



RÉPUBLIQUE DU BÉNIN
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ D'ABOMEY-CALAVI

INSTITUT DE FORMATION ET DE
RECHERCHE EN INFORMATIQUE

BP 526 Cotonou Tel : +229 21 14 19 88
<http://www.ifri-uac.net> Courriel : contact@ifri.uac.bj



MÉMOIRE

pour l'obtention du

Diplôme de Master en Informatique

Option : Sécurité Informatique

Présenté par :

Farold Hufranc ADOUKONOU SAGBADJA

Prototype d'un outil d'anonymisation des transactions financières sur la blockchain Ethereum.

Sous la supervision :

Prof. Eugène EZIN

Enseignant-Chercheur à l'université d'Abomey-Calavi

Membres du jury :

...

Année Académique : 2023-2024

Sommaire

Dédicace	ii
Remerciements	iii
Résumé	iv
Abstract	v
Liste des figures	vi
Liste des tableaux	vii
Sigles et Abréviations	viii
Glossaire	ix
Introduction	1
1 Etat de l’art	3
2 Matériels et Méthodes	23
3 Modélisation et Développement	27
4 Résultats et Discussion	48
Conclusion	56
Bibliographie	57
Webographie	58
Table des matières	60
Annexes	64

Dédicace

À mes géniteurs, ADOUKONOU SAGBADJA Hubert mon père et HOUEDOKOHO Francine ma mère pour leurs soutiens indéfectibles. Que Dieu vous accorde longue vie et une santé inoxydable afin de jouir du fruit de vos efforts!

Remerciements

Nous tenons à exprimer nos profonds remerciements et nos profondes reconnaissances envers toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce mémoire. Nos remerciements s'adressent en particulier à :

- Professeur Eugène C. EZIN, Enseignant-Chercheur à l'université d'Abomey-Calavi et Directeur de l'Institut de Formation et de Recherche en Informatique (IFRI), mon maître de mémoire pour avoir accepté de superviser ce travail et tout le temps consenti pour parfaire le présent document ;
- Dr. Gaston EDAH, Directeur Adjoint de l'IFRI pour tous les efforts consentis pour nous offrir une formation de qualité ;
- Tous les enseignants de l'IFRI, pour l'enseignement et les savoirs partagés tout au long de notre formation ;
- Nos parents, amis, et proches qui nous ont soutenus tout le long de ce travail ;
- Tous les camarades de notre promotion et à nos aînés.

Résumé

L'émergence des cryptomonnaies et de la technologie blockchain a transformé le fonctionnement des transactions financières en apportant efficacité et sécurité grâce à la décentralisation. Néanmoins, les blockchains publiques, comme Ethereum, posent des défis en matière de confidentialité, car toutes les transactions sont ouvertes et facilement accessibles au public. Cette étude présente la création d'un outil prototype permettant d'anonymiser les transactions sur la blockchain Ethereum en intégrant les preuves à divulgation nulle de connaissance (zk-SNARKS) et un système zk-KYC afin de maintenir un équilibre entre anonymat et normes réglementaires.

Les résultats démontrent qu'il est possible d'améliorer la confidentialité des transactions sur Ethereum tout en respectant les règles de KYC/AML de manière efficace, offrant ainsi un moyen de protéger l'anonymat des utilisateurs sans compromettre les obligations légales. **Mots clés** : Blockchain Ethereum, Anonymisation, zk-SNARKs, zk-KYC, Transactions financières, Confidentialité, Conformité KYC/AML

Abstract

The emergence of cryptocurrencies and blockchain technology has transformed financial transactions by enhancing efficiency and security through decentralization. However, public blockchains like Ethereum present privacy challenges, as all transactions are transparent and easily accessible. This study explores the development of a prototype tool designed to anonymize transactions on the Ethereum blockchain by leveraging Zero-Knowledge Proofs (zk-SNARKs) and a zk-KYC system to balance privacy with regulatory compliance. The findings demonstrate that it is possible to enhance transaction privacy on Ethereum while effectively adhering to KYC and AML regulations, providing a solution that protects user anonymity without compromising legal requirements. **Key**

words: Ethereum Blockchain, Anonymization, zk-SNARKs, zk-KYC, Financial Transactions, Privacy, KYC/AML Compliance

Liste des figures

1.1	Structure d'une blockchain	5
1.2	Ethereum Virtual Machine	8
1.3	Processus utilisateur à l'adresse du portefeuille	9
1.4	Répartition des fonds envoyés sur les mixers en 2022	14
1.5	Fonctionnement des ZKPs	17
1.6	Structure d'un arbre de Merkle	19
1.7	Processus de mélange du mixeur	21
2.1	Architecture globale du système	25
3.1	Flux de fonctionnement du système	28
3.2	Diagramme de composant du système	29
3.3	Processus de vérification d'identité et de whitelist	30
3.4	Processus de dépôt des fonds sur le mixer	31
3.5	Processus de retrait des fonds du mixer	32
3.6	Déploiement des smart contracts du Mixer	42
4.1	Connexion avec Metamask à ZK KYC	48
4.2	Vérification d'identité et génération de preuve	49
4.3	Whitelist d'adresse avec un hash valide	49
4.4	Connexion avec Metamask à Phantom ETH	50
4.5	Formulaire de dépôt sur Phantom ETH	51
4.6	Note secrète après dépôt sur Phantom ETH	51
4.7	Erreur de dépôt avec une adresse non whitelisted	52
4.8	Soumission de la demande de retrait sur Phantom ETH	52
4.9	Fonds retirés avec succès sur Phantom ETH	53

Liste des tableaux

4.1 Comparaison entre Phantom ETH et Tornado Cash 54

Sigles et Abréviations

AML : Anti-Money Laundering [1](#), *see* [Blanchiment d'argent](#)

KYC : Know Your Customer [1](#)

zk-SNARKs : Zero-Knowledge Succinct Non-Interactive Argument of Knowledge [15](#)

Glossaire

- Circom :** Circom est un langage qui permet de définir des circuits arithmétiques qui peuvent être utilisés pour générer des preuves à connaissance nulle. [21](#)
- DeFi :** DeFi, (ou « decentralized finance » (finance décentralisée), est un terme générique désignant les services financiers sur les blockchains publiques, principalement l'Ethereum. La DeFi offre des services bancaires classiques (percevoir des intérêts, emprunter, prêter, acheter des assurances, négocier des produits dérivés et des actifs, et bien plus encore) mais avec plus de rapidité et sans intermédiaire. [8](#)
- keccak256 :** Keccak256 est un membre de la famille de fonctions de hachage SHA-3. Il est conçu pour être sécurisé contre une large gamme d'attaques, notamment les attaques de pré-image, les attaques de collision et les attaques d'extension de longueur. Il est également efficace, avec un coût de calcul relativement faible et une mise en œuvre simple, ce qui le rend idéal pour son utilisation dans les contrats intelligents. [38](#)
- MimcSponge :** MiMC est une famille de fonctions de chiffrement par bloc et de hachage conçue spécifiquement pour les applications SNARK. La faible complexité multiplicative de MiMC sur les champs premiers la rend adaptée à la plupart des applications ZK-SNARK. [30](#)
- NFT :** Selon le dictionnaire Robert, c'est un certificat cryptographique associé à un objet numérique (image, vidéo, musique...) dont l'authenticité et la traçabilité sont garanties par la blockchain. [8](#)

OFF-CHAIN :	L'analyse hors chaîne est l'évaluation des transactions et des transferts qui ont lieu en dehors de la blockchain, souvent au sein ou entre les bourses de crypto-monnaies et d'autres acteurs du marché. 9
ON-CHAIN :	Se réfère aux transactions et données qui sont directement stockées et traitées sur la blockchain. Une transaction on-chain est validée par les nœuds du réseau et enregistrée de manière permanente dans les blocs de la chaîne. Toutes les opérations on-chain sont publiques et traçables. 9
OSINT :	L'Open-Source Intelligence (OSINT) est défini comme le renseignement produit par la collecte, l'évaluation et l'analyse d'informations accessibles au public dans le but de répondre à une question de renseignement spécifique. 9
Reentrancy attack :	Le mot "Reentrancy" fait référence à une caractéristique qui est présente dans les caractéristiques des smart contrats. Cette caractéristique donne la possibilité à certaines fonctions des smart contrats d'être exécutées plusieurs fois, ce qui est en réalité une faille qui est profitable pour les hackers. Cette faille est souvent utilisée pour attaquer un smart contrat en faisant une demande de retrait de fonds, suivi de plusieurs autres demandes en simultané. L'objectif étant de tromper le protocole et de pouvoir réaliser un retrait qui sera supérieur au montant disponible. 38

Introduction Générale

Ces dernières années, les cryptomonnaies et la technologie blockchain ont progressivement gagné en popularité, et ont changé notre manière de réaliser des transactions financières. La blockchain Ethereum, en particulier, a émergé comme une plateforme majeure pour les transactions financières décentralisées et l'exécution de contrats intelligents (smart contracts). Grâce à sa nature décentralisée, elle offre des avantages comme la transparence, la sécurité et l'immutabilité des transactions.

Mais cette transparence pose beaucoup de problèmes en matière de confidentialité et de protection des données personnelles car toutes les transactions sur Ethereum sont visibles publiquement, ce qui permet à quiconque de consulter les adresses des expéditeurs et des destinataires, les montants transférés et l'historique complet des transactions associées à une adresse donnée. Ce qui expose les utilisateurs à des risques tels que la dé-anonymisation, la surveillance et le traçage et surtout des atteintes à la vie privée parce que les transactions financières en général révèlent des informations sensibles sur les habitudes et les préférences des utilisateurs.

Parallèlement, les réglementations financières internationales, telles que le [KYC/AML](#), imposent aux institutions financières des obligations strictes de vérification de l'identité des clients et de surveillance des transactions suspectes. Ces exigences visent à prévenir le blanchiment d'argent, le financement du terrorisme et d'autres activités illicites. Cependant, ces réglementations entrent en conflit avec le désir des utilisateurs de préserver leur anonymat sur les plateformes décentralisées.

Notre mémoire propose le développement du prototype d'un outil d'anonymisation des transactions financières sur la Blockchain Ethereum, l'objectif étant de concilier l'anonymat des transactions avec la conformité réglementaire, offrant ainsi une solution viable pour protéger la confidentialité des utilisateurs sans compromettre les obligations légales.

Problématique

La blockchain Ethereum, même s'il offre une plateforme décentralisée et sécurisée pour les transactions financières, soulève un problème majeur en matière de confidentialité. En effet, toutes les transactions sont publiques et peuvent être consultées par tout le monde, ce qui expose les utilisateurs à des risques de dé-anonymisation et de violation de leur vie privée. D'autre part, les réglementations financières telles que le Know Your Customer (KYC) et l'Anti-Money Laundering (AML) imposent aux institutions le monitoring des transactions afin de prévenir les activités illégales. L'objectif de

ce mémoire est de répondre à comment garantir la confidentialité et l'anonymat des transactions financières sur la blockchain Ethereum tout en respectant les réglementations KYC/AML.

Ainsi, dans le cadre de notre étude, plusieurs questions sont soulevées : Comment mettre en place une solution technique qui assure l'anonymat des transactions sans compromettre la sécurité et la transparence de la blockchain ? De quelle manière intégrer des mécanismes de conformité réglementaire qui protègent les informations personnelles des utilisateurs tout en satisfaisant aux obligations légales ? Quels outils et technologies peuvent être utilisés pour atteindre cet équilibre entre anonymat et conformité ?

Objectifs du mémoire

Notre travail vise à développer un outil d'anonymisation des transactions financières sur la blockchain Ethereum. Les objectifs spécifiques sont :

- Concevoir un modèle de transaction anonyme en intégrant les techniques cryptographiques (les Zk-SNARKS) et en utilisant les smart contracts pour automatiser le processus.
- Intégrer un mécanisme de Zero-Knowledge KYC pour permettre de vérifier l'identité des utilisateurs sans divulguer leurs informations personnelles sur la blockchain.
- Evaluer les performances et la sécurité de notre solution.

Méthodologie

Pour atteindre ces objectifs, une approche méthodologique sera adoptée :

- Étudier les solutions existantes liées à l'anonymisation sur la blockchain.
- Analyser les réglementations actuelles en matière de KYC/AML.
- Définir l'architecture du système en combinant les technologies appropriées.
- Développer le prototype en utilisant Solidity pour les smart contracts et Circom pour les zk-SNARKs.
- Effectuer des tests fonctionnels et de sécurité sur des réseaux de test Ethereum.

Organisation du mémoire

Le présent mémoire est structuré en quatre (04) chapitres principaux. Le premier chapitre établit un état de l'art en explorant les concepts fondamentaux liés à la blockchain, à la confidentialité et aux réglementations KYC/AML. Le deuxième chapitre se concentre sur les matériels et méthodes à utiliser pour la conception du prototype. Le troisième chapitre présente le développement pratique du prototype. Enfin, le quatrième et dernier chapitre discute des résultats obtenus, des limites rencontrées et des perspectives d'amélioration pour des recherches futures.

Etat de l'art

Introduction

Dans un contexte marqué par l'adoption croissante des technologies de la blockchain, la confidentialité des transactions financières et la conformité réglementaire représentent des enjeux majeurs.

Ce chapitre explore les concepts fondamentaux et les technologies existantes qui permettent d'aborder ces problématiques. Il examine tout d'abord les principes de base de la blockchain et ses limites en matière de confidentialité. Ensuite, il s'intéresse aux exigences des réglementations KYC/AML et aux défis qu'elles posent dans les environnements décentralisés. Enfin, une analyse des solutions actuelles d'anonymisation, telles que les services de mixage et les preuves à divulgation nulle (Zero-Knowledge Proofs), permet d'évaluer leurs forces, leurs faiblesses et leur compatibilité avec les exigences réglementaires.

1.1 Généralités sur la blockchain

1.1.1 Définition

La plupart des gens confondent la blockchain et les cryptomonnaies, en particulier le Bitcoin. Cependant, la blockchain n'est pas une cryptomonnaie, mais la technologie derrière les cryptomonnaies.

La blockchain est une technologie de registre distribué (distributed ledger technology) qui permet le stockage et la transmission d'informations de manière transparente, sécurisée et immuable entre plusieurs parties, sans nécessiter une autorité centrale. Elle permet donc de conserver un registre public, anonyme et infalsifiable de toutes les transactions effectuées entre ses utilisateurs.

[3]La blockchain, conceptualisée dès 1982 par David Chaum, a été initialement développée pour sécuriser les transactions numériques grâce à la cryptographie. En 1991, Stuart Haber et W. Scott Stornetta ont introduit un système de chaînes de blocs pour horodater les documents numériques. Cependant, c'est en 2008 que la blockchain a véritablement pris son essor avec la création du Bitcoin par Satoshi Nakamoto. Ce système décentralisé de pair-à-pair a révolutionné le monde des transactions en ligne, offrant une alternative sécurisée et transparente aux systèmes traditionnels.

1.1.2 Les types de blockchain

Les blockchains peuvent être classées en plusieurs catégories en fonction de leur architecture, de leur niveau d'accessibilité et de leur mode de fonctionnement. Il existe quatre (04) types principaux de blockchain :

- Les blockchains publiques

Les blockchains publiques sont des réseaux ouverts où quiconque peut participer à la validation des transactions, visualiser l'historique des transactions et créer de nouvelles transactions. Sur ce type de blockchain, la confidentialité est garantie, car les identités des nœuds et des mineurs restent protégées. En parallèle, la transparence est maintenue puisque toutes les transactions sont visibles par l'ensemble des nœuds du réseau. La robustesse du registre est assurée par sa nature décentralisée, chaque nœud en possédant une copie complète, ce qui élimine tout risque de défaillance unique.

Enfin, comme ces blockchains fonctionnent sur un réseau décentralisé, ils s'appuient sur un mécanisme de consensus, comme le Proof of Work ou le Proof of Stake, pour valider les transactions.

- Les blockchains privées

Les blockchains privées, également appelées blockchains autorisées, sont des réseaux où l'accès est contrôlé par une ou plusieurs entités. Seuls les utilisateurs autorisés peuvent valider des transactions ou participer à la maintenance du réseau. Cela signifie que les transactions et les données sur une blockchain privée ne sont visibles que par les participants approuvés, ce qui renforce la confidentialité et le contrôle des informations sensibles.

Cet accès contrôlé dans les blockchains privées offre un niveau plus élevé de confidentialité et de sécurité, ce qui les rend idéaux pour certaines applications professionnelles. Ce contrôle centralisé soulève quand même des inquiétudes quant à une manipulation potentielle par l'entité dirigeante. De plus, l'accès restreint entre en conflit avec les principes fondamentaux de la décentralisation qui sous-tendent la technologie blockchain.

- Les blockchains de Consortium

Les blockchains de consortium sont un type de blockchain dans lequel plusieurs entités autorisées collaborent pour gérer le réseau. Elles peuvent être considérées comme un sous-ensemble des blockchains privées, mais ne sont pas aussi rigides, ce qui permet à certaines entités de partager et d'accéder aux informations en toute sécurité au sein de leur cercle.

Les blockchains de consortium conservent un certain niveau de décentralisation, ce qui les rend plus sûres et plus efficaces que les systèmes traditionnels.

1.1.3 Architecture et fonctionnement de la blockchain

1.1.3.1 Architecture de la blockchain

La blockchain repose sur une structure composée de blocs de données reliés entre eux de façon chronologique et sécurisés par des mécanismes cryptographiques. Chaque bloc de la blockchain contient un ensemble d'informations, principalement des transactions validées, structurées de manière à garantir l'intégrité et la transparence. Un bloc est composé de deux parties principales : l'en-tête et le corps.

