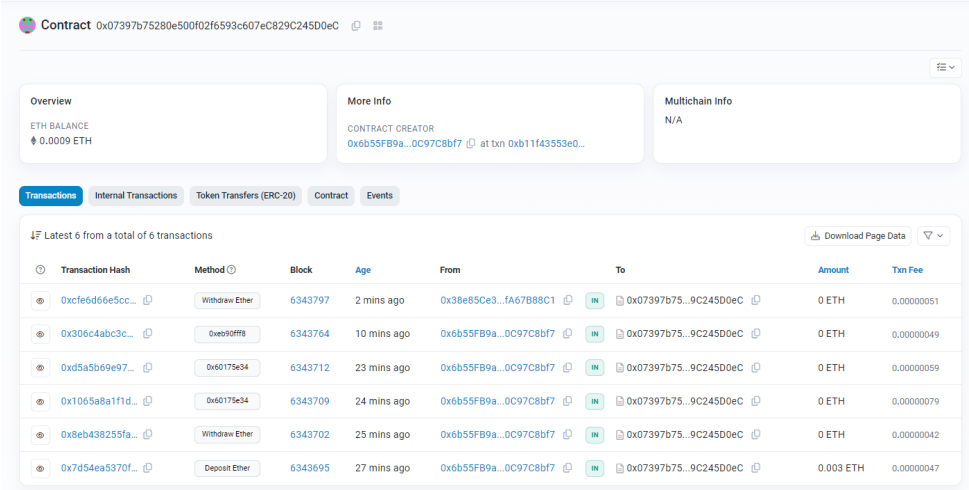

FIGURE 5.3 – *Transactions externes des gardiens depuis le smart wallet*

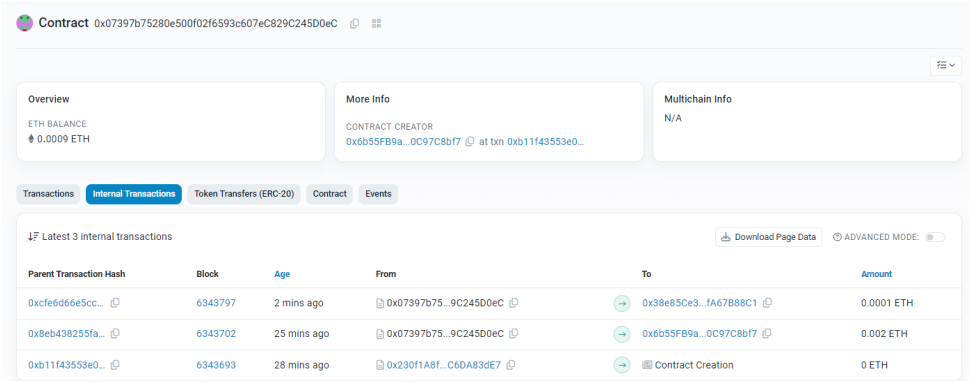

FIGURE 5.4 – *Transactions internes des gardiens depuis le smart wallet*

Enfin, il est important de noter que le coût du déploiement d'un *smart contract* est d'environ 0.000014 ETH, alors qu'auparavant, déployer un *smart wallet* sans passer par un *smart contract* coûtait environ 0.03 ETH. Sur la figure 5.5, nous présentons deux extraits de création de *smart wallets* effectués dans le cadre des tests préalables à la mise en place du *mixer*.