L'en-tête du bloc contient des métadonnées essentielles au fonctionnement de la blockchain. Elle inclut le hash du bloc précédent, un identifiant cryptographique qui crée un lien entre les blocs et assure leur ordre chronologique. Cette chaîne de hachage empêche toute modification rétroactive sans invalider l'ensemble de la chaîne. L'en-tête intègre également une marque temporelle (timestamp)

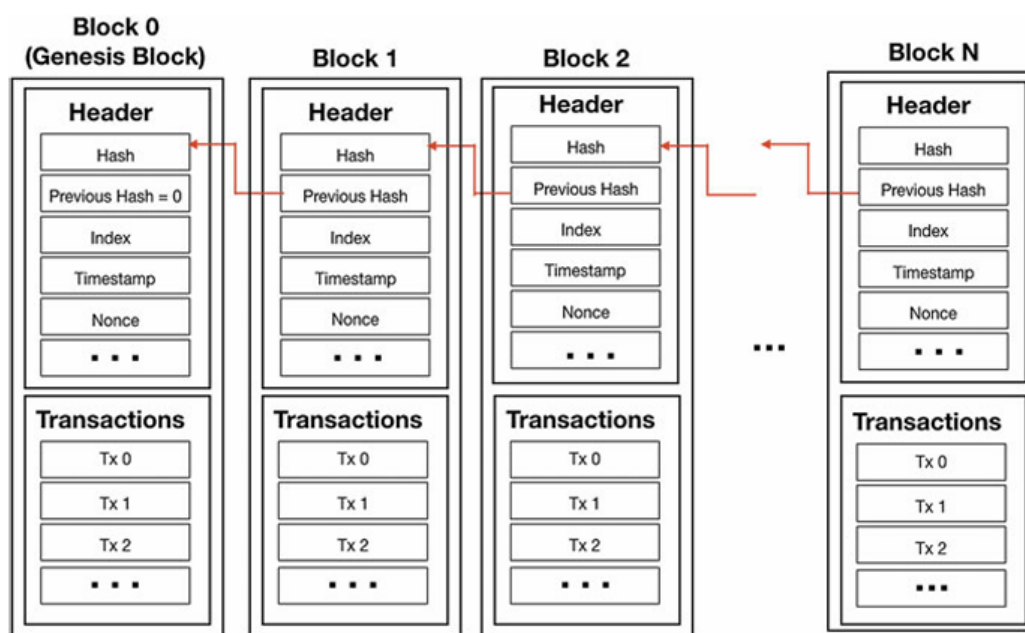


FIGURE 1.1 – Structure d'une blockchain

qui indique la date et l'heure de création du bloc, ainsi qu'un nonce, une valeur utilisée dans le cadre de l'algorithme de consensus pour résoudre un problème cryptographique. Une autre composante clé de l'en-tête est la racine de Merkle (Merkle Root), un hash unique qui résume toutes les transactions incluses dans le bloc et permet de vérifier l'intégrité des données sans devoir analyser chaque transaction individuellement.

Le corps du bloc, quant à lui, contient la liste des transactions validées par le réseau. Chaque transaction est représentée par un hash, garantissant son unicité et sa sécurité. Ces transactions incluent des informations essentielles telles que l'adresse de l'expéditeur, celle du destinataire, et le montant transféré. La structure organisée en arbre de Merkle optimise le stockage et permet une vérification rapide de l'appartenance d'une transaction au bloc, tout en minimisant l'espace nécessaire.

1.1.3.2 Fonctionnement de la blockchain

Le fonctionnement d'une blockchain repose sur un processus structuré de validation, d'enregistrement et de sécurisation des transactions.

- Création des transactions

Le processus commence lorsqu'un utilisateur souhaite effectuer une transaction, telle que le transfert de crypto-monnaies. Pour ce faire, l'utilisateur crée une requête qui contient les détails essentiels de la transaction, notamment l'adresse du destinataire, le montant à transférer, et éventuellement d'autres informations spécifiques. Afin de prouver qu'il est l'initiateur légitime de cette opération, l'utilisateur signe numériquement la transaction à l'aide de sa clé privée. Cette signature cryptographique garantit l'authenticité de la transaction et empêche toute modification ultérieure sans invalider la signature.

- Création et ajouts blocs à la chaîne

Les transactions validées sont regroupées dans un bloc par des nœuds spécifiques, souvent appelés "mineurs" ou "validateurs", en fonction du mécanisme de consensus utilisé. Dans les blockchains fonctionnant sur le principe du Proof of Work (PoW), les mineurs doivent résoudre un problème

cryptographique complexe en trouvant un nonce approprié, un processus connu sous le nom de "minage". Ce processus est énergivore et demande du temps et des ressources informatiques. Une fois le bloc créé avec un hash valide, il est prêt à être soumis au réseau pour validation.

Le bloc nouvellement créé est soumis aux autres nœuds du réseau pour validation via un mécanisme de consensus, qui peut varier selon la blockchain (PoW, Proof of Stake (PoS), etc.). Une fois que les nœuds parviennent à un accord sur la validité du bloc, celui-ci est ajouté à la blockchain. Toutes les copies locales de la blockchain détenues par les nœuds sont alors mises à jour pour inclure ce nouveau bloc, garantissant la synchronisation du réseau.

- Confirmation des Transactions

Après l'ajout du bloc à la chaîne, chaque nœud met à jour son état pour refléter les nouvelles transactions confirmées. Les transactions incluses dans le bloc sont considérées comme définitives une fois qu'un certain nombre de blocs supplémentaires ont été ajoutés après celui-ci. Ce processus, connu sous le nom de "finalité", rend les transactions immuables et irréversibles, renforçant ainsi la sécurité et la confiance dans le réseau blockchain.

1.1.3.3 Mécanismes de consensus

Pour qu'un registre distribué puisse fonctionner correctement, il doit disposer d'un moyen convenu de valider les transactions. Ce processus est appelé consensus, et la logique utilisée pour y parvenir est connue sous le nom de mécanisme de consensus ou algorithme de consensus. Ces mécanismes s'assurent que tous les participants du réseau s'accordent sur l'état actuel de la chaîne et la validité des transactions [15]. Voici les principaux mécanismes de consensus utilisés :

- Proof of Work (PoW) :

Le Proof of Work, utilisé notamment par Bitcoin et initialement par Ethereum, est le premier protocole de validation inventé. Il repose sur un système compétitif où les nœuds (mineurs) doivent résoudre des problèmes mathématiques complexes pour valider un bloc. Le premier à avoir résolu le problème est celui qui obtient le droit de valider la transaction et donc créer un nouveau bloc dans la blockchain. Après chaque transaction réussie, des jetons de crypto-monnaie sont créés et distribués entre les différents mineurs. Il a l'avantage d'être sécurisé et robuste mais nécessite une grande consommation d'énergie.

- Proof of Stake (PoS) :

Le Proof of Stake remplace la compétition énergétique par un système basé sur la possession de tokens. Les validateurs sont choisis proportionnellement à leur participation dans le réseau. Les utilisateurs qui détiennent et "stakent" (verrouillent) leurs tokens deviennent des validateurs. Plus un utilisateur stake de tokens, plus il a de chances d'être sélectionné pour valider un bloc.

La validation des blocs avec ce mécanisme de consensus est plus rapide et la consommation énergétique réduite. Cependant, il y a un risque de centralisation à long terme si quelques acteurs détiennent la majorité des tokens. Il est utilisé par Ethereum après sa mise à jour vers Ethereum 2.0 [8].

- Delegated Proof of Stake (DPoS) :

La DPoS est un mécanisme de consensus où les utilisateurs du réseau votent et élisent des délégués pour valider le bloc suivant. À l'instar d'un mécanisme traditionnel de PoS, le DPoS utilise un

système de jalonnement collatéral. Cependant, il utilise également un processus démocratique spécifique conçu pour remédier aux limites du PoS. Cela lui permet d'offrir un moyen plus abordable, plus efficace et plus équitable de valider les transactions. Cependant, il y a un potentiel de corruption parmi les délégués en raison de la centralisation, car il est plus centralisé que le PoS standard.

1.1.4 La blockchain Ethereum

1.1.4.1 Définition

Ethereum est une plateforme blockchain décentralisée et open-source, créée en 2015 par Vitalik Buterin et Gavin Wood. Contrairement à Bitcoin, qui se concentre principalement sur les transactions financières et le stockage de valeur, Ethereum a été conçu pour étendre les capacités de la technologie blockchain en permettant de créer et de gérer des contrats intelligents et des applications décentralisées (dapps) sans avoir besoin d'une autorité centrale. Ethereum possède sa propre cryptomonnaie native, l'Ether (ETH), qui est utilisée pour payer certaines activités sur le réseau. Il peut être transféré à d'autres utilisateurs ou échangé contre d'autres jetons sur Ethereum. L'Ether est spécial parce qu'il est utilisé pour payer le calcul nécessaire pour construire et exécuter des applications et des organisations sur Ethereum [1].

Avant la mise à jour vers Ethereum 2.0, Ethereum utilisait le modèle PoW, mais à partir de la mise à niveau d'Ethereum 2.0, Ethereum a transitionné vers un modèle PoS, où les validateurs sont choisis en fonction de la quantité d'Ether qu'ils détiennent et sont prêts à "staker". Ce changement vise à réduire les coûts énergétiques et à améliorer l'efficacité du réseau.

1.1.4.2 Ethereum Virtual Machine

La spécificité d'Ethereum réside dans sa machine virtuelle, l'Ethereum Virtual Machine (EVM), qui sert de moteur d'exécution pour ces contrats intelligents. La machine virtuelle Ethereum (EVM) est un environnement virtuel décentralisé qui exécute le code de manière cohérente et sécurisée sur tous les nœuds Ethereum. Les nœuds exécutent l'EVM qui exécute des contrats intelligents, utilisant du «gaz» pour mesurer l'effort de calcul requis pour les opérations, garantissant une allocation efficace des ressources et la sécurité du réseau.

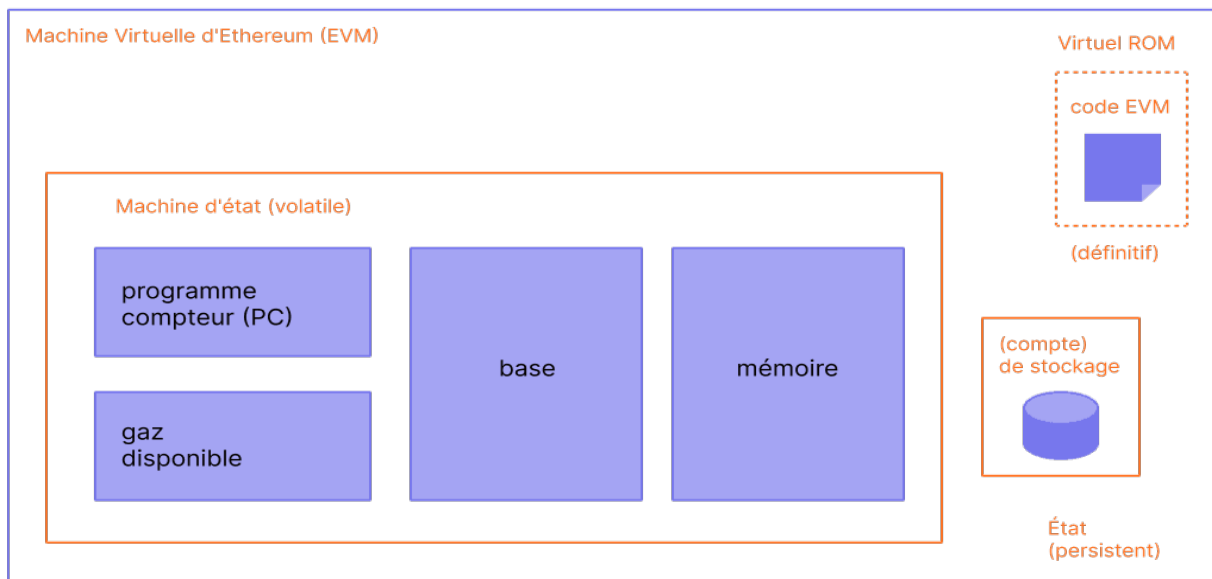


FIGURE 1.2 – Ethereum Virtual Machine

Cette capacité qu'a Ethereum a permis de révolutionner des secteurs variés, tels que la finance décentralisée (DeFi) [6], les jetons non fongibles (NFT), et la gestion de la chaîne d'approvisionnement.

1.1.4.3 Le gaz

Le gaz est l'unité de mesure qui quantifie l'effort de calcul nécessaire pour exécuter des opérations sur le réseau Ethereum [10]. Étant donné que chaque transaction sur Ethereum mobilise des ressources informatiques, un coût est associé à leur utilisation. Cela permet de protéger le réseau contre les spams et d'éviter les blocages causés par des boucles de calcul infinies. Ces coûts sont couverts par des frais de gaz.

Les frais de gaz sont déterminés en multipliant la quantité de gaz consommée par le prix unitaire du gaz. Ils doivent être payés en Ether (ETH). Le prix du gaz est généralement indiqué en gwei, qui est une dénomination de l'ETH. Chaque gwei est égal à un milliardième d'ETH (0,000000001 ETH ou 10^{-9} ETH).

Ils sont dus indépendamment du succès ou de l'échec de la transaction, assurant ainsi une compensation pour l'effort fourni par le réseau.

1.1.4.4 Les contrats Intelligents (smarts contracts)

Un contrat intelligent, ou smart contract, est un programme autonome exécuté sur la blockchain d'Ethereum. C'est un ensemble de code (ses fonctions) et de données (son état) qui réside à une adresse spécifique sur la blockchain Ethereum. Il est utilisé pour automatiser des processus spécifiques en fonction de règles prédéfinies, sans nécessiter d'intervention humaine après leur déploiement.

1.1.4.5 Les dApp

Une application décentralisée (dApp) est une application construite sur un réseau décentralisé qui combine un contrat intelligent et une interface utilisateur en frontend. La dApp a son code backend qui s'exécute sur la blockchain, principalement Ethereum, contrairement aux applications traditionnelles, dont le code du backend est exécuté sur des serveurs centralisés.

1.2 Anonymat et Confidentialité sur la blockchain Ethereum

1.2.1 Différence entre pseudonymat et anonymat réel

1.2.1.1 Introduction des concepts

La blockchain, en tant que technologie publique et décentralisée, offre un certain niveau de confidentialité à ses utilisateurs. Cependant, il est crucial de distinguer deux notions souvent confondues : le pseudonymat et l'anonymat réel.

Le pseudonymat signifie que chaque utilisateur d'une blockchain publique, comme Bitcoin ou Ethereum [22], est identifié par une clé publique, souvent représentée par une adresse alphanumérique unique. Cette adresse ne contient pas directement d'informations personnelles. Toutefois, toutes les transactions associées à cette adresse sont visibles sur le registre public, ce qui permet de retracer l'historique complet des opérations. Ainsi, bien que l'identité réelle de l'utilisateur ne soit pas immédiatement apparente, des techniques d'analyse, comme l'**OSINT**[20], l'**ON-CHAIN** et l'**OFF-CHAIN ANALYSIS** [19], peuvent relier ces adresses à des personnes physiques ou morales.

User to Wallet Address - Process overview

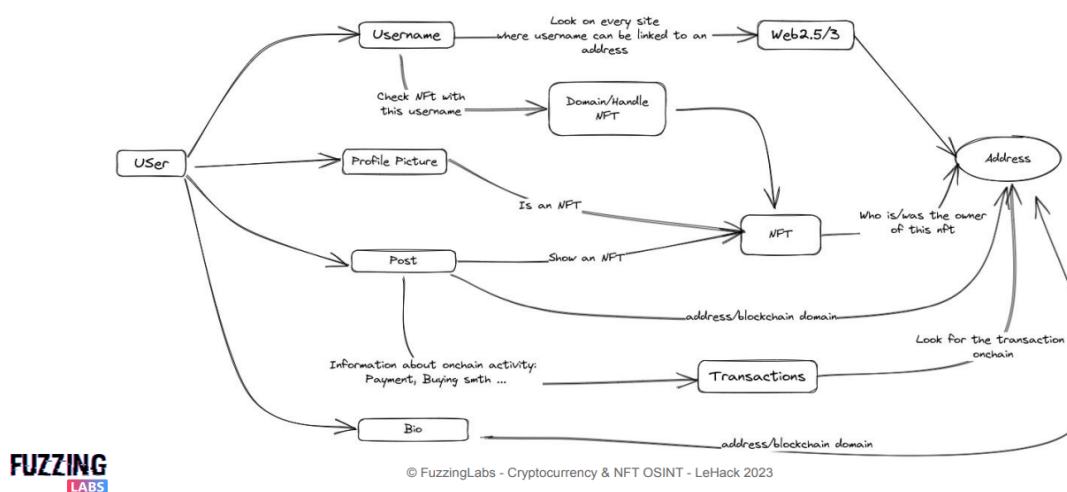


FIGURE 1.3 – Processus utilisateur à l'adresse du portefeuille

En revanche, l'anonymat réel vise à garantir qu'aucune information identifiable, même indirectement, ne puisse être reliée à une transaction ou à une adresse. Cela nécessite des mécanismes sophis-

tiqués, tels que l'obfuscation des données ou l'utilisation de protocoles cryptographiques avancés, comme les preuves à divulgation nulle de connaissance (Zero-Knowledge Proofs). Par exemple, des blockchains comme Monero ou Zcash sont conçues pour masquer non seulement les identités des utilisateurs, mais aussi les montants des transactions et les adresses des parties impliquées.

Ainsi, la distinction fondamentale réside dans le niveau de traçabilité. Le pseudonymat est souvent suffisant pour des transactions ordinaires, mais il reste vulnérable à des analyses approfondies. À l'inverse, l'anonymat réel offre une protection accrue, essentielle dans des contextes où la vie privée est une priorité absolue.

1.2.1.2 Implication pour les utilisateurs

Les concepts de pseudonymat et d'anonymat réel sur la blockchain ont des implications significatives pour les utilisateurs, en particulier en ce qui concerne leur confidentialité, leur sécurité et les conformités réglementaires.

Le pseudonymat, bien qu'il offre une protection de base en masquant l'identité réelle derrière une adresse publique, reste insuffisant face aux techniques avancées d'analyse. Par exemple, des acteurs malveillants ou des agences de surveillance peuvent utiliser des outils d'analyse blockchain pour identifier des utilisateurs en croisant des informations issues d'autres sources. Cela peut compromettre la confidentialité, surtout si les utilisateurs ne prennent pas de mesures supplémentaires, comme l'utilisation de mixeurs de transactions ou de portefeuilles non custodials [5].

En revanche, les blockchains axées sur l'anonymat réel, telles que Monero ou les solutions basées sur le ZK-SNARK sur Ethereum, offrent une confidentialité renforcée. Ces mécanismes réduisent les risques de suivi, ce qui est crucial pour les utilisateurs opérant dans des environnements sensibles. Cependant, l'anonymat réel peut entrer en conflit avec les exigences de conformité imposées par les réglementations, telles que les directives KYC et AML. Les autorités financières craignent que des systèmes offrant un anonymat total puissent être exploités pour des activités illicites, comme le blanchiment d'argent ou le financement du terrorisme. Cela peut limiter l'adoption des solutions totalement anonymes dans certains contextes, notamment dans les systèmes financiers régulés.

Pour les utilisateurs, le défi consiste à trouver un équilibre entre préserver leur vie privée et respecter les contraintes réglementaires. Les innovations, comme le Zero-Knowledge KYC (zk-KYC), offrent une voie prometteuse. Elles permettent aux utilisateurs de prouver leur identité ou leur conformité sans divulguer d'autres informations personnelles, combinant les avantages de l'anonymat réel avec les exigences réglementaires.

1.3 Régulations et Conformité : KYC et AML

1.3.1 Introduction aux réglementations financières

Les réglementations financières jouent un rôle fondamental dans la stabilité, la transparence et la sécurité des systèmes économiques modernes. Elles visent à instaurer des règles et des mécanismes de contrôle pour protéger les consommateurs, maintenir la confiance dans les institutions du secteur de la finance, et prévenir les abus tels que la fraude, le blanchiment d'argent ou le financement du

terrorisme.

Avec l'émergence des technologies de la blockchain et des cryptomonnaies, ces régulations ont dû évoluer pour répondre aux défis spécifiques qu'elles posent. Les cryptomonnaies, bien qu'innovantes et porteuses d'opportunités économiques, introduisent également des risques liés à leur pseudo-anonymat, leur caractère transfrontalier, et l'absence d'autorité centrale. Ces caractéristiques rendent difficile la mise en œuvre des régulations financières traditionnelles.

1.3.2 Blanchiment d'argent via les cryptomonnaies : Une tendance ces dernières années

Selon un rapport du FBI [9], les pertes liées aux escroqueries en matière d'investissement en cryptomonnaies aux États-Unis seulement ont totalisé 3,94 milliards de dollars en 2023, soit une augmentation de 53% par rapport à 2022.

Le blanchiment d'argent demeure une problématique majeure dans l'univers des cryptomonnaies, où les criminels tentent de tirer parti de l'anonymat et de la décentralisation pour dissimuler des fonds d'origine illégale. Plusieurs tendances significatives se dessinent dans ce contexte :

- **Utilisation des mixers et tumblers** : Ces outils permettent de brouiller les pistes en mélangeant des fonds légitimes avec des fonds illicites, rendant difficile l'identification de leur origine.
- **Cryptomonnaies axées sur la confidentialité** : Les monnaies numériques comme Monero et Zcash sont spécialement conçues pour offrir des fonctionnalités de confidentialité avancées. Ces caractéristiques rendent le suivi des transactions et l'identification des parties impliquées plus complexes pour les autorités.
- **Exploitation des plateformes décentralisées et des échanges décentralisés (DeFi et DEXs [7])** : Les échanges décentralisés (DEXs) permettent aux utilisateurs d'échanger des cryptomonnaies sans passer par une autorité centrale, réduisant ainsi le risque de détection et de régulation. Parallèlement, les services financiers décentralisés (DeFi), souvent soumis à une réglementation limitée, sont également devenus une cible privilégiée pour les criminels cherchant à blanchir des actifs numériques.

1.3.3 Objectifs des régulations KYC et AML

Les régulations Know Your Customer et Anti-Money Laundering sont des composantes essentielles des systèmes de réglementation financière à travers le monde. Leur mise en œuvre vise à sécuriser le système financier et à protéger les utilisateurs contre diverses formes de fraude et d'activités illégales.

1.3.4 KYC : Know Your Customer

Par définition, KYC, ou Know Your Customer par ses sigles en anglais (Connaissance du Client), est la procédure que les entreprises réalisent pour vérifier l'identité de leurs clients conformément aux exigences légales et aux réglementations en vigueur [13]. L'objectif est d'assurer que le client est bien celui qu'il prétend être, en identifiant ses informations personnelles et financières, notamment à travers des documents officiels (passeport, justificatif de domicile, relevés bancaires, etc.). Ce processus est crucial pour prévenir la fraude, le blanchiment d'argent, et d'autres activités illégales.

Le processus de KYC est généralement instauré dès le premier contact avec l'utilisateur et se poursuit tout au long de la relation entre celui-ci et l'entreprise. Dans le secteur financier, cela inclut notamment l'examen des transactions afin d'identifier celles qui pourraient présenter un risque potentiel. Bien que le KYC soit principalement associé aux banques, il s'applique également à des entreprises opérant dans les domaines de la DeFi, des Fintech, de l'investissement, du droit, de l'assurance, et bien d'autres secteurs nécessitant une conformité réglementaire stricte.

1.3.5 AML : Anti Money Laundering

Le blanchiment d'argent représente un défi global qui affecte les économies à l'échelle internationale. Ce procédé est exploité par les criminels pour dissimuler leurs activités illicites et donner une apparence légitime aux revenus tirés de leurs infractions.

La lutte contre le blanchiment d'argent (AML) est le processus de prévention, de détection et de signalement de l'utilisation illégale de fonds ou d'actifs provenant d'activités criminelles ou utilisés pour financer le terrorisme [2]. Le blanchiment d'argent est une préoccupation mondiale qui ne touche pas seulement le secteur financier, mais également d'autres secteurs qui traitent des transferts d'argent, comme les entreprises dans le domaine de la blockchain et des cryptomonnaies. Ces entreprises sont plus vulnérables aux risques de blanchiment d'argent car elles opèrent dans un environnement numérique et sans frontières en évolution rapide, où les transactions peuvent être anonymes, complexes et difficiles à retracer.

Par conséquent, il est important pour ces entreprises de se conformer aux réglementations AML pour protéger leur réputation et leurs clients contre les effets néfastes du blanchiment d'argent.

1.3.6 Impact des réglementations sur les transactions financières sur la blockchain

Les réglementations financières ont un impact significatif sur les transactions effectuées via la blockchain, notamment sur des plateformes comme Ethereum. Dans un environnement en constante évolution, où les innovations technologiques rencontrent des préoccupations réglementaires, il est crucial de comprendre comment ces réglementations affectent la dynamique des transactions financières. Cette section présente les principaux impacts des réglementations sur les transactions financières sur la blockchain.

1.3.6.1 Conformité et adhésion aux normes

La technologie blockchain a bouleversé les systèmes financiers traditionnels en permettant des transactions décentralisées, transparentes et infalsifiables. Cependant, cette innovation pose également des défis aux régulateurs. Ils doivent trouver un équilibre entre la promotion de l'innovation et la protection contre les risques divers.

Les plateformes de blockchain doivent se conformer aux lois et règlements en matière de KYC et d'AML. Cela signifie que, même si la blockchain permet une certaine forme d'anonymat, les acteurs doivent mettre en œuvre des processus rigoureux pour vérifier l'identité des utilisateurs.

Ces exigences peuvent entraîner des coûts supplémentaires pour les entreprises qui développent des applications basées sur la blockchain, car elles doivent intégrer des systèmes d'identification et de vérification des clients.

1.3.6.2 Sécurisation des transactions

Les réglementations peuvent renforcer la sécurité des transactions financières sur la blockchain. En instituant des normes pour l'identification des utilisateurs et la vérification des transactions, les risques de fraude ou d'activités criminelles sont réduits.

La clarté juridique apportée par certaines réglementations permet également une meilleure gestion des litiges associés aux transactions. En cas de différend, les parties peuvent avoir des recours judiciaires définis, ce qui est souvent difficile à établir dans un cadre totalement décentralisé.

1.4 Techniques d'anonymisation des transactions sur la blockchain

1.4.1 Importance de l'anonymisation dans le contexte de la blockchain Ethereum

1.4.2 Solutions Traditionnelles d'Anonymisation : Mixers et Tumblers

1.4.2.1 Définition

Les mélangeurs de crypto-monnaies, également connus sous le nom de crypto mixers, crypto tumblers ou crypto blenders, sont des services d'anonymisation de transactions sur la blockchain qui permettent de cacher l'origine et la destination des transactions en les mélangeant au hasard avec d'autres transactions [4]. Il devient donc difficile de faire un lien et de retrouver les adresses exactes impliquées dans une transaction spécifique.

Un élément clé pour comprendre l'importance des mixeurs de cryptomonnaies réside dans la nature pseudonyme de la plupart des blockchains. Sur une blockchain publique comme Bitcoin et Ethereum, chaque transaction et chaque adresse de portefeuille sont visibles publiquement, permettant à n'importe qui d'y accéder. Cette transparence, bien qu'avantageuse pour la traçabilité, limite l'anonymat complet des utilisateurs, d'où la nécessité d'outils comme les mixeurs pour renforcer la confidentialité des transactions.

1.4.2.2 Fonctionnement

Les mixeurs et tumblers regroupent les fonds de plusieurs utilisateurs dans un pool commun, où les cryptomonnaies sont mélangées. Ensuite, des montants équivalents, mais provenant d'adresses différentes, sont redistribués aux destinataires finaux. Cela rompt le lien direct entre l'adresse d'origine et l'adresse de destination. Voici les étapes clés du processus :

- **Envoi des fonds au mixeur :** L'utilisateur envoie ses fonds sur une des adresses fournies par le mixeur en échange d'un secret. La transaction est enregistrée sur la blockchain de manière à ne pas lier l'adresse de dépôt à une adresse de retrait plus tard.
- **Mélange des fonds :** Une fois les fonds de plusieurs utilisateurs regroupés dans un pool unique, le mixer mélange les fonds avec différentes techniques.
- **Retrait des fonds :** L'utilisateur peut maintenant fournir une adresse de son choix ou une adresse de relai créée pour l'occasion pour retirer ses fonds en fournissant le secret.

Pour renforcer l'anonymat, certains mixers prennent en charge plusieurs blockchains, ce qui rend le traçage inter-chaînes difficile. Les mixers peuvent aussi fractionner les montants lors du retrait et mettre des délais aléatoires entre chaque retrait pour rendre plus complexe le traçage des transactions.

1.4.2.3 Limitations et préoccupations

Bien que les mixers de crypto-monnaies offrent une certaine confidentialité et un anonymat, ils sont largement utilisés par des criminels et des hackers pour le blanchiment d'argent et d'autres activités illégales. Ce qui fait que les mixeurs sont souvent considérés comme des plateformes illégales bien qu'ils ne soient pas explicitement illégaux dans la plupart des juridictions.

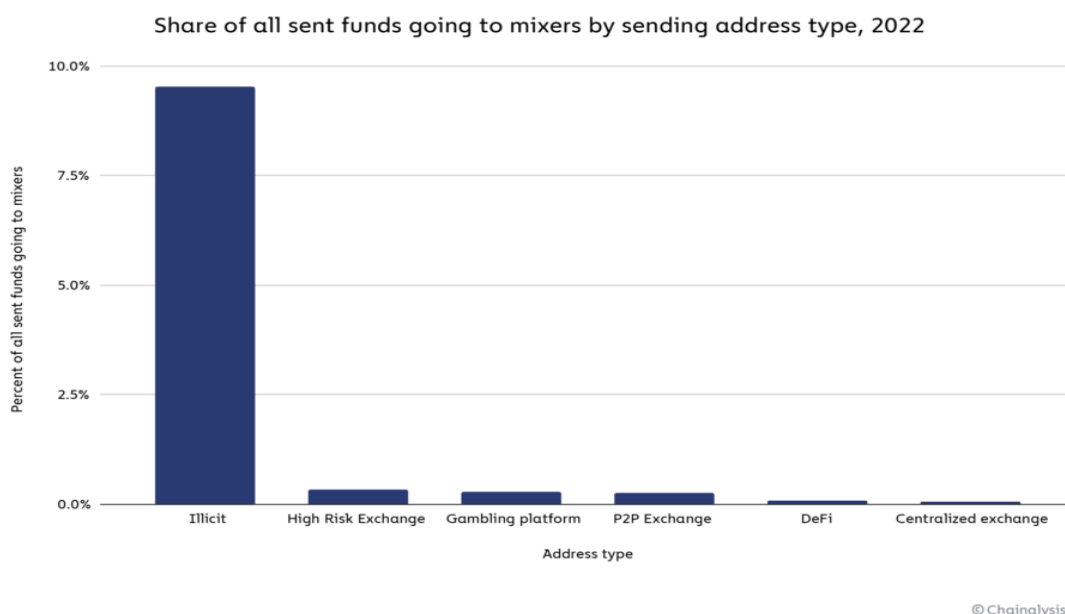


FIGURE 1.4 – Répartition des fonds envoyés sur les mixers en 2022

Comme exemple d'utilisation illégale des mixers, nous avons :

- la Corée du Nord , dont le groupe Lazarus, est considéré comme le plus grand groupe de hackers, a été responsable du vol de 1,7 milliard de dollars de crypto-monnaies en 2022 [14]. dont 455 millions de dollars ont été détournés via Tornado Cash, un mixer de crypto-monnaies.
- ChipMixer, un mélangeur de crypto-monnaies sans licence créé à la mi-2017, s'est spécialisé dans le mélange de crypto-monnaies, ont déclaré les autorités. Europol l'a décrit comme "l'une des plus grandes laves de crypto-monnaies du darkweb" et a déclaré que plus de 40 millions d'euros (42,2 millions de dollars) de crypto-monnaies avaient été saisis.

Au vu de l'utilisation détournée des mixers, les autorités de surveillance financière et de régulation des principales juridictions, notamment des États-Unis et de l'Union européenne, ont intensifié leurs efforts pour encadrer ces outils. Elles considèrent les mixeurs comme un point vulnérable dans la lutte contre le blanchiment d'argent et le financement du terrorisme, car ils permettent d'obscurcir

les origines des fonds.

Par exemple, des mixeurs comme Tornado Cash ont été ajoutés à la liste des entités sanctionnées, rendant illégal tout commerce ou interaction avec ces services pour les citoyens et entreprises américaines.

1.4.3 Les cryptomonnaies axées sur la confidentialité (Privacy coins)

1.4.3.1 Définition

Les cryptomonnaies axées sur la confidentialité, souvent appelées "privacy coins", sont conçues pour offrir aux utilisateurs un anonymat renforcé et une confidentialité accrue dans les transactions. Contrairement aux cryptomonnaies traditionnelles comme Bitcoin et Ethereum, où les transactions sont transparentes et traçables, les privacy coins offrent un voile d'invisibilité, ce qui permet de dissimuler les informations sur les transactions et les parties impliquées [21].

1.4.3.2 Principe de fonctionnement

L'anonymat n'étant pas une caractéristique par défaut des transactions de la blockchain, les privacy coins ont été conçues pour répondre à ce besoin. Elles utilisent plusieurs caractéristiques de conception pour obscurcir ou supprimer les données de transaction qui compromettent la vie privée de l'expéditeur ou du destinataire, telles que :

- **Signatures en anneau (Ring signatures)** : Elles permettent à un groupe d'utilisateurs de signer ensemble une transaction, ce qui rend difficile l'identification de l'utilisateur à l'origine de la transaction.
- **Adresses furtives (Stealth addresses)** : Ces adresses temporaires, générées de manière unique pour chaque transaction, empêchent de relier une adresse publique à un utilisateur spécifique.
- **Mixing et CoinJoin** : Ces processus combinent plusieurs transactions provenant de différents utilisateurs, brouillant ainsi les pistes et compliquant la traçabilité.
- **zk-SNARKs** : Utilisation de preuves cryptographiques pour valider des transactions sans révéler d'informations sur les parties impliquées ou les montants échangés.

1.4.3.3 Exemples de privacy coins

Il existe une multitude de privacy coins, mais les plus populaires sont :

- **Monero (XMR)** : Monero se distingue par une confidentialité obligatoire pour toutes les transactions, grâce à des technologies comme les signatures en anneau (qui masquent l'identité de l'expéditeur), les adresses furtives (qui créent des adresses uniques à usage unique pour chaque transaction), et les transactions confidentielles en anneau (qui dissimulent le montant des transactions). Ces mesures rendent Monero quasiment intraçable et très prisé pour des échanges nécessitant une discrétion totale.
- **Zcash (ZEC)** : Développé en 2016 par des chercheurs de grandes universités comme MIT et Johns Hopkins, Zcash utilise des preuves à divulgation nulle de connaissance (zk-SNARKs)

pour permettre des transactions protégées appelées "shielded transactions". Cette méthode garantit la validité des échanges sans révéler les informations relatives à l'expéditeur, au destinataire ou au montant. Cependant, Zcash offre également des transactions transparentes, permettant un équilibre entre confidentialité et conformité réglementaire.

1.4.4 Technologies avancées pour l'anonymisation

1.4.4.1 Zero-Knowledge Proofs (ZKP) : principes et applications.

Ce sont Shafi Goldwasser, Silvio Micali et Charles Rackoff qui ont utilisé, pour la première fois en 1985, le terme « preuve à divulgation nulle de connaissance », ou plus précisément sa forme anglaise, « zero knowledge proof », dans leur article fondateur⁸. Shafi Goldwasser et Silvio Micali ont reçu le prix Turing en 2012 pour leurs travaux [27].

Les zero-knowledge proofs (ZKP), ou preuves à divulgation nulle de connaissance en français, sont des protocoles permettant à une partie (Prover) de prouver à une autre partie (Verifier) l'exactitude d'une déclaration sans révéler aucune autre information que le fait que la déclaration est valide. Cela est particulièrement utile lorsque les informations sont sensibles et que le prouver ne souhaite pas que le verifier y ait accès.

Par exemple, cela permettrait de prouver à un utilisateur qu'il possède bien des ethers, sans pour autant indiquer sa signature ou son adresse publique.

Il est important de comprendre qu'une vérification de preuve à connaissance nulle n'effectue pas le calcul, elle vérifie uniquement que quelqu'un a effectué un calcul et produit un résultat revendiqué. Ce qui signifie que pour générer une ZKP, il faut avoir auparavant fait le calcul.