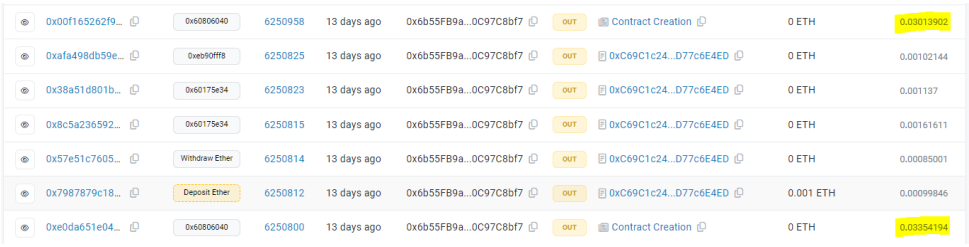

FIGURE 5.5 – *Coût de création du smart wallet sans mixer*

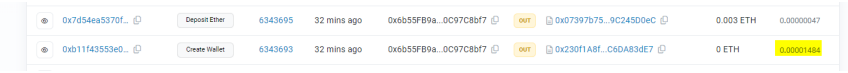

FIGURE 5.6 – *Coût de création du smart wallet avec mixer*

6 | Améliorations

6.1 Gestion des mises à jour des *smart contracts*

Il est important de noter que les *smart contracts*, une fois déployés, sont immuables. Par conséquent, comment peut-on mettre à jour le code d'un *smart contract* ? Pour ce faire, plusieurs approches existent (SOLIDITY ACADEMY, 2023) :

— **Utilisation d'un proxy** : Cette méthode permet de déléguer les appels à une autre entité, ce qui signifie que seule l'adresse de l'entité appelée par le proxy doit être modifiée pour mettre à jour le *smart contract*.

— **Utilisation d'un stockage éternel** : Cette approche consiste à séparer la logique des données, ce qui permet de mettre à jour uniquement le *smart contract* contenant la logique, tout en conservant les données inchangées.

— **Utilisation de bibliothèques** : Les bibliothèques peuvent être mises à jour indépendamment du *smart contract*. En liant un *smart contract* à une bibliothèque externe, il est possible de mettre à jour uniquement la librairie, ce qui entraînera un changement de comportement du *smart contract*.

Lors de la mise en place de solutions pour la mise à jour des *smart contracts*, il est important de considérer que de nombreux contrats pourraient être déployés. Il peut donc être utile de conserver une liste des adresses de déploiement à l'aide du *mixer*. Cependant, il est également crucial de sécuriser cette liste afin d'empêcher un accès non autorisé à ces informations.

Aucune solution de mise à jour n'a été testée durant ce travail, car cela sort du cadre du travail. Toutefois, il s'agit d'un point crucial à considérer avant de réaliser des tests plus poussés en production.

6.2 Rembourser les gardiens

Un des axes d'amélioration pertinent à explorer est le remboursement des gardiens ayant initié une demande de changement de propriétaire qui s'est concrétisée avec succès. Actuellement, ces gardiens peuvent assumer des frais pour garantir la validité et la sécurité des transactions. Mettre en place un mécanisme de remboursement permettrait de récompenser ces efforts, incitant ainsi les gardiens à continuer de jouer leur rôle de manière proactive et efficace. Cela pourrait se traduire par une augmentation de la fiabilité et de la robustesse du système dans son ensemble.

6.3 ERC-4337

Afin d'améliorer ce projet, il serait judicieux de mettre en place le protocole ERC-4337.

Ce protocole propose une abstraction des comptes qui pourrait considérablement accroître l'efficacité et la flexibilité du système. Il simplifie l'expérience utilisateur en introduisant le concept de *UserOperations*. Ces opérations permettent de spécifier les actions qu'un utilisateur souhaite effectuer ; si le compte de l'utilisateur n'existe pas encore, il sera créé lors de la première requête (ROSS, 2023 ; DUPUIS, 2023 ; WERKHEISER, 2024 ; BUTERIN, 2022 ; VBUTERIN, 2021 ; ERC-4337 TEAM, 2023).

L'ERC-4337 contribue à la réduction des frais de transaction en regroupant les demandes des utilisateurs. De plus, il permet l'introduction de nouveaux acteurs, appelés *Paymasters*, qui peuvent couvrir les frais de transaction pour les utilisateurs. Ce protocole offre ainsi une plus grande flexibilité en permettant aux utilisateurs de déléguer le paiement des frais, ce qui améliore l'accessibilité et l'utilisation des applications décentralisées (ROSS, 2023 ; DUPUIS, 2023 ; WERKHEISER, 2024 ; BUTERIN, 2022 ; VBUTERIN, 2021 ; ERC-4337 TEAM, 2023).

Pour mettre en œuvre ces améliorations, il est nécessaire de suivre les directives et recommandations détaillées dans la documentation officielle de l'ERC-4337 (ROSS, 2023 ; DUPUIS, 2023 ; WERKHEISER, 2024 ; BUTERIN, 2022 ; VBUTERIN, 2021 ; ERC-4337 TEAM, 2023).