Principe de fonctionnement :

Les Zero-Knowledge Proofs (ZKP) reposent sur des concepts mathématiques avancés et des algorithmes cryptographiques complexes. Ils s'appuient notamment sur des fonctions de hachage cryptographique pour créer des défis aléatoires qui permettent au vérificateur et au prouveur d'établir une confiance réciproque sans révéler d'informations sensibles [28]. Les interactions régies par les ZKP doivent respecter certains critères fondamentaux à savoir :

- **Zéro connaissance (Zero-Knowledge)** : Le prouveur veut convaincre le vérificateur qu'une "déclaration spécifique" de ses "données confidentielles" est vraie, ZKP ne révèle rien d'autre que cette "déclaration spécifique".
- **Complétude (Completeness)** : Si la déclaration est vraie, le prouveur honnête devrait facilement pouvoir en convaincre le vérificateur.
- **Solidité (Soundness)** : Si la déclaration est fausse, le prouveur malhonnête ne devrait pas être en mesure de tromper le vérificateur.

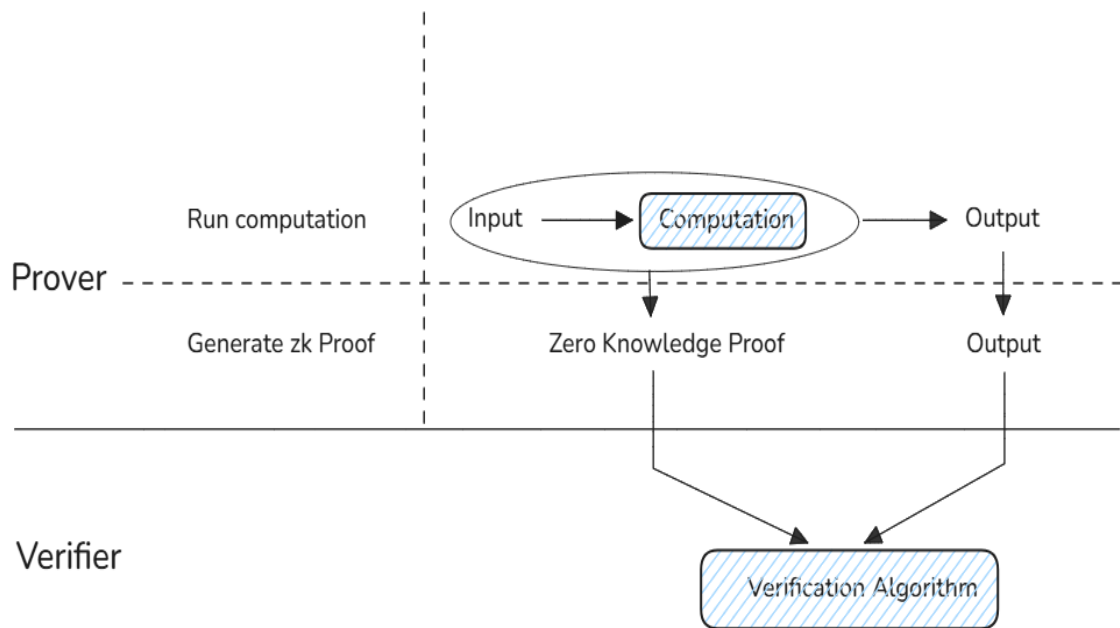


FIGURE 1.5 – Fonctionnement des ZKPs

Types de ZKP :

Il existe deux types de ZKP à savoir :

- **Interactive ZKP** : Ici, pour confirmer ces connaissances, le vérificateur pose au prouveur de nombreuses questions aléatoires, en exigeant des réponses immédiates. Ce processus garantit que le prouveur ne peut pas deviner les réponses. Le vérificateur continue de poser des questions jusqu'à ce qu'il soit certain que le prouveur connaît vraiment le secret revendiqué, d'où le caractère interactif. Lorsque le prouveur fournit systématiquement des réponses exactes, le vérificateur peut être certain des connaissances du prouveur.
- **Non-Interactive ZKP** : Dans une preuve à connaissance zéro non interactive (NIZKP), il n'y a pas de dialogue entre le prouveur et le vérificateur. Au lieu de cela, le prouveur génère une preuve complète en réponse à un défi simulé, souvent déterminé par une fonction de hachage cryptographique. Cette méthode repose sur une astuce ingénieuse : la preuve contient tous les éléments nécessaires pour convaincre le vérificateur sans qu'il soit nécessaire de poser des questions en temps réel. Grâce à l'heuristique de Fiat-Shamir [12], le défi est simulé comme s'il venait d'un vérificateur, mais il est en réalité dérivé d'un processus déterministe, rendant l'échange totalement autonome. Le vérificateur, en consultant la preuve, peut alors valider l'affirmation du prouveur sans nécessiter d'interaction.

Exemple d'applications :

Les preuves à connaissance zéro (ZKP) trouvent des applications dans divers domaines grâce à leur capacité à prouver des informations sans révéler les données elles-mêmes. Voici quelques cas d'utilisation majeurs des ZKP :

- **Les transactions anonymes** : La mise en place d'un protocole ZKP peut rendre anonymes des transactions de cryptomonnaie par exemple. Par exemple, certaines blockchains utilisent des

procédés ZKP pour garantir la confidentialité de leurs utilisateurs. Les systèmes ZKP peuvent aussi être déployés au sein de smart-contracts afin d'apporter de l'anonymat au sein des transactions. C'est le cas de Tornado Cash par exemple qui utilise les ZKPs.

- **Les identités décentralisées :** Là où les protocoles ZKP montrent tout leur potentiel, c'est dans le domaine des identités décentralisées. Une des grandes problématiques liées à ces identités est la capacité à choisir et restreindre les informations partagées. Ce qui a conduit à l'avènement du zk-KYC.
- **Systèmes de votes sécurisés :** Les ZKP peuvent permettre aux utilisateurs de prouver leur droit de vote et le nombre de voix dont ils disposent sans révéler leur historique ou leurs préférences de vote.

1.4.4.2 Les zk-SNARKs

zk-SNARK (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge) représentent une famille d'algorithmes permettant de démontrer l'exécution d'un calcul sans révéler de données sensibles. Ils facilitent les calculs confidentiels, l'évolutivité et les services sans confiance en permettant la vérification de calculs importants avec une interaction minimale entre les parties. Les principales propriétés sont la concision, la non-interactivité et l'absence de connaissance, qui garantissent des processus de preuve efficaces et sûrs. La procédure d'installation de confiance, une étape cruciale dans la mise en œuvre des zk-SNARK, génère des paramètres initiaux à utiliser dans les opérations cryptographiques, ce qui permet de se prémunir contre les fausses preuves. Dans l'ensemble, les zk-SNARKs offrent une solution robuste pour les calculs préservant la vie privée et les interactions sans confiance, étayée par une gestion minutieuse du processus d'installation [29].

1.4.4.3 Les arbres de Merkle

Les arbres de Merkle, également appelés arbres de hachage, sont un composant indispensable de la technologie blockchain, assurant la vérification sécurisée et efficace des données. Ils ont été développés par Ralph Merkle dans les années 1980. [17]

Ils sont utilisés pour vérifier et garantir l'intégrité des données dans divers systèmes. C'est un arbre binaire dans lequel chaque feuille représente un bloc de données, et chaque nœud interne contient le hachage cryptographique de la concaténation de ses enfants. La racine de l'arbre, appelée Merkle Root, reflète donc l'ensemble de l'arbre, par conséquent l'ensemble des données.

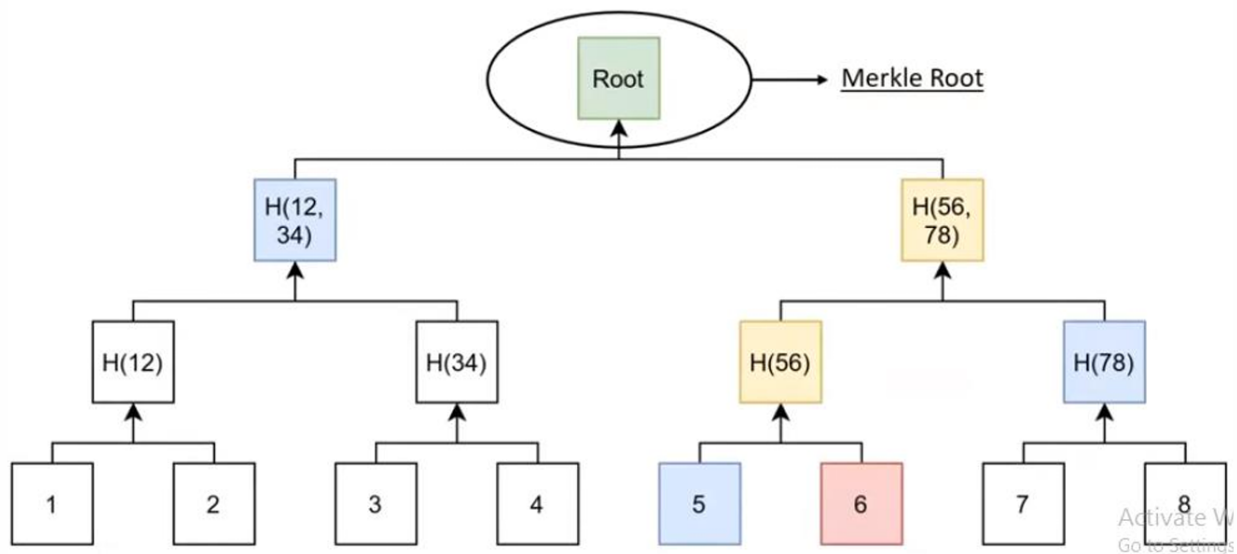


FIGURE 1.6 – Structure d'un arbre de Merkle

Les arbres de Merkle sont essentiels dans divers domaines comme les blockchains, où ils garantissent l'intégrité des données en liant chaque bloc au précédent par un hash. Ils permettent aussi la déduplication efficace de données en identifiant les doublons et servent à vérifier l'intégrité des fichiers en comparant les hachages, offrant ainsi une structure fiable et sécurisée pour gérer de grandes quantités de données [16].

Sur la blockchain Bitcoin par exemple, les arbres de Merkle sont utilisés pour stocker efficacement chaque transaction minée sur le réseau Bitcoin. Toutes les transactions par bloc sont organisées dans un grand arbre de Merkle. Ce bloc, une fois validé, est immuable et le hash de ce dernier est le Merkle root de cet arbre. Ce processus rend efficace le stockage sur la blockchain : on a besoin de valider qu'une seule donnée au lieu de milliers de transactions par bloc, ce qui rend rapide la vérification.

Les arbres de Merkle présentent plusieurs avantages, notamment :

- La vérification rapide de l'intégrité des données : Ils réduisent considérablement la mémoire nécessaire pour vérifier que les données ont conservé leur intégrité et n'ont pas été modifiées.
- Le Stockage : ils ne stockent que les hash au lieu des données complètes, ce qui nécessite moins de données à diffuser sur le réseau blockchain pour vérifier les données et les transactions.
- Dans le développement blockchain, elle favorise l'économie des frais de Gas,

Un des plus grands avantages de l'utilisation de l'arbre de Merkle est qu'il permet de prouver efficacement que certaines données existent dans la construction du hachage racine ; en d'autres termes, il permet d'effectuer des preuves Merkle. **Une preuve Merkle** confirme des transactions spécifiques représentées par un hachage de feuille ou de branche dans une racine de hachage Merkle.

Lors de la vérification des données à l'aide d'un arbre de Merkle, il existe un prouveur et un vérificateur :

- Le prouveur effectue tous les calculs pour créer la racine de Merkle (Le Hash)

- Le vérificateur n'a pas besoin de connaître toutes les valeurs pour savoir avec certitude qu'une valeur est présente dans l'arbre.

Ainsi, si quelqu'un a besoin de prouver qu'une transaction a existé à un moment donné dans la blockchain, il lui suffit de fournir une preuve Merkle. L'utilisateur a donc besoin de la racine de l'arbre de Merkle et d'un chemin de hachages (**Merkle path**) pour prouver que la transaction est bien incluse dans le bloc sans avoir à vérifier tous les autres éléments.

1.4.5 Etude de cas de projet existant : Le Fléau Tornado Cash

1.4.5.1 Introduction

Tornado Cash est un protocole de mixage de cryptomonnaies décentralisé et open source créé en 2019 et hébergé sur la blockchain Ethereum. Il vise à renforcer la confidentialité et l'anonymat des transactions. Il permet aux utilisateurs de déposer des cryptomonnaies avec une adresse et de les retirer avec une autre adresse sans créer de lien traçable entre ces deux adresses. Tornado Cash était probablement le mixeur de cryptomonnaies avec contrat intelligent le plus emblématique et le plus sûr de la blockchain Ethereum, ce qui a contribué à sa popularité auprès des utilisateurs.

Entre 2022 et 2023, plusieurs développeurs du projet ont été arrêtés et accusés d'avoir contribué aux blanchiments d'argent sur la plateforme. Mais le 26 novembre 2024, une Cour d'appel fédérale américaine a renversé la décision de justice qui avait placé le service de mixage de cryptomonnaies Tornado Cash sous les sanctions de l'OFAC en jugeant que les sanctions de l'OFAC contre Tornado Cash étaient illégales [25].

1.4.5.2 Fonctionnement

Tornado Cash utilise une stratégie d'anonymisation basée sur le principe de mélange, similaire à celle de cryptomonnaies axées sur la confidentialité comme Monero. Concrètement, plusieurs utilisateurs envoient leurs fonds à un contrat intelligent commun, où les dépôts sont mélangés pour masquer leur origine. Lorsqu'un utilisateur décide de retirer ses fonds, les liens entre les adresses de dépôt et de retrait sont rompus, rendant impossible l'association entre le déposant et le destinataire final.

Il est important de comprendre que lorsqu'on fait le dépôt sur le smart contract de Tornado Cash, c'est entièrement public. Lorsqu'on fait le retrait de l'argent, c'est également public. Ce qui n'est pas public, c'est que les deux adresses concernées sont associées l'une à l'autre (à condition qu'il y ait suffisamment d'autres déposants et retirants sur ce smart contract).

1.4.5.3 Le processus de dépôt

Le processus de dépôt dans Tornado Cash est conçu pour garantir l'anonymat des utilisateurs grâce à des mécanismes de preuve à divulgation nulle de connaissance (ZKPs). Voici les étapes du processus de dépôt :

- **Génération de notes secrètes** : Lorsqu'un utilisateur veut faire un dépôt sur Tornado Cash, une note secrète est générée. Cette note contient un secret et un nullifier. Ils sont concaténés et hashés pour créer un commitment cryptographique, utilisé comme identifiant pour le dépôt. Il n'existe aucun lien entre le commitment et l'adresse de dépôt de l'utilisateur.

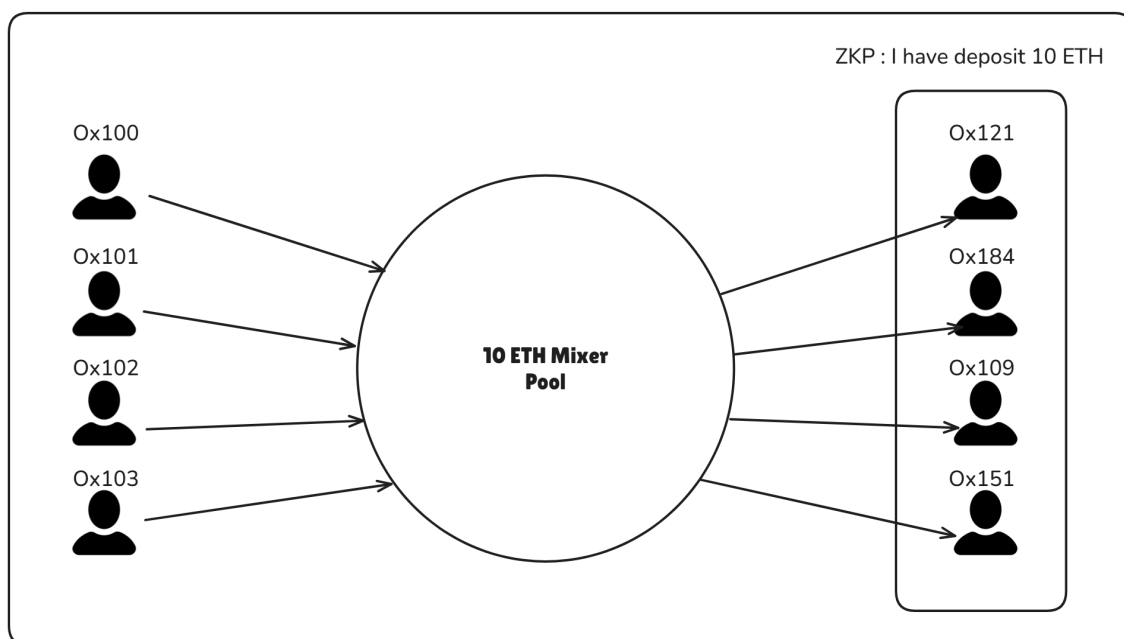


FIGURE 1.7 – Processus de mélange du mixeur

- **Transfert des fonds au contrat intelligent** : L'utilisateur transfère ensuite une certaine quantité de cryptomonnaie (correspondant au montant de dépôt exigé sur ce smart contract) au contrat intelligent Tornado Cash en y incluant le commitment généré auparavant. Ce dernier est enregistré dans un Merkle Tree géré par le contrat.
- **Construction d'un Merkle Tree** : Le Merkle Tree permet de stocker de façon efficace les commitments tout en assurant la confidentialité et l'intégrité des données. Chaque nouveau dépôt ajoute une feuille (leaf) au Merkle Tree. Le Merkle Tree est donc composé des commitments de chaque dépôt fait sur le smart contract.

1.4.5.4 Le processus de retrait

Le processus de retrait sur Tornado Cash est conçu pour s'assurer que les fonds retirés proviennent bien des dépôts précédents et sans révéler de lien entre ces deux actions. Les étapes de ce processus sont :

- **Génération de la preuve cryptographique** : Lorsqu'un utilisateur veut retirer ses fonds de Tornado Cash, il utilise les informations contenues dans la note secrète générée lors du dépôt (composée du nullifier et du secret) pour produire une preuve à divulgation nulle de connaissance. Cette preuve est générée localement par l'utilisateur, souvent via [Circom](#). Elle prouve que l'utilisateur possède une note valide enregistrée dans le Merkle Tree, sans révéler quelle feuille (leaf) correspond au dépôt. L'utilisateur soumet cette preuve au contrat intelligent Tornado Cash en même temps qu'une adresse de retrait.
- **Validation de la preuve** : Le contrat intelligent Tornado Cash valide la preuve ZKP grâce à des algorithmes tels que Groth16 [11]. Cette validation permet de confirmer que la note est valide et

correspond à un engagement (commitment) stocké dans le Merkle Tree et que les fonds associés à ce commitment n'ont pas encore été retirés. Si la preuve échoue, la transaction est annulée.

- **Transfert des fonds :** Le contrat intelligent transfère ensuite le montant demandé (déposé initialement) à l'adresse de retrait spécifiée. Comme cette nouvelle adresse n'a aucun lien direct avec l'adresse utilisée pour le dépôt, l'anonymat de l'utilisateur est conservé.

Lors du retrait des fonds, l'utilisateur peut être confronté à un problème de frais de transaction. Si l'adresse de retrait ne dispose pas de fonds pour payer les frais de gas, il peut choisir d'utiliser un relayeur. Ce dernier permettra d'effectuer la transaction en échange de la réception d'une partie du retrait de Tornado Cash.

Conclusion

Dans ce chapitre, nous avons parlé de l'aspect théorique du projet, des notions sur le fonctionnement de la blockchain, des avancées technologiques dans le domaine de la sécurité et de la confidentialité, des transactions sur la blockchain Ethereum et des conformités réglementaires. Les solutions telles que les preuves à zéro connaissance, les arbres de Merkle, et les mixeurs de cryptomonnaies offrent des opportunités prometteuses pour garantir la confidentialité des utilisateurs tout en maintenant l'intégrité des systèmes. Cependant, ces technologies doivent encore relever des défis, notamment en matière de conformité et de régulation.

Dans le chapitre suivant, nous avons abordé les matériels de travail ainsi que les méthodes utilisées pour la mise en place de la solution de confidentialité.

Matériels et Méthodes

Introduction

Ce chapitre présente les matériels et méthodes utilisées pour concevoir et développer le prototype d'anonymisation des transactions financières sur Ethereum, intégrant un mécanisme de zk-KYC. Dans ce chapitre, nous avons détaillé les outils logiciels utilisés pour le développement, avant de présenter les méthodologies adoptées pour concevoir et développer les fonctionnalités clés de notre système. L'objectif est de présenter une architecture technique combinant des mécanismes cryptographiques (zk-SNARKs, Merkle Tree), des contrats intelligents et des interfaces web décentralisées conformes aux exigences définies dans le cadre du mémoire.

2.1 Matériels

Les technologies et outils sélectionnés ont été choisis pour leur adéquation avec les exigences du projet :

2.1.1 Environnement de développement blockchain

2.1.1.1 Langage Solidity

Le langage de programmation choisi pour le développement des smart contracts au cœur du système zk-KYC et du Mixer est Solidity. C'est un langage de programmation de haut niveau, statiquement typé et orienté objet, conçu spécifiquement pour la mise en œuvre de contrats intelligents sur la blockchain Ethereum. Il est inspiré par des langages tels que C++, Python et JavaScript, utilise une syntaxe en accolades et est compilé pour s'exécuter sur l'Ethereum Virtual Machine (EVM) [24] .

Avec l'adoption massive par la communauté Ethereum, il est devenu le langage principal pour le développement de contrats intelligents sur Ethereum et d'autres blockchains compatibles avec l'EVM.

2.1.1.2 Truffle Suite et Ganache

La Truffle Suite est un ensemble d'outils open-source qui offre une boîte à outils complète permettant de faciliter le processus de développement de smart contracts, en passant par leur test et leur déploiement. La Truffle Suite est conçue pour offrir une expérience de développement structurée et cohérente. En combinant des fonctionnalités telles que la compilation, le déploiement automatique, les tests unitaires et une intégration facile avec les réseaux Ethereum, elle permet de créer et de déployer des applications décentralisées de manière simple et efficace.

Ganache est un simulateur de blockchain Ethereum qui rend le développement d'applications Ethereum plus rapide. Il comprend toutes les fonctions et fonctionnalités RPC populaires, telles que les événements, qui seront utiles dans la suite.

2.1.1.3 Circom et SnarkJS

Circom (Circuit Compiler) est un langage de définition de circuits développé pour créer et tester des preuves à divulgation nulle de connaissance. Il produit la représentation du circuit sous forme de contraintes et tout ce qui est nécessaire pour calculer les différentes preuves ZK.

SnarkJS est une bibliothèque JavaScript qui compile les circuits Circom, génère les clés cryptographiques (clé de vérification et clé de proving), et produit les preuves zk-SNARKs exploitables sur Ethereum.

2.1.1.4 Interface utilisateur avec Nuxt.js

Nuxt.js est un framework open-source basé sur Vue.js, conçu pour faciliter le développement d'applications web modernes, réactives, performantes et sécurisées. Il offre une architecture préétablie et des conventions qui simplifient la configuration initiale, permettant le développement rapide d'applications web.

2.1.1.5 Web3.js

Web3.js est une bibliothèque JavaScript open-source créée par la fondation Ethereum pour permettre aux utilisateurs finaux d'interagir avec un nœud Ethereum local ou distant via HTTP, IPC ou WebSocket [26]. La bibliothèque web3.js fait partie intégrante des applications décentralisées (Dapps) car elle permet de relier le backend au frontend.

Elle fournit des fonctions qui permettent au frontend de communiquer avec un nœud Ethereum via le protocole JavaScript Object Notation-Remote Procedure Call (JSON-RPC).

2.1.1.6 OpenZeppelin

OpenZeppelin est une bibliothèque open-source de contrats intelligents audités et standardisés, largement adoptée dans l'écosystème Ethereum pour sa fiabilité et son alignement avec les meilleures pratiques de sécurité. Son intégration dans le projet vise à renforcer la robustesse des smart contracts zk-KYC et Mixer, tout en accélérant le développement grâce à des composants modulaires.

2.2 Architecture du système

L'architecture du système mis en place repose sur une combinaison de smart contracts sur Ethereum, de preuves Zero-Knowledge (ZK) et d'une interface web conviviale qui permet aux utilisateurs de faire une vérification KYC pour whitelister leurs adresses afin d'effectuer des dépôts et des retraits sur le mixeur.

Le système est conçu pour permettre aux utilisateurs de prouver leur identité de manière confidentielle avant d'effectuer des transactions anonymes sur la blockchain Ethereum. La figure ci-dessous représente l'architecture globale du système :

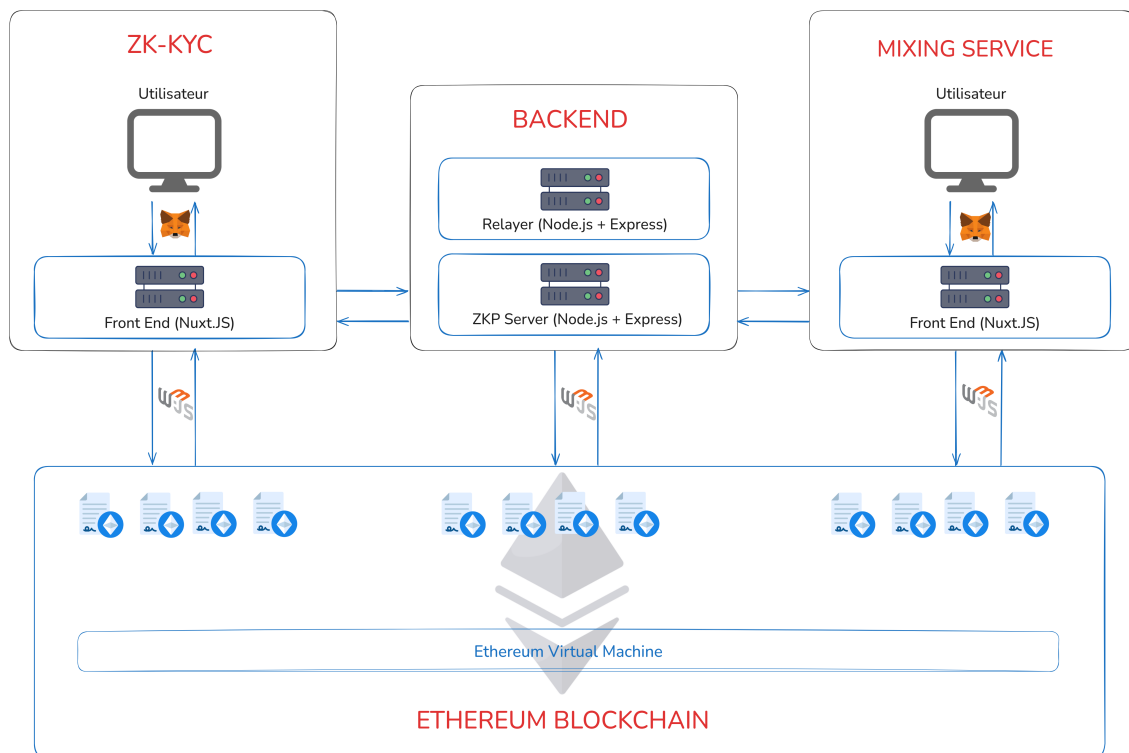


FIGURE 2.1 – Architecture globale du système

Il repose essentiellement sur les composants suivants :

- **Smart Contract zk-KYC** : Le contrat intelligent qui assure la vérification d'identité basé sur des preuves ZK, permettant de générer une preuve KYC validé par une autorité de confiance.
- **Smart Contract Mixer** : Le contrat intelligent qui gère les dépôts et retraits d'ETH en appliquant des mécanismes de confidentialité basés sur les arbres de Merkle Tree et les relayeurs de transactions .
- **Frontend Web KYC** : Une interface utilisateur décentralisée, développé en Nuxt.js permettant aux utilisateurs de faire une vérification d'identité et whitelister leurs adresses Ethereum.
- **Frontend Web Mixer (Phantom ETH)** : Une interface utilisateur décentralisée, développé en Nuxt.js permettant aux utilisateurs d'interagir avec le smart contract Mixer pour mélanger leurs cryptomonnaies.

- **Backend API** : Un serveur (Node.js + Express) utilisé pour faciliter la génération de preuves pour le KYC et pour les retraits sur le mixer.
- **Relayeur de transaction** : Un serveur (Node.js + Express) intermédiaire qui exécute les transactions de retrait pour le compte de l'utilisateur, garantissant ainsi son anonymat en masquant l'adresse de la requête de retrait et en payant les frais de gaz.

Conclusion

Ce chapitre a présenté l'ensemble des outils et méthodologies utilisés pour développer la solution d'anonymisation des transactions financières sur Ethereum. En résumé, l'architecture du système repose sur quatre éléments clés : le client, le smart contract KYC, le smart contract Mixer et le relayer API. Ethereum a été choisi comme blockchain et Solidity pour l'écriture des smart contracts et des outils et frameworks tels que Circom pour les preuves ZK, Truffle pour le développement des smart contracts, Ganache pour le test en local et Nuxt.js pour l'interface utilisateur.

Modélisation et Développement

Introduction

Ce chapitre détaille les étapes concrètes de conception et de développement du prototype d'anonymisation des transactions financières sur Ethereum. Il s'articule autour de la modélisation des composants clés identifiés dans le chapitre précédent (zk-KYC, Mixer, interfaces, backend, relayeur) et de leur intégration dans un système cohérent. L'objectif est de démontrer comment les choix technologiques et méthodologiques répondent aux défis de confidentialité, de sécurité et de conformité réglementaire posés par les blockchains publiques.

3.1 Modélisation du Système

3.1.1 Flux de fonctionnement du système

Le flux de fonctionnement du système se déroule en plusieurs étapes clés, garantissant à la fois la conformité réglementaire et la préservation de la confidentialité des utilisateurs. Ce flux se présente en plusieurs étapes, à savoir :

- **Phase 1 - Vérification KYC :** L'utilisateur soumet ses informations d'identification à une autorité centrale de contrôle, qui génère une preuve à divulgation nulle de connaissance basé sur ses données. Cette preuve est ensuite utilisée pour whitelister ses adresses dans le smart contract KYC sur Ethereum.
- **Phase 2 - Dépôt dans le mixer :** L'utilisateur dépose ses fonds sur un des smart contracts (pool) du Mixer. Le système lui génère un commitment unique et l'ajoute au Merkle Tree du smart contract via pour enregistrer ce dépôt.
- **Phase 3 - Génération et soumission de la preuve ZK :** Lors du retrait, l'utilisateur fournit son commitment et le système génère une preuve ZK démontrant qu'il possède un dépôt valide dans le mixer sans révéler son identité.
- **Phase 4 - Utilisation du relayeur :** L'utilisateur envoie la preuve à un Relayeur API. Ce dernier signe la transaction de retrait et paie les frais de gas en échange d'un petit frais, garantissant

ainsi que l'adresse de retrait ne puisse être liée à l'utilisateur.

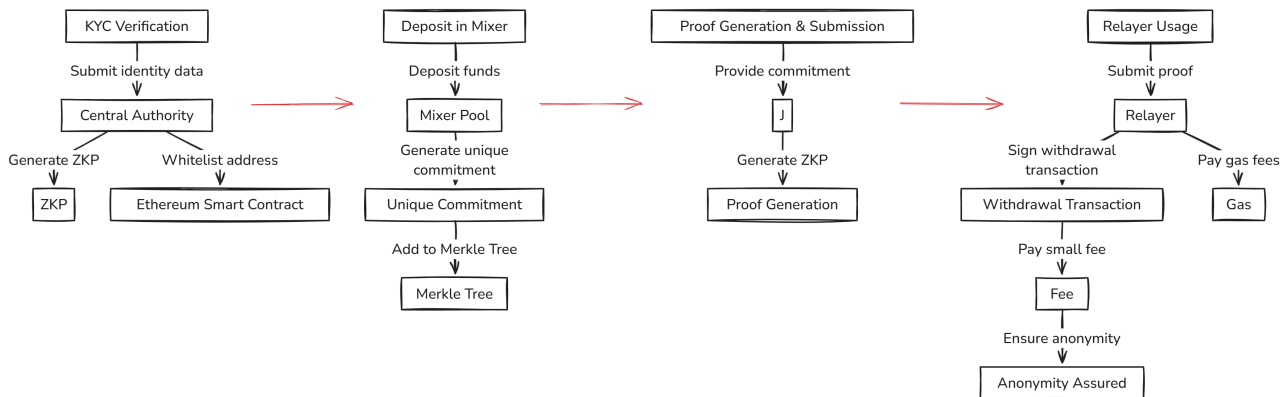


FIGURE 3.1 – Flux de fonctionnement du système

Ce flux assure que les utilisateurs peuvent prouver leur identité et utiliser le mixeur de transactions tout en préservant la confidentialité de leurs données personnelles et de leurs transactions, grâce à l'utilisation de preuves à divulgation nulle de connaissance.

3.1.2 Diagramme de composant du système

Le système se compose de trois couches principales interconnectées, assurant une interaction fluide et sécurisée entre les utilisateurs et la blockchain.

3.1.2.1 La couche Frontend

regroupe l'interface KYC, qui permet la vérification d'identité via une application web, et l'interface Phantom ETH, le mixeur de cryptomonnaies. Le frontend est développé avec Nuxt.js et communique avec la blockchain via Web3.js.

La couche blockchain intègre deux smart contracts essentiels : le smart contract zk-KYC, qui gère les vérifications d'identité et le whitelisting, et le smart contract Mixer, responsable des dépôts et retraits anonymes. Le Mixer utilise un arbre de Merkle pour stocker les engagements cryptographiques (commitments), le tout implémenté en Solidity avec l'appui de la bibliothèque d'OpenZeppelin.

Enfin, la couche Backend, basée sur Node.js et Express, comprend une API Backend pour la gestion des requêtes, un générateur de preuves ZK (utilisant Circom et SnarkJS) ainsi qu'un service Relayeur qui exécute anonymement les transactions de retrait en masquant l'origine des fonds et en prenant en charge les frais de gas.

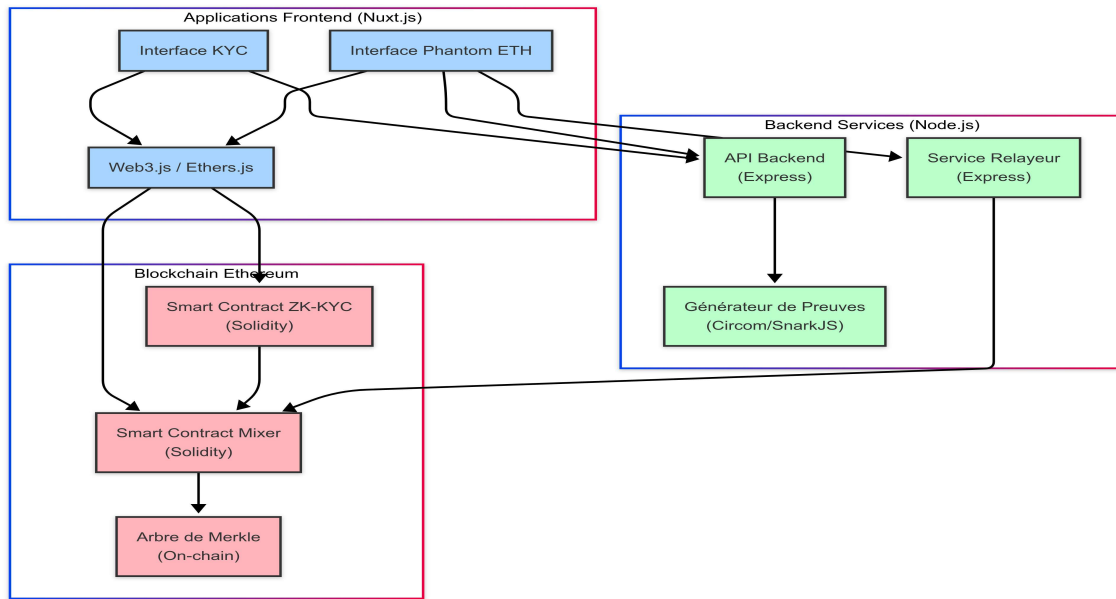


FIGURE 3.2 – Diagramme de composant du système

3.1.3 Diagramme de flux de données

3.1.3.1 Le processus de vérification KYC

Le système implémente un mécanisme de whitelisting d'adresses' Ethereum flexible et sécurisé qui se déploie en plusieurs phases. Tout d'abord, lors de la vérification KYC initiale, l'utilisateur soumet ses informations personnelles, que le système vérifie pour générer ensuite une preuve à divulgation nulle qui valide son identité sans exposer ses données sensibles. //

Ensuite, l'utilisateur utilise la preuve générée pour inscrire sa première adresse Ethereum, le smart contract KYC vérifie la validité de la preuve et enregistre l'adresse si celle-ci est correcte. Enfin, lors du whitelisting en cascade, les adresses déjà whitelistées obtiennent le pouvoir de valider de nouvelles adresses, chaque demande devant être signée par une adresse déjà whitelistée.