7 | Conclusion

Dans ce dernier chapitre, nous présentons une synthèse du travail effectué depuis le 20 mai 2024

7.1 Synthèse

Dans ce travail, nous avons exploré deux approches pour la récupération de l'accès à un portefeuille numérique via des gardiens. La première approche utilise la *MPC* (Multi-Party Computation). Bien que cette méthode soit réputée inviolable, elle ne répond pas de manière satisfaisante aux besoins en cas de vol. Nous nous sommes donc principalement concentrés sur la création d'un portefeuille via un *smart contract*.

Le choix de la blockchain s'est porté sur *Ethereum* en raison de la simplicité d'accès à sa documentation, étant donné qu'il s'agissait d'une découverte de la technologie. D'autres blockchains, telles que *Secret Network*, ont toutefois été considérées pour la création de *smart contracts* sécurisés.

Au cours de la réalisation, afin de préserver l'anonymat des gardiens, les demandes de changement de propriétaire sont traitées par l'intermédiaire d'un *smart contract* unique. Ce *smart contract* unique redirige ensuite la demande vers le *smart contract* approprié. Grâce à cette approche, il a été possible de réduire les coûts de création du *smart wallet* tout en conservant l'anonymat des gardiens.

7.2 Limites du travail

Dans cette étude, les gardiens sont responsables du paiement des frais de transaction sans possibilité de remboursement. Une solution optimale consisterait à mettre en place un mécanisme de remboursement des frais de transaction une fois la transaction validée.

En outre, ce travail s'est concentré sur la compréhension du fonctionnement des *smart contracts* et sur leur création, sans examiner toutes les solutions existantes. Il existe en effet de nombreuses solutions de portefeuilles, que ce soit des *wallets MPC* ou des *smart wallets*. Cependant, pour faire un choix éclairé, il est crucial de comprendre comment créer ces solutions à partir de zéro.

7.3 Perspectives de recherches

Ce projet évoque également les *SoulBound tokens*. Il pourrait être intéressant d'intégrer la gestion de ces *tokens* au sein du *smart contract*, étant donné que ce travail s'inscrit dans le cadre de la gestion des données électroniques.

8 | Utilisation de l'IA

Durant ce travail de bachelor, ChatGPT a été utilisé à plusieurs étapes du projet. Au début, ChatGPT a servi pour orienter mes recherches en fournissant des informations pertinentes et en répondant à des questions spécifiques concernant les technologies et les concepts liés aux *smart contracts*, aux *wallets* décentralisés, et à la blockchain en général. Cette assistance a permis de mieux définir les objectifs du projet et de comprendre les différents aspects techniques à aborder, tout en veillant à la pertinence de ses propos.

Par la suite, ChatGPT a été un outil pour la rédaction du rapport. Il a aidé à structurer les idées, à clarifier certains concepts techniques et à reformuler des paragraphes pour améliorer la clarté et la cohérence du texte. De plus, il a facilité l'intégration de la mise en forme des éléments en LaTeX, assurant ainsi une présentation professionnelle du document final.

Ainsi, l'utilisation de ChatGPT a non seulement amélioré l'efficacité de mes recherches, mais a également rehaussé la qualité de la rédaction du rapport, en apportant un soutien continu tout au long du projet.

Bibliographie

ALCHEMY. (2023a). *How do i distinguish between a contract address and a wallet address ?* [Alchemy docs]. Récupérée juillet 10, 2024, à partir de https://docs.alchemy.com/reference/contract-address-vs-wallet-address

ALCHEMY. (2023b). *How to send value from within a smart contract using solidity* [Alchemy docs]. Récupérée juin 28, 2024, à partir de https://docs.alchemy.com/docs/how-to-send-value-from-within-a-smart-contract-using-solidity

ALLISON, I. (2023). *L'empreinte énergétique d'Ethereum (ETH) avant la fusion était équivalente à l'utilisation annuelle de la Suisse*. Récupérée août 11, 2024, à partir de https://www.coindesk.com/fr/tech/2023/04/26/ethereums-lifetime-energy-use-before-the-merge-equaled-switzerlands-for-a-year/

ARTHUR, V. (2024). *Polygon vs ethereum (2024) : which offers better scalability ?* [CoinWire]. Récupérée juin 28, 2024, à partir de https://coinwire.com/polygon-vs-ethereum/