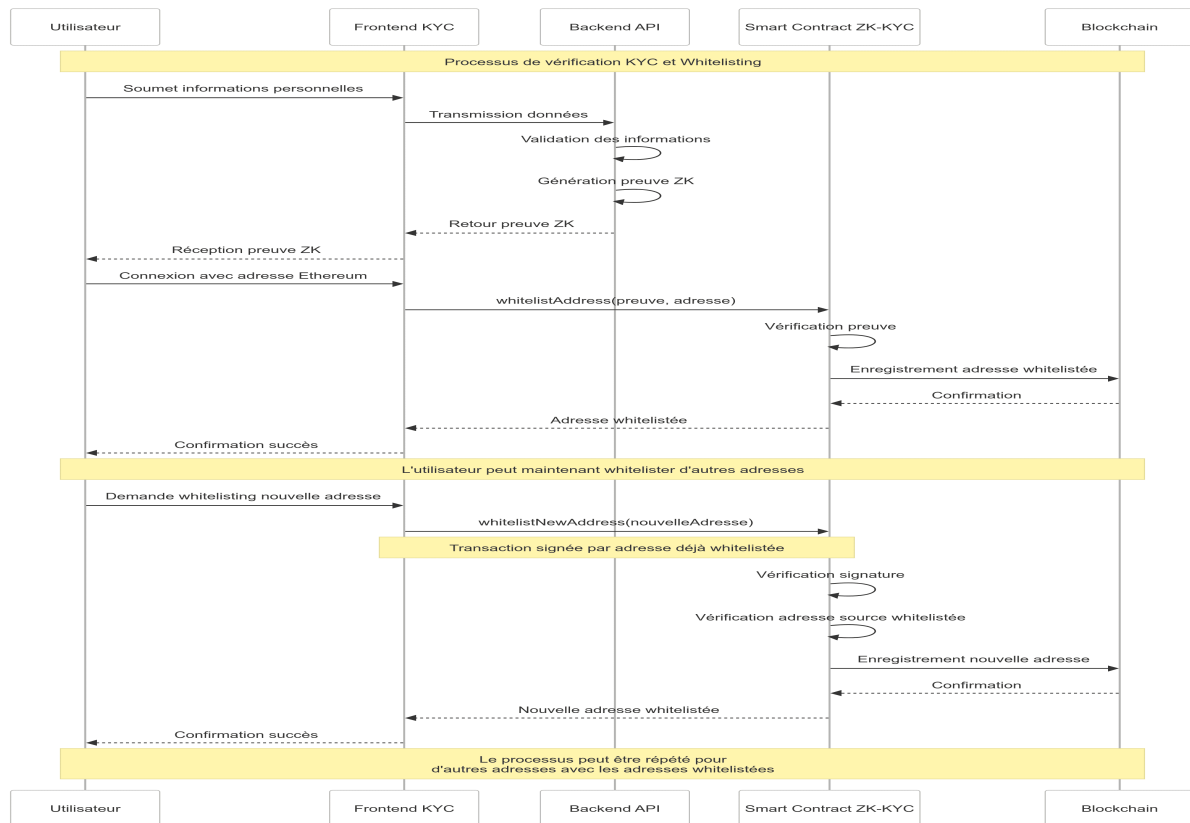


FIGURE 3.3 – Processus de vérification d'identité et de whiteliste

Cette approche permet une vérification KYC unique pour plusieurs adresses, une gestion flexible des adresses multiples, tout en maintenant la sécurité et en préservant la confidentialité des informations personnelles de l'utilisateur.

3.1.3.2 Le processus de dépôt

Dans le cadre du système, le processus de dépôt représente la première phase d'anonymisation des transactions. Lorsqu'un utilisateur initie un dépôt via l'interface du mixeur, le smart contract dédié reçoit les fonds et génère immédiatement un commitment unique en appliquant la fonction de hachage [18] sur les données associées au dépôt.

Ce commitment est ensuite intégré dans l'arbre de Merkle, qui se met à jour en temps réel et dont l'historique des racines est enregistré sur la blockchain Ethereum. Voici ci-dessous le diagramme de flux du processus de dépôt.

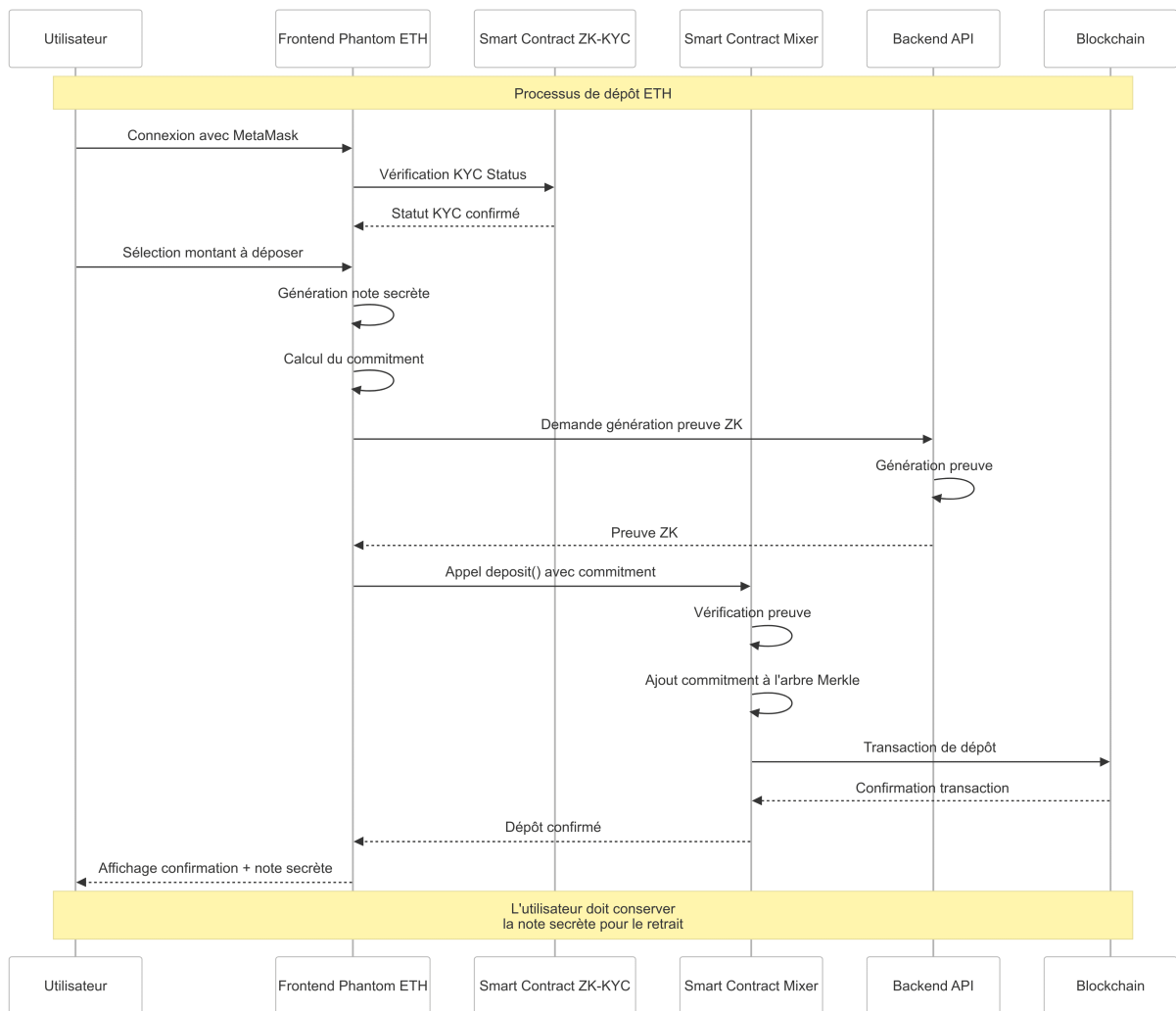


FIGURE 3.4 – Processus de dépôt des fonds sur le mixer

3.1.3.3 Le processus de retrait

Dans le système, le processus de retrait intervient lorsque l'utilisateur souhaite récupérer les fonds déposés sur le smart contract de manière anonyme. Pour ce faire, il génère une preuve à divulgation nulle de connaissance (ZKP) attestant qu'il possède un dépôt valide enregistré dans l'arbre de Merkle. La preuve est ensuite soumise avec le chemin de Merkle et un nullifier.

Le smart contract de retrait vérifie alors la validité de la preuve en comparant le chemin fourni à la racine stockée dans l'arbre et en s'assurant que le nullifier n'a pas déjà été utilisé. Une fois ces vérifications effectuées, la transaction de retrait est autorisée et, grâce à l'intervention d'un relayeur qui signe la transaction et prend en charge les frais de gas, les fonds sont transférés à une nouvelle adresse de l'utilisateur.

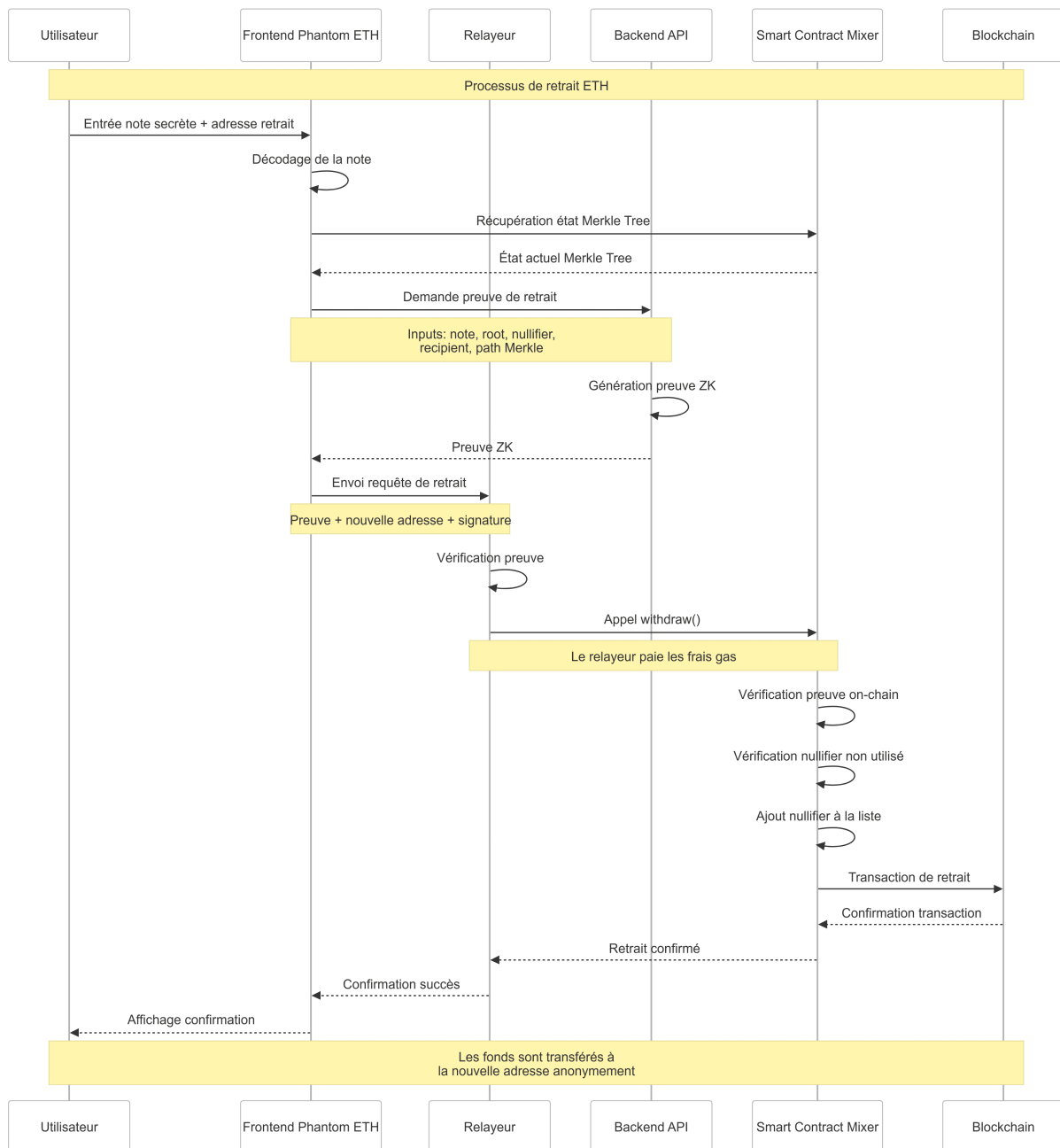


FIGURE 3.5 – Processus de retrait des fonds du mixer

Ce processus complexe garantit que :

- Aucun observateur ne peut lier le retrait au dépôt original,
- Les fonds ne peuvent être retirés qu'une seule fois,
- L'utilisateur légitime est le seul à pouvoir effectuer le retrait,
- L'anonymat est préservé même en cas de compromission du relayeur,

3.2 Développement des Circuits ZK avec Circom

3.2.1 Conception des Circuits

Les circuits zk-SNARKs pour les preuves à connaissance nulle reposent sur des circuits arithmétiques. Un circuit arithmétique est un système d'équations utilisant uniquement l'addition, la multiplication et l'égalité. Comme un circuit booléen, il vérifie qu'un ensemble d'entrées proposé est valide, mais ne calcule pas de solution. Ces circuits définissent des relations entre des entrées publiques (public inputs) et privées (private inputs), permettant de prouver une déclaration sans révéler les données sous-jacentes.

Ces circuits seront la fondation du système ZK implémenté.

3.2.1.1 Le circuit `kyc_verifier.circom`

Ce circuit Circom permet de vérifier la conformité KYC d'un utilisateur en validant que des données d'identité fournies correspondent à un hash préenregistré par une autorité de confiance. Il garantit la confidentialité en ne révélant aucune donnée personnelle sur la blockchain, uniquement la preuve cryptographique de leur validité.

Ce circuit prend en entrée plusieurs ensembles de données privées (le nom, la date de naissance, le pays et le numéro d'identité) et applique, pour chacun, un hachage via le composant `Poseidon` de `circomlib`, une fonction de hachage optimisée pour les zk-SNARKs. Ces hachages individuels sont ensuite combinés à l'aide d'un autre composant `Poseidon` pour produire un hash final représentatif de l'identité vérifiée.

Pour garantir l'intégrité du processus, le circuit utilise un composant de comparaison, `IsEqual`, qui vérifie que le hash final obtenu correspond au hash attendu `expectedHash` fourni en entrée publique.

```
component areEqual = IsEqual();
areEqual.in[0] <== finalHasher.out;
areEqual.in[1] <== expectedHash;
isValid <== areEqual.out;
```

Si la comparaison est concluante, le signal de sortie renvoie une valeur indiquant que l'identité est validée.

3.2.1.2 Le circuit `merkle_tree.circom`

Le circuit `merkle_tree.circom` permet de vérifier qu'un élément (le `leaf`) appartient bien à un Merkle Tree donné en comparant le hash final calculé à partir du chemin Merkle avec la racine attendue (`root`). Ce circuit utilise deux templates importants :

- `HashLeftRight` : il combine deux valeurs en appliquant la fonction de hachage `MiMCSponge`
- `DualMux` : il sélectionne conditionnellement entre deux entrées selon un bit (0 ou 1).

Dans le template `MerkleTreeChecker`, pour chaque niveau de l'arbre, le circuit utilise `DualMux` pour choisir entre la valeur courante (soit le `leaf` initial ou le résultat du hachage du niveau précédent) et l'élément de preuve (`pathElements[i]`), en fonction de la valeur binaire fournie dans `pathIndices[i]` (permet de savoir si `pathElement` donné est sur le côté gauche ou droit du chemin Merkle).

```
for (var i = 0; i < levels; i++) {
  selectors[i] = DualMux();
  selectors[i].in[0] <== i == 0 ? leaf : hashers[i - 1].hash;
  selectors[i].in[1] <== pathElements[i];
  selectors[i].s <== pathIndices[i];

  hashers[i] = HashLeftRight();
  hashers[i].left <== selectors[i].out[0];
  hashers[i].right <== selectors[i].out[1];
}

root == hashers[levels - 1].hash;
```

Le résultat de cette sélection est ensuite haché via `HashLeftRight`, et ce processus se répète sur tous les niveaux, jusqu'à obtenir un hash final qui est comparé à la racine attendue `root`. Ce qui permet de prouver ainsi que le `leaf` appartient à l'arbre.

3.2.1.3 Le circuit `withdraw.circom`

Le circuit `withdraw.circom` permet le retrait anonyme des fonds déposés dans le mixeur. Le circuit reçoit en entrée publique une racine de Merkle `root`, le montant du dépôt `amount`, le hash du nullifier `nullifierHash` ainsi que le chemin de preuve (**Merkle path**). En entrée privée, le circuit reçoit le `secret` et le nullifier correspondant au dépôt de l'utilisateur. Comme l'arbre Merkle est entièrement public sur la blockchain, n'importe qui peut calculer une preuve Merkle pour n'importe laquelle des feuilles. Il n'est donc pas suffisamment sûr de simplement fournir une preuve ZK et d'avoir une preuve Merkle et une racine pour procéder au retrait, l'utilisateur doit également prouver qu'il connaît la préimage de la feuille.

```
...
component tree = MerkleTreeChecker(levels);
tree.leaf <== hasher.commitment;
tree.root <== root;

for (var i = 0; i < levels; i++) {
  tree.pathElements[i] <== pathElements[i];
  tree.pathIndices[i] <== pathIndices[i];
}
```

Ce circuit utilise `merkle_tree.circom` pour calculer une nouvelle racine Merkle à l'aide des entrées. On vérifie que la racine calculée par le circuit est bien celle reçue en entrée publique `root`.

3.2.2 Compilation des circuits et génération des preuves

Pour obtenir des preuves zk-SNARKs exploitables sur Ethereum, on passe par plusieurs étapes à savoir :

- **Compilation du circuit :** Dans cette étape, on traduit les circuits en artefacts exploitables pour la génération de preuves.

```
$ circom kyc_verifier.circom --rlcs --wasm --c --json
$ circom withdraw.circom --rlcs --wasm --c --json
```

Cette commande génère un fichier de contraintes `.rlcs` pour SnarkJS et un fichier `.wasm` pour générer le témoin (witness).

- **Configuration de Confiance (Trusted Setup) :** Dans cette étape, on génère les clés cryptographiques nécessaires aux preuves.

```
$ snarkjs powersoftau new bn128 16 pot16_0000.ptau -v
$ snarkjs powersoftau contribute pot16_0000.ptau pot16_0001.ptau
--name="Premiere contribution" -v
$ snarkjs powersoftau prepare phase2 pot16_0001.ptau
pot16_final.ptau -v
$ snarkjs groth16 setup verifier.rlcs pot16_final.ptau
verifier_0000.zkey
$ snarkjs zkey contribute verifier_0000.zkey
verifier_final.zkey --name="1st Contributor" -v
```

- **Génération des fichiers de vérification et du contrat vérificateur solidity :** Enfin, on génère les fichiers de vérification `.json` et `.zkey`, puis le contrat vérificateur solidity.

```
$ snarkjs zkey export solidityverifier verifier_final.zkey
../../contracts/Verifier.sol
$ snarkjs zkey export verificationkey verifier_final.zkey
verification_key.json
```

Cette preuve est ensuite vérifiable à la fois hors chaîne via SnarkJS et sur la blockchain via le Verifier généré.

3.3 Modélisation des Smart Contracts

3.3.1 Smart Contract Zero Knowledge KYC

Le smart contract zk-KYC est l'élément le plus important du système de vérification d'identité décentralisé. Il implémente un mécanisme permettant de valider l'identité des utilisateurs tout en préservant leur confidentialité, grâce au Vérificateur de preuve obtenu lors de la génération des preuves. Voici ci-dessous la structure globale des variables du contrat :

```

...
import "@openzeppelin/contracts/access/Ownable.sol";

contract KYCRegistry is Ownable {

    IKYCVerifier public verifier;
    mapping(address => bool) public whitelistedAddresses;
    mapping(uint256 => bool) public kycVerified;
    mapping(uint256 => address[]) public kycAddresses;

    event KYCVerified(uint256 indexed expectedHash, address indexed firstAddress);
    event AddressWhitelisted(address indexed userAddress, uint256 indexed expectedHash);

    constructor(address _verifierAddress) Ownable(msg.sender) {}
}

```

Le contrat de vérification KYCRegistry implémente l'interface du contrat de vérification de preuve KYCVerifier généré précédemment. Lors du déploiement du smart contract, l'adresse du KYCVerifier est mise en paramètre pour permettre plus tard de vérifier les preuves grâce à la fonction `verifyProof` du contrat KYCVerifier.

```

interface IKYCVerifier {
    function verifyProof(
        uint[2] calldata _pA, uint[2][2] calldata _pB, uint[2] calldata _pC, uint[1]
        calldata _pubSignals
    ) external view returns (bool);
}

```

Cette fonction `verifyProof` prend en paramètres les trois points de la preuve Groth16 [11] (`_pA`, `_pB`, `_pC`) qui sont des points sur une courbe elliptique, ainsi que les signaux publics (`_pubSignals`) qui sont les sorties publiques du circuit ZK.

Le contrat KYCRegistry implémente elle-même la fonction `verifyKYC` qui permet la vérification de preuve et le whitelist de la première adresse Ethereum de l'utilisateur :

```

require(!kycVerified[expectedHash], "KYC already verified");
require(userAddress != address(0), "Invalid address");
require(
    !whitelistedAddresses[userAddress],
    "Address already whitelisted"
);

require(
    verifier.verifyProof(_pA, _pB, _pC, _pubSignals),
    "Invalid KYC proof"
);
require(_pubSignals[0] == 1, "Invalid proof result");

kycVerified[expectedHash] = true;
whitelistedAddresses[userAddress] = true;
kycAddresses[expectedHash].push(userAddress);

```

Après avoir whitelisted sa première adresse, l'utilisateur peut whitelister d'autres adresses Ethereum avec la fonction `addWhitelistedAddress` :

```
bool isAuthorized = false;
address[] memory addresses = kycAddresses[expectedHash];
for (uint i = 0; i < addresses.length; i++) {
    if (addresses[i] == msg.sender) {
        isAuthorized = true;
        break;
    }
}
require(isAuthorized, "Not authorized to add addresses for this KYC");

whitelistedAddresses[newAddress] = true;
kycAddresses[expectedHash].push(newAddress);
emit AddressWhitelisted(newAddress, expectedHash);
}
```

Enfin, la fonction `isWhitelisted` prend en paramètre une adresse et vérifie si elle est whitelisted. Le contrat du mixeur utilise cette fonction pour vérifier si les adresses impliquées dans le processus de mixage sont whitelistedes.

```
function isWhitelisted(address _address) external view returns (bool) {
    return whitelistedAddresses[_address];
}
```

3.3.2 Smart Contract MerkleTreeWithHistory

Le contrat `MerkleTreeWithHistory` est une implémentation d'un arbre de Merkle dynamique en solidity. La particularité de notre arbre est qu'il dispose d'un historique des racines, essentiel lorsqu'un utilisateur veut retirer ses fonds. Ce qui permet de prouver l'appartenance d'un dépôt à l'arbre sans révéler le commitment, tout en gardant une trace des états passés. Lors de la construction, l'arbre est initialisé avec des ZEROS (0) et une profondeur fixe de 20. Voici ci-dessous les fonctions principales de ce contrat :

```
function insert(bytes32 _leaf) public returns (uint32 index);
function isKnownRoot(uint256 _root) public view returns (bool);
function hashpair(uint256 _left, uint256 _right) returns (uint256 _hash);
```

La fonction `insert` permet d'insérer une feuille dans l'arbre. Lorsqu'un utilisateur effectue un dépôt, son engagement (commitment) est ajouté comme une nouvelle feuille de l'arbre. Le remplissage de l'arbre s'effectue séquentiellement de gauche à droite. Les index pairs sont toujours des feuilles de gauche et les impairs, les feuilles de droite.

```
function insert(uint256 _leaf) public returns (uint32) {
    ....

    for (uint32 i = 0; i < TREE_HEIGHT; i++) {
        if (currentLevelHash % 2 == 0) {
```



```

        left = currentLevelValue;
        right = zeros[i];
        filledSubtrees[i] = currentLevelValue;
    } else {
        left = filledSubtrees[i];
        right = currentLevelValue;
    }
    currentLevelValue = hashPair(left, right);
    currentLevelHash = currentLevelHash / 2;
}

currentRootIndex = (currentRootIndex + 1) % 100;
roots[currentRootIndex] = currentLevelValue;
nextIndex++;

....
}

```

La fonction `isKnownRoot` vérifie si une racine donnée correspond à un état valide de l'arbre de Merkle. Cette vérification est essentielle lors des retraits, car elle permet de s'assurer que la preuve fournie par l'utilisateur correspond bien à un engagement qui existait au moment du dépôt. Il est crucial dans le cas de notre système de maintenir un historique des racines valides, car la racine de l'arbre change à chaque nouveau dépôt.

```

function isKnownRoot(uint256 _root) public view returns (bool) {
    if (_root == 0) return false;
    uint32 i = currentRootIndex;
    do {
        if (_root == roots[i]) return true;
        if (i == 0) i = 100;
        i--;
    } while (i != currentRootIndex);
    return false;
}

```

Enfin, la fonction `hashPair` calcule le hash de deux nœuds frères dans l'arbre. Elle prend en paramètres les valeurs des nœuds gauche et droit, et retourne leur hash combiné avec la fonction de hashage [keccak256](#).

3.3.3 Smart Contract Mixer

Le contrat Mixer permet aux utilisateurs de déposer des ETH et de les retirer ultérieurement sans créer de lien traçable entre les adresses de dépôt et de retrait. Il hérite du smart contract `MerkleTree-WithHistory` et implémente l'interface du contrat de vérification de preuve `Verifier` généré après la compilation du circuit `withdraw.circom`.

Pour vérifier qu'une adresse est déjà whitelisted, il implémente aussi l'interface du contrat `KYCRegistry` et utilise la fonction `isWhitelisted`.

Pour éviter les attaques de réentrée ([Reentrancy attack](#) [23]), ce contrat étend aussi le contrat `ReentrancyGuard` de la librairie `OpenZeppelin`.

Lors du déploiement de ce contrat, on passe l'adresse du KYCVerifier, l'adresse du KYCRegistry et le nombre d'ETH que ce contrat reçoit (`denomination`).

```
...
import "./MerkleTreeWithHistory.sol";

interface IVerifier {
    function verifyProof(
        uint[2] calldata _pA,
        uint[2][2] calldata _pB,
        uint[2] calldata _pC,
        uint[4] calldata _pubSignals
    ) external view returns (bool);
}

interface IKYCRegistry {
    function isWhitelisted(address _address) external view returns (bool);
}
```

La structure de ce contrat s'articule autour de trois points essentiels :

3.3.3.1 La gestion des dépôts :

Le processus de dépôt est simple. L'utilisateur envoie un montant fixe d'ETH (`denomination`) accompagné du `commitment`. Ce `commitment`, généré à partir d'une note secrète, est composé de deux éléments, le **nullifier** et le **secret** que l'utilisateur conserve pour le retrait futur. Le contrat vérifie simplement que l'utilisateur est whitelisted via le contrat zk-KYC et stocke le hash de ce `commitment` dans l'arbre de Merkle.

Un élément aussi important est l'événement émis après le dépôt. Cet événement sera utile plus tard dans la reconstruction de l'arbre de Merkle.

```
function deposit(uint256 _commitment) external payable nonReentrant {
    require(kycRegistry.isWhitelisted(msg.sender), "Your address is not whitelisted. Please complete KYC first.");
    require(msg.value == denomination, "Wrong amount");
    require(_commitment != 0, "Commitment cannot be 0");
    uint32 insertedIndex = insert(_commitment);

    denominationAnonymitySet[denomination]++;
    ...
}
```

La variable `denominationAnonymitySet` permet à l'utilisateur de savoir en temps réel le nombre de dépôts actifs (les dépôts qui n'ont pas encore été retirés) car plus l'`anonymitySet` est élevé, plus les transactions sur le mixer le sont.

3.3.3.2 La gestion des retraits :

Le processus de retrait est l'opération la plus complexe. L'utilisateur doit fournir une preuve ZK démontrant qu'il connaît une note secrète correspondant à un `commitment` existant dans l'arbre de

Merkle, sans révéler quel est le commitment et sa position dans l'arbre. Le nullifier, dérivé du commitment, permet d'empêcher qu'un utilisateur retire plusieurs fois les fonds avec une même note.

```
function withdraw(
    uint[2] calldata _pA, uint[2][2] calldata _pB,
    uint[2] calldata _pC, uint[4] calldata _pubSignals,
    uint256 root, uint256 nullifierHash,
    address payable recipient, address payable relayer, uint256 fee
) external nonReentrant {
    require(kycRegistry.isWhitelisted(recipient), "The withdrawal address is not
whitelisted. Please complete KYC to withelist this address.");
    require(fee <= denomination, "Fee exceeds transfer value");
    require(!nullifierHashes[nullifierHash], "The note has been already spent");
    require(isKnownRoot(root), "Cannot find your merkle root");
    require(recipient != address(0), "Invalid withdraw address");

    require(
        verifier.verifyProof(_pA, _pB, _pC, _pubSignals),
        "Invalid withdraw proof !"
    );
};
```

3.3.3.3 Le schéma de nullification :

Pour prévenir les retraits multiples tout en préservant l'anonymat, le système utilise un mécanisme de **nullifier**. Plutôt que de retirer les feuilles de l'arbre de Merkle, ce qui révélerait quel dépôt est utilisé, chaque feuille est construite à partir de deux valeurs secrètes : un **nullifier** et un **secret**.

La feuille est calculée comme le hash de la concaténation de ces deux valeurs. Lors du retrait, l'utilisateur soumet uniquement le hash du nullifier (`nullifierHash`) ainsi qu'une preuve ZK démontrant qu'il connaît le nullifier et le secret correspondant à une feuille existante dans l'arbre. Le smart contract stocke les `nullifierHash` utilisés pour empêcher leur réutilisation.

```
...
mapping(uint256 => bool) public nullifierHashes;
...
nullifierHashes[nullifierHash] = true;
...
```

Avec cette approche, on ne révèle qu'un seul des deux nombres secrets (via son hash) et il devient donc impossible de déterminer quelle feuille de l'arbre est associée à un dépôt, préservant ainsi l'anonymat de la transaction tout en garantissant qu'un même dépôt ne peut être retiré qu'une seule fois.

3.3.4 Déploiement des smart contracts sur Ganache

Le déploiement des smart contracts a été réalisé sur Ganache, l'environnement de blockchain Ethereum local utilisé qui permet de tester rapidement et efficacement les contrats avant leur déploiement sur le mainnet. Le déploiement s'est fait avec des fichiers de migration.

3.3.4.1 Déploiement du contrat zk-KYC

: Le déploiement du contrat zk-KYC se fait en deux étapes :

- On déploie de contrat KYCVerifier qui permet de vérifier les preuves ZK on chain. Ce fichier est généré lors de la génération des preuves avec SnarkJS.
- Vu que le smart contract du KYC utilise le KYCVerifier, on passe l'adresse du KYCVerifier au contrat zk-KYC lors du déploiement.

Ce code ci-dessous illustre le processus :

```
const KYCVerifier = artifacts.require("Groth16Verifier");
const KYCRegistry = artifacts.require("KYCRegistry");
module.exports = function (deployer) {
  deployer.deploy(KYCVerifier).then(() => {
    return deployer.deploy(KYCRegistry, KYCVerifier.address);
  });
};
```

Déploiement du contrat du mixer : Le même processus est suivi pour le déploiement du contrat du mixer. La seule différence ici est que le mixer a besoin en plus de l'adresse du contrat du KYC et d'un montant fixe de dépôt sur ce contrat (denomination). Vu que les déploiements sont faits séparément, on passe l'adresse du contrat KYC manuellement au deployer :

```
const WithdrawVerifier = artifacts.require("Groth16Verifier");
const Mixer = artifacts.require("Mixer");

module.exports = function (deployer) {
  deployer.deploy(WithdrawVerifier).then(() => {
    return deployer.deploy(
      Mixer,
      WithdrawVerifier.address,
      "0x272bbc80B0a73336ed3D546edf8BD101733670BB",
      BigInt(1000000000000000000)
    );
  });
};
```

Dans cet exemple, 0x272bbc80B0a73336ed3D546edf8BD101733670BB est l'adresse du contrat KYC obtenue après déploiement et BigInt(1000000000000000000) est le montant que reçoit ce contrat en WEI (ici 10 ETH).

Pour exécuter la migration, nous exécutons la commande suivante dans le répertoire du projet Truffle :

```
$ truffle migrate
```

La figure ci-dessous présente le résultat de la migration des smart contracts sur Ganache.

```

1_deploy_verifier_and_mixer.js
=====
Deploying 'Groth16Verifier'
-----
> transaction hash: 0x7907024a0b864f1d7226a599d997a8e36d37e1d8c551fc2e723c4e7521d03d00
> Blocks: 0
> contract address: 0x4558598edF70F38e295C45651F350e91B5820815
> block number: 3
> block timestamp: 1739575152
> account: 0xF3B546e1e25198F374c118B3F62a030Bbb84aE47
> balance: 999.992511455225788307
> gas used: 439157 (0x6b375)
> gas price: 3.181767596 gwei
> value sent: 0 ETH
> total cost: 0.001397295512156572 ETH

Deploying 'Mixer'
-----
> transaction hash: 0xe20144773e750e369d380b4ceb817603dedeaebbb39689ffe4caa363be27c9b
> Blocks: 0
> contract address: 0xa7773c2c33e8f0a4448E3749d1dd857f2c359679
> block number: 4
> block timestamp: 1739575153
> account: 0xF3B546e1e25198F374c118B3F62a030Bbb84aE47
> balance: 999.983108361578855939
> gas used: 3034194 (0x2e4c52)
> gas price: 3.099041672 gwei
> value sent: 0 ETH
> total cost: 0.009403093646932368 ETH

> Saving artifacts
-----
> Total cost: 0.01080038915908894 ETH

Summary
=====
> Total deployments: 2
> Final cost: 0.01080038915908894 ETH

```

FIGURE 3.6 – Déploiement des smart contracts du Mixer

3.4 Développement des interfaces utilisateur

Les interfaces utilisateur constituent un élément crucial de notre solution d’anonymisation car c’est ce que verront les utilisateurs du système. Nous avons développé deux interfaces web distinctes et intuitives en utilisant Nuxt.js. La première interface est dédiée au processus de vérification KYC, permettant aux utilisateurs de soumettre leurs informations d’identité et de gérer leurs adresses whitelisted. La seconde interface, baptisée Phantom ETH, est notre solution d’anonymisation.