BLOCKGAMES. (2023). *The use cases of soulbound tokens in life and games* [Medium]. Récupérée août 9, 2024, à partir de https://medium.com/@BlockGames.com/the-uses-cases-of-soulbound-tokens-in-life-and-games-de97e4406dcb

BUTERIN, V. (2022). *ERC 4337 : account abstraction without ethereum protocol changes* [Medium]. Récupérée juillet 19, 2024, à partir de https://medium.com/infinitism/erc-4337-account-abstraction-without-ethereum-protocol-changes-d75c9d94dc4a

CHAINSPECT. (2024a). *Ethereum vs polygon* [Chainspect]. Récupérée août 9, 2024, à partir de https://chainspect.app/compare/ethereum-vs-polygon

CHAINSPECT. (2024b). *Ethereum vs solana* [Chainspect]. Récupérée août 9, 2024, à partir de https://chainspect.app/compare/ethereum-vs-solana

COINBASE. (2024a). *Coinbase - Acheter et vendre des bitcoins, des ethereums et bien plus encore, en toute confiance* [Coinbase]. Récupérée août 15, 2024, à partir de https://www.coinbase.com/fr

COINBASE. (2024b). *Qu'est-ce que polygon (MATIC) ?* [Coinbase]. Récupérée août 4, 2024, à partir de https://www.coinbase.com/fr-fr/learn/crypto-basics/what-is-polygon

COINBASE. (2024c). *Why smart wallet ?* [SmartWallet]. Récupérée août 15, 2024, à partir de https://www.smartwallet.dev/why

de BALASY, C. (2023). *Maîtriser – Les Smart contracts (Avancé)* [Coinhouse]. Récupérée juillet 15, 2024, à partir de https://www.coinhouse.com/fr/academie/approfondir-smart-contract-avance/

DEBELLOIR, M. (2021). *Une transaction sur Solana (SOL) coûterait moins d'énergie que deux recherches Google* [Cryptoast]. Récupérée août 9, 2024, à partir de https://cryptoast.fr/transaction-solana-cout-energie-vs-google/

DFNS. (2024). *Dfns — the most powerful web3 wallet infrastructure* [Dfns]. Récupérée août 11, 2024, à partir de https://www.dfns.co/

DUPUIS, L. (2023). *Qu'est-ce que l'ERC-4337 synonyme d'Account Abstraction sur Ethereum ?* [Coin Academy]. Récupérée juillet 15, 2024, à partir de https://coinacademy.fr/academie/erc-4337/

ERC-4337 TEAM. (2023). *ERC-4337* [ERC-4337]. Récupérée juillet 19, 2024, à partir de https://www.erc4337.io/

ETHEREUM. (2024a). *Accueil* [Ethereum]. Récupérée août 11, 2024, à partir de https://ethereum.org/fr/

ETHEREUM. (2024b). *Centre d'apprentissage* [Ethereum]. Récupérée août 11, 2024, à partir de https://ethereum.org/fr/learn/

ETHEREUM. (2024c). *Contrats intelligents* [Ethereum]. Récupérée juin 28, 2024, à partir de https://ethereum.org/fr/smart-contracts/

FIREBLOCKS. (2024). *What Is MPC (Multi-Party Computation) ?* Récupérée août 11, 2024, à partir de https://www.fireblocks.com/what-is-mpc/

FLAYDEM. (2023a). *Qu'est-ce que la congestion d'un réseau blockchain en crypto ?* [Coin Academy]. Récupérée août 9, 2024, à partir de https://coinacademy.fr/academie/congestion-reseau-blockchain/

FLAYDEM. (2023b). *Qu'est-ce qu'un portefeuille Social Recovery en crypto ?* [Coin Academy]. Récupérée août 11, 2024, à partir de https://coinacademy.fr/academie/portefeuille-social-recovery-crypto/

HAFZA, Y. (2024). *Beginner's Guide to Smart Contracts : Understanding the Building Blocks of Blockchain* [Rise In]. Récupérée juin 28, 2024, à partir de https://www.risein.com/blog/beginners-guide-to-smart-contracts

HALPERN, E. (2021a). *Deploy Your First Smart Contract* [Web3 University]. Récupérée juin 28, 2024, à partir de https://www.web3.university/tracks/create-a-smart-contract/deploy-your-first-smart-contract

HALPERN, E. (2021b). *Interact With Your Smart Contract* [Web3 University]. Récupérée août 11, 2024, à partir de https://www.web3.university/tracks/create-a-smart-contract/interact-with-your-smart-contract

HELBIG, J. M. (2019). *De la blockchain à la crypto-investisseur*. KLHE finance.

IBM. (2021). *What are smart contracts on blockchain ?* [IBM]. Récupérée juin 28, 2024, à partir de https://www.ibm.com/topics/smart-contracts

LEARNCRYPTO. (2024, février 7). *What are soulbound tokens ?* Récupérée août 11, 2024, à partir de https://learncrypto.com/knowledge-base/basics/what-are-soulbound-tokens-sbt-explained

LEDGER. (2023). *Qu'est-ce qu'un token soulbound ?* [Ledger]. Récupérée juillet 9, 2024, à partir de https://www.ledger.com/fr/academy/sujets/blockchain/quest-ce-quun-token-soulboundnbsp

LEROUX, A. (2023). *C'est quoi un NFT ? Définition d'un Non Fungible Token* [Coin Academy]. Récupérée août 9, 2024, à partir de https://coinacademy.fr/academie/definition-nft/

MIGUEL, A. (2024). *Safe wallet review 2024 : pros, cons, & features* [Milk road]. Récupérée août 14, 2024, à partir de https://milkroad.com/reviews/safe-wallet-review/

PICARDO, E. (2023). *What is solana (SOL) and how does SOL crypto work ?* [Investopedia]. Récupérée août 4, 2024, à partir de https://www.investopedia.com/solana-5210472

POLYGON. (2024). *Polygon Knowledge Layer* [Polygon docs]. Récupérée juin 28, 2024, à partir de https://docs.polygon.technology/

PORCELLI, A. (2024). *Coinbase : est arrivé le Smart Wallet* [The Cryptonomist]. Récupérée août 15, 2024, à partir de https://fr.cryptonomist.ch/2024/06/05/coinbase-est-arrive-le-smart-wallet/

RABBITCOIN. (2024, juin 6). *Coinbase introduit smart wallet, son nouveau portefeuille pour promouvoir l'adoption de la DeFi* [Cryptoast]. Récupérée août 15, 2024, à partir de https://cryptoast.fr/coinbase-introduit-smart-wallet-nouveau-portefeuille-promouvoir-adoption-defi/

ROSS, L. (2023). *What is account abstraction (ERC-4337) ?* [Alchemy]. Récupérée juillet 15, 2024, à partir de https://www.alchemy.com/overviews/what-is-account-abstraction

SAFE. (2023). *Say hello to safe{RecoveryHub} - account recovery, your way* [Mirror]. Récupérée août 14, 2024, à partir de https://safe.mirror.xyz/WxKSxD9J1bRI-SDOuDvAAIezwVrvWWkpuwuzcLDPS

SAFE ECOSYSTEM FOUNDATION. (2024, août 9). *Safe docs* [Safe docs]. Récupérée août 14, 2024, à partir de https://docs.safe.global/

SANKRIT K. (2024). *What are smart wallets and how to use coinbase's smart wallet* [CoinGecko]. Récupérée août 11, 2024, à partir de https://www.coingecko.com/learn/what-are-smart-wallets-crypto

SECRET NETWORK. (2024). *Secret network introduction* [Secret network]. Récupérée juin 28, 2024, à partir de https://docs.scrt.network/secret-network-documentation/

SERAH, O. (2023). *Mastering addresses in ethereum* [Medium]. Récupérée juillet 6, 2024, à partir de https://medium.com/@ajaotosinserah/mastering-addresses-in-ethereum-5411ba6c3b0f

SOLANA. (2024). *Programs* [Solana]. Récupérée juin 28, 2024, à partir de https://solana.com/docs/core/programs

SOLIDITY ACADEMY. (2023). *Managing and upgrading smart contracts in solidity* [Medium]. Récupérée août 2, 2024, à partir de https://medium.com/coinmonks/managing-and-upgrading-smart-contracts-in-solidity-d0b7f8ae663e

SOUL WALLET. (2024). *Smart contract wallet for Ethereum* [Soul Wallet]. Récupérée août 14, 2024, à partir de https://www.soulwallet.io/

STAN. (2022). *Secret Network* [Tokens Invaders]. Récupérée août 9, 2024, à partir de https://tokensinvaders.com/crypto/secret-network/

STRUCK. (2023, avril 20). *Why we invested : soul wallet* [Struck capital]. Récupérée août 14, 2024, à partir de https://struckcapital.com/why-we-invested-soul-wallet/

TUSHARMAHAJAN. (2023). *Implementing multi-sig wallets in smart contracts* [Medium]. Récupérée juillet 6, 2024, à partir de https://medium.com/simform-engineering/implementing-multi-sig-wallets-in-smart-contracts-27c71a58d071

VBUTERIN. (2021). EIPs/EIPS/eip-4337.md at 3fd65b1a782912bfc18cb975c62c55f733c7c96e · ethereum/EIPs. Récupérée juillet 19, 2024, à partir de https://github.com/ethereum/EIPs/blob/3fd65b1a782912bfc18cb975c62c55f733c7c96e/EIPS/eip-4337.md

WARDZALA, C. (2023). *Qu'est-ce qu'un NFT dynamique et quels sont ses usages ?* [Cryptoast]. Récupérée août 10, 2024, à partir de https://cryptoast.fr/qu-est-ce-qu-un-nft-dynamique/

WERKHEISER, B. (2024). *How do ERC-4337 smart contract wallets work ?* [Alchemy]. Récupérée juin 28, 2024, à partir de https://www.alchemy.com/overviews/how-do-smart-contract-wallets-work

WIKIPÉDIA. (2024a). NFT [Page Version ID : 217330405]. In *Wikipédia*. Récupérée août 9, 2024, à partir de https://fr.wikipedia.org/w/index.php?title=NFT&oldid=217330405

WIKIPÉDIA. (2024b, juillet 29). Solidity [Page Version ID : 217202185]. In *Wikipédia*. Récupérée août 11, 2024, à partir de https://fr.wikipedia.org/w/index.php?title=Solidity&oldid=217202185

WIKIPÉDIA. (2024c, août 7). Rust (langage) [Page Version ID : 217478590]. In *Wikipédia*. Récupérée août 11, 2024, à partir de https://fr.wikipedia.org/w/index.php?title=Rust_(langage)&oldid=217478590

ZENGO. (2024). *Zengo wallet : secure by default* [Zengo]. Récupérée août 11, 2024, à partir de https://zengo.com/

Informations sur ce travail

Informations de contact

Auteur : Céline Vialard

HES-SO Valais-Wallis

E-mail : *celine.vialard@students.hevs.ch*

Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de bachelor, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail.

Lieu, date : Sierre, 16.08.2024 _____

Signature : Céline Vialard _____

9 | Annexe - Test

```
Choisir votre compte?
            a1. Account_1
            a2. Account_2
            g1. Guardien_1
            g2. Guardien_2
            e. Quitter

a1

Que voulez vous faire?
            1. Créer mon wallet
            2. Entrer l'addresse de mon wallet
            e. Quitter

1
L'addresse du nouveau portefeuill est : 0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

2
Le montant est : 0 ETH

Entrez un montant en ETH
0.003
Le nouveau montant est :  0.003 ETH
```

FIGURE 9.1 – *Tests 1 et 2*

```
Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

1
Le montant est : 0.003 ETH

Entrez un montant en ETH
0.002
Le nouveau montant est : 0.001 ETH

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

3

Entrez une addresse
0x63da006fa6770a51B0c660df0775760be52f4AC1
Guardien ajouté : 0x63da006fa6770a51B0c660df0775760be52f4AC1

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

3

Entrez une addresse
0x0d3726F382f7A17af9326d1E61A91aD13E05f28d
Guardien ajouté : 0x0d3726F382f7A17af9326d1E61A91aD13E05f28d
```

FIGURE 9.2 – *Tests 3 et 4*

```
Choisir votre compte?
            a1. Account_1
            a2. Account_2
            g1. Guardien_1
            g2. Guardien_2
            e. Quitter

a2

Que voulez vous faire?
            1. Créer mon wallet
            2. Entrer l'addresse de mon wallet
            e. Quitter

2

Entrez une addresse

0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

1
Le montant est : 0.001 ETH

Entrez un montant en ETH
0.0001
execution reverted: "Only owner can call this function"
```

FIGURE 9.3 – *Test 5*

```
Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

3

Entrez une addresse
0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
execution reverted: "Only owner can call this function"

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

4
execution reverted: "Only owner can call this function"
```

FIGURE 9.4 – *Tests 6 et 7*

```
Choisir votre compte?
            a1. Account_1
            a2. Account_2
            g1. Guardien_1
            g2. Guardien_2
            e. Quitter

a1

Que voulez vous faire?
            1. Créer mon wallet
            2. Entrer l'addresse de mon wallet
            e. Quitter

2

Entrez une addresse

0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

4
Portefeuille bloqué
```

FIGURE 9.5 – *Test 8*

```
Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

1
Le montant est : 0.001 ETH

Entrez un montant en ETH
0.0001
execution reverted: "This smart wallet is blocked"

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

5

Entrez une addresse
0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
execution reverted: "Only guardiens can call this function"
```

FIGURE 9.6 – *Tests 9 et 10*

```
Choisir votre compte?
            a1. Account_1
            a2. Account_2
            g1. Guardien_1
            g2. Guardien_2
            e. Quitter

a2

Que voulez vous faire?
            1. Créer mon wallet
            2. Entrer l'addresse de mon wallet
            e. Quitter

2

Entrez une addresse

0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

5

Entrez une addresse
0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
execution reverted: "Only guardiens can call this function"
```

FIGURE 9.7 – *Test 11*

```
Choisir votre compte?
            a1. Account_1
            a2. Account_2
            g1. Guardien_1
            g2. Guardien_2
            e. Quitter

g1

Que voulez vous faire?
            1. Créer mon wallet
            2. Entrer l'addresse de mon wallet
            e. Quitter

2

Entrez une addresse

0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

5

Entrez une addresse
0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
demande de changement de propriétaire fait :0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
```

FIGURE 9.8 – *Test 12*

```
Choisir votre compte?
          a1. Account_1
          a2. Account_2
          g1. Guardien_1
          g2. Guardien_2
          e. Quitter

g2

Que voulez vous faire?
          1. Créer mon wallet
          2. Entrer l'addresse de mon wallet
          e. Quitter

2

Entrez une addresse

0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
       0. Connaitre la balance
       1. Transfèrer de l'argent
       2. Ajouter de l'argent
       3. Ajouter un gardien
       4. Bloquer le compte
       5. Changer le propriétaire
       e. Quitter

5

Entrez une addresse
0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
The address previous owner was 0x6b55FB9aa67c9C0DdC2cA828b9B554E0C97C8bf7 now it is: 0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
demande de changement de propriétaire fait :0x38e85Ce34069c54B10Dd678691621E7fA67B88C1
```

FIGURE 9.9 – *Test 13*

```
Choisir votre compte?
            a1. Account_1
            a2. Account_2
            g1. Guardien_1
            g2. Guardien_2
            e. Quitter

a2

Que voulez vous faire?
            1. Créer mon wallet
            2. Entrer l'addresse de mon wallet
            e. Quitter

2

Entrez une addresse

0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

1
Le montant est : 0.001 ETH

Entrez un montant en ETH
0.0001
Le nouveau montant est : 0.0009 ETH
```

FIGURE 9.10 – *Test 14*

```
Choisir votre compte?
            a1. Account_1
            a2. Account_2
            g1. Guardien_1
            g2. Guardien_2
            e. Quitter

a1

Que voulez vous faire?
            1. Créer mon wallet
            2. Entrer l'addresse de mon wallet
            e. Quitter

2

Entrez une addresse

0x07397b75280e500f02f6593c607eC829C245D0eC
smart contract is defined

Que voulez vous faire?
        0. Connaitre la balance
        1. Transfèrer de l'argent
        2. Ajouter de l'argent
        3. Ajouter un gardien
        4. Bloquer le compte
        5. Changer le propriétaire
        e. Quitter

1
Le montant est : 0.0009 ETH

Entrez un montant en ETH
0.0009
execution reverted: "Only owner can call this function"
```

FIGURE 9.11 – *Test 15*