3.4.1 L’interface du KYC

L’interface de vérification KYC permet aux utilisateurs de soumettre leurs informations d’identité et de gérer leurs adresses whitelisted. Voici ci-dessous l’architecture de l’interface KYC :

```

zkkyc-ui/
|-- pages/
|   |-- index.vue           Page d'accueil et connexion MetaMask
|   |-- kyc.vue             Interface principale de vérification KYC
|   |-- verify-proof.vue    Page de vérification des preuves générées
|   |-- whitelist.vue       Gestion des adresses whitelisted
|-- stores/
|   |-- web3.js             Store Pinia pour l'état Web3 et les fonctions
|-- truffle/
|   |-- circuits/
|   |   |-- kyc_verifier.circom  Circuit Circom pour la vérification ZK
|   |-- contracts/
|   |   |-- KYCRegistry.sol      Smart contract de gestion du registre KYC
|   |   |-- KYCVerifier.sol      Smart contract de vérification des preuves

```

3.4.1.1 Composants principaux de l'interface

Voici une liste des composants principaux de l'interface KYC :

1. **Le formulaire KYC** : C'est le formulaire qui permettra aux utilisateurs de soumettre leurs informations personnelles. Après avoir soumis ses informations, on envoie une requête POST vers le serveur backend.

```
app.post("/generate-proof", async (req, res) => {
  try {
    const { fullName, dateOfBirth, country, idNumber } = req.body;

    const poseidon = await circomlibjs.buildPoseidon();

    const nameHash = poseidon.F.toString(poseidon([name]));
    const dateHash = poseidon.F.toString(poseidon([date]));
    const idHash = poseidon.F.toString(poseidon([id]));
    const expectedHash = poseidon.F.toString(
      poseidon([nameHash, dateHash, countryHash, idHash]));
```

Les informations de l'utilisateur sont récupérées et hashées avec Poseidon, le hash attendu est calculé et prêt pour être passé à SnarkJS pour la génération de la preuve.

2. **Génération de la preuve** : Après le hashage des informations, SnarkJS génère la preuve qui est retournée à l'utilisateur.

```
const { proof, publicSignals } = await snarkjs.groth16.fullProve(
  input,
  wasmPath,
  zkeyPath
);
```

3. **Whitelist des adresses** : Après la vérification d'identité de l'utilisateur, l'utilisateur peut maintenant whitelister ses adresses Ethereum avec la fonction `addWhitelistedAddress`. Cette fonction prend en paramètre l'adresse à whitelister et le hash obtenu lors de la soumission KYC.

```
const receipt = await this.kycRegistry.methods
  .addWhitelistedAddress(_hash, _address)
  .send({ from: this.address });
```

Les informations de l'utilisateur sont récupérées et hashées avec Poseidon, le hash attendu est calculé et prêt pour être passé à SnarkJS pour la génération de la preuve.

3.4.2 L'interface du Mixer (Phantom ETH)

L'interface Phantom ETH représente la partie visible de notre système d'anonymisation. Il a été développée pour simplifier l'utilisation du système de mixage tout en maintenant un niveau élevé de

confidentialité. L'application est structurée autour de deux fonctionnalités principales : le dépôt et le retrait d'ETH. Voici ci-dessous l'architecture de l'interface KYC :

```

zkkyc-ui/
|-- pages/
|   |-- index.vue           Page d'accueil et connexion MetaMask
|-- stores/
|   |-- web3.js             Store Pinia pour l'état Web3 et les fonctions
|-- truffle/
|   |-- circuits/
|       |-- merkle_tree.circom  Circuit Circom pour l'arbre de Merkle
|       |-- withdraw.circom     Circuit Circom pour la gestion du retrait
|   |-- contracts/
|       |-- MerkleTreeWithHistory.sol  Smart contract de l'arbre de Merkle
|       |-- Mixer.sol               Smart contract du mixer

```

3.4.2.1 Fonctionnalités principales de l'interface du mixer

Voici une liste des fonctionnalités principales de l'interface du mixer :

1. **Processus de dépôt** : Le mixer est constitué de plusieurs pools et chaque pool reçoit un montant fixe et prédéfinis d'Ethereum. Lorsque l'utilisateur effectue un dépôt sur l'un des pools, on génère à l'utilisateur deux nombres secrets, le **nullifier** et le **secret** qui sont concaténés et hashés avec MimcSponge , plus placé dans l'arbre de Merkle sur la blockchain. On peut voir ci-dessous le processus de génération du commitment :

```

app.post("/create-deposit", async (req, res) => {
  try {
    const { poolAmount, selectedPool } = req.body;

    const _nullifier = "0x" + crypto.randomBytes(31).toString("hex");
    const _secret = "0x" + crypto.randomBytes(31).toString("hex");
    const { commitmentHash } = await build(_nullifier, _secret, poolAmount);
    ...
  }
});

```

Voici ci dessous le format de la note généré à l'utilisateur :

```
const note = "phantom-eth." + _amount + "." + nullifier + "." + secret;
```

2. **Processus de retrait** : L'utilisateur commence par entrer la note obtenue lors du dépôt, le montant du retrait et la nouvelle adresse Ethereum de réception des fonds. L'interface parse la note et vérifie si elle est dans le format attendu puis calcul un nouveau commitment avec Mimsponge. Le nullifier est aussi hashé pour plus tard.

```

const { amount, nullifier, secret } = parseNote(note);
const { commitmentHash } = await build(nullifier, secret, amount);

```

```
const _nullifierHash = mimcsponge.multiHash([nullifier], 0);
const nullifierHash = mimcsponge.F.toString(_nullifierHash);
```

Grâce à la fonction `getPastEvents` de `web3.js`, le système récupère tous les événements de dépôt qui ont eu lieu sur le smart contract, ce qui permet d'extraire les commitments de l'arbre de Merkle on chain pour vérifier si le nouveau commitment calculé est valide.

```
const pastEvents = await mixer.getPastEvents("Deposit", {
  fromBlock: 0,
  toBlock: "latest",
});
const leaves = pastEvents
  .sort((a, b) =>
    String(a.returnValues.leafIndex).localeCompare(
      String(b.returnValues.leafIndex)
    )
  )
  .map((e) => e.returnValues.commitment.toString());

const currentDepositEvent = pastEvents.find(
  (e) => e.returnValues.commitment === BigInt(commitmentHash)
);
```

Un nouveau arbre est reconstruit avec ces commitments et en utilisant Keccak256 comme dans le smart contract pour vérifier si la racine de cette dernière est parmi l'historique des racines connue. Cette vérification se fait avec la fonction `isKnownRoot` du smart contract.

```
const tree = new MerkleTree(20, leaves, {
  hashFunction: (left, right) => {
    const input = Buffer.concat([
      Buffer.from(BigInt(left).toString(16).padStart(64, "0"), "hex"),
      Buffer.from(BigInt(right).toString(16).padStart(64, "0"), "hex"),
    ]);

    const hash = createKeccakHash("keccak256").update(input).digest("hex");

    return BigInt("0x" + hash).toString();
  },
  zeroElement: 0,
});

const isValidRoot = await mixer.methods.isKnownRoot(root).call();
const isSpent = await mixer.methods.isSpent(nullifierHash).call();

if (!isValidRoot) {
  throw new Error("Deposit not found for this note !");
}
```


Une fois la validité de ce nouvel arbre est prouvé et que le commitment calculé est valide, on recrée l'arbre avec Mimcsponge, la fonction de hashage utilisé dans notre circuit circom. La racine de l'arbre et une preuve Merkle sont calculées et passées à SnarkJS pour la génération de la preuve ZK. Une fois la preuve générée, toutes les informations de retrait plus la preuve sont passés à un relayeur de transaction pour continuer le processus de retrait.

3.5 Mise en place du relayeur de transactions

Le relayeur de transactions est un composant essentiel de notre système, conçu pour exécuter les retraits de manière anonyme et sécurisée. Le relayeur est utile à cause du problème de la transaction initiale. Les retraits du mixeur se font généralement avec des adresses entièrement nouvelles pour améliorer la confidentialité, mais ces adresses n'ont pas de gaz pour payer le retrait.

Pour résoudre ce problème, le système utilise un relayeur qui permet de payer les frais de gas et d'effectuer la transaction pour l'utilisateur en échange d'une commission. Ce qui permet de masquer complètement le lien entre l'adresse de dépôt et l'adresse de retrait.

3.5.1 Architecture du relayeur

Développé en Node.js avec Express, il agit comme un intermédiaire qui exécute les transactions de retrait pour le compte des utilisateurs.

```
server/  
|-- config/  
|   |-- web3.js           Connexion avec web3  
|-- relayer.js
```

3.5.2 Fonctionnement

Lorsqu'un utilisateur initie une demande de retrait, il soumet sa note de dépôt et une preuve Zero-Knowledge attestant de la validité de son dépôt est générée. Le relayeur récupère cette preuve et l'adresse de destination pour poursuivre la procédure de retrait. Il utilise la fonction `estimateGas()` pour estimer les frais de gas et vérifier si la commission du relayeur couvre ces frais.

```
...  
const account = web3.eth.accounts.privateKeyToAccount(PRIVATE_KEY);  
const fee = web3.utils.toWei(amount * FEES, "ether");  
  
const gasEstimate = await mixer.methods  
    .withdraw(  
        ...  
    )  
    .estimateGas();  
  
const gasPrice = await web3.eth.getGasPrice();  
const gasCost = BigInt(gasPrice) * BigInt(gasEstimate);
```

```
if (BigInt(fee) <= gasCost) {  
    throw new Error("Fee does not cover gas costs");  
}
```

Une fois ces vérifications effectuées, le relayeur signe la transaction de retrait, fait le paiement des frais de gas et soumet le retrait au smart contract. L'utilisateur reçoit ses fonds sur sa nouvelle adresse et le hash de la transaction lui est retourné.

```
...  
return {  
    success: true,  
    txHash: relayer.txHash,  
    message:  
        "Withdraw successful ! Transaction Hash: " + relayer.txHash,  
};
```

Conclusion

Dans ce chapitre, on a détaillé la modélisation et le développement du système d'anonymisation, intégrant des circuits ZK, des smart contracts sur Ethereum et une API de relayeur pour les transactions. Nous avons présenté la conception des différents modules (vérification KYC, dépôt dans le mixer, génération de preuves ZK, interfaces web et retraits via le relayeur) ainsi que les choix technologiques qui ont permis de garantir la confidentialité des transactions dans le système. Ces éléments constituent la base technique solide de notre solution, et les résultats sont présentés dans le chapitre suivant.

Chapitre 4

Résultats et Discussion

Introduction

Dans ce chapitre, nous présentons les résultats obtenus lors du déploiement de notre système d'anonymisation des transactions, ainsi que les discussions sur les performances, les limitations et les perspectives d'amélioration.

4.1 Résultats obtenus

4.1.1 Vérification d'identité ZK

Le rendu de l'interface de vérification KYC offre une expérience utilisateur fluide et intuitive. Pour pouvoir utiliser la plateforme, l'utilisateur connecte son portefeuille Metamask. Voici ci-dessous la page d'accueil de notre plateforme de vérification d'identité à divulgation nulle de connaissance.

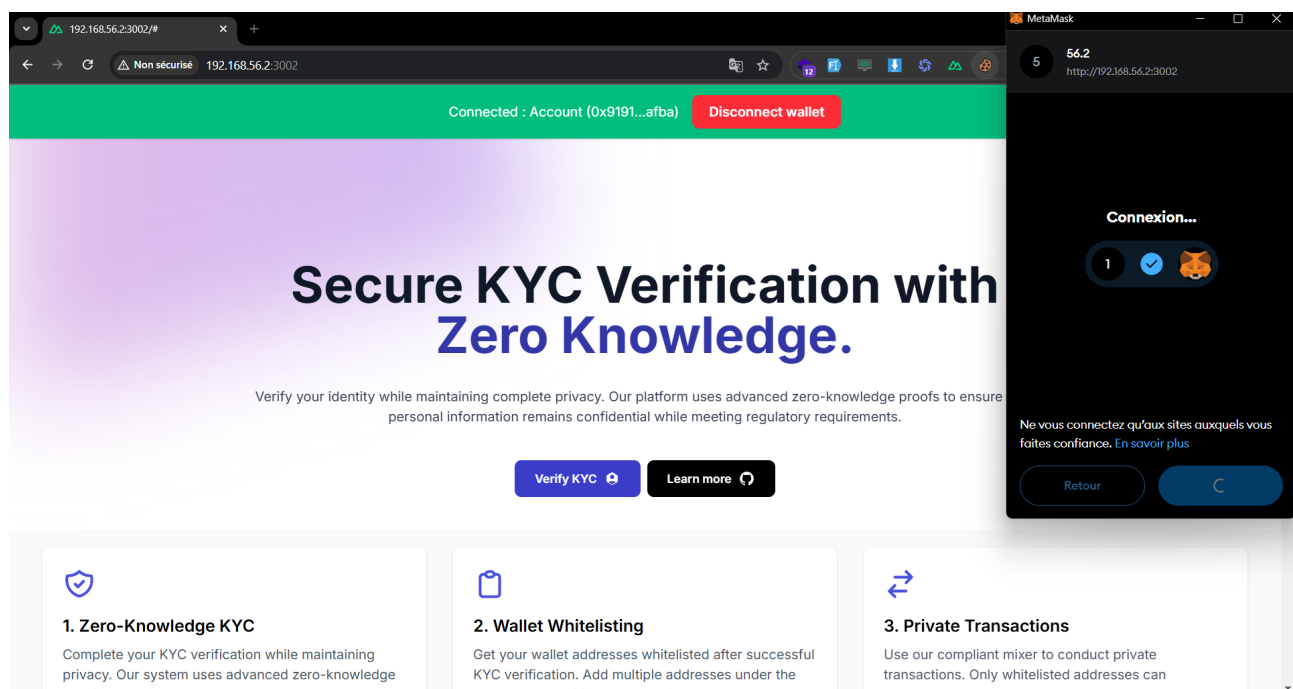
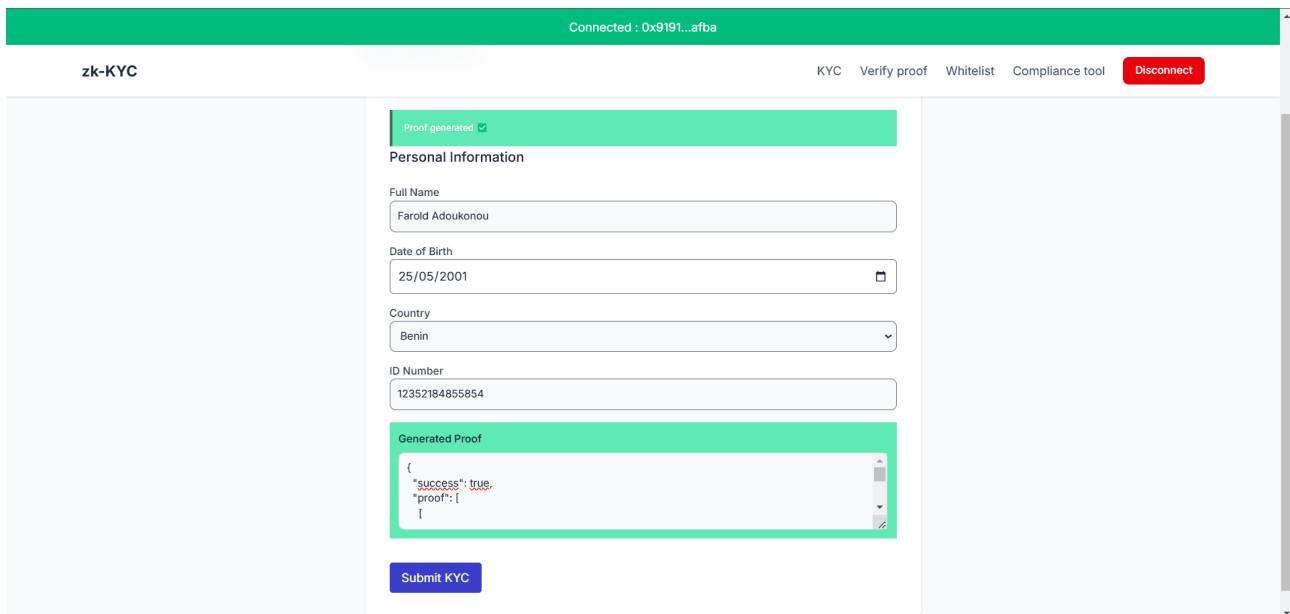


FIGURE 4.1 – Connexion avec Metamask à ZK KYC

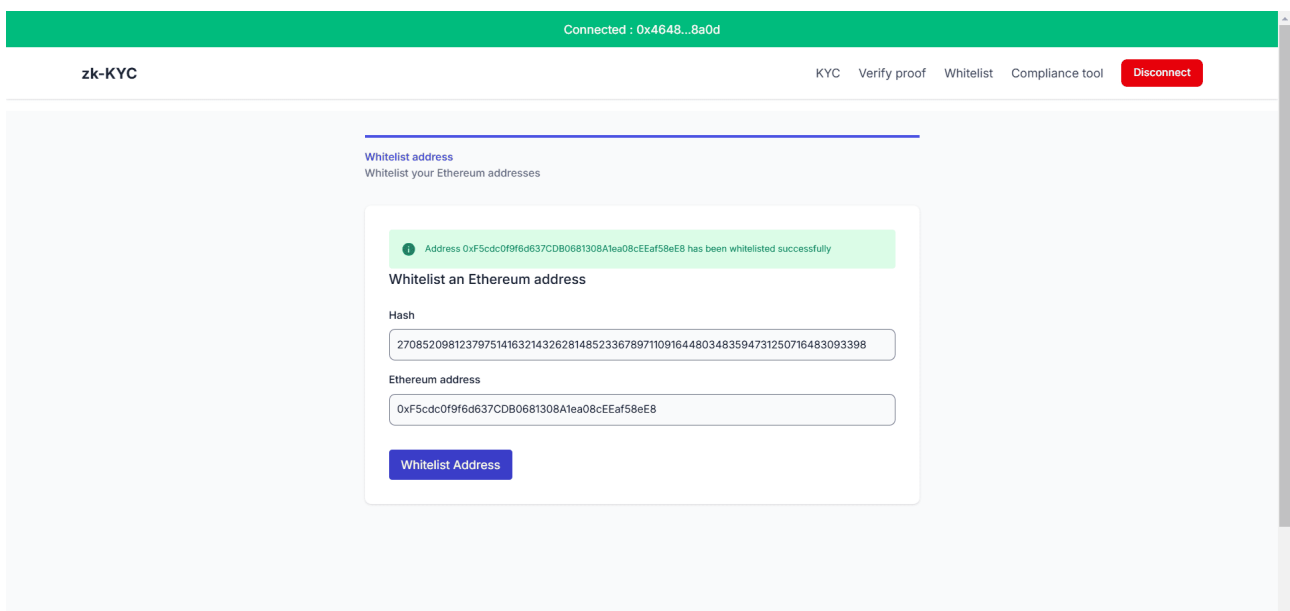
Une fois connecté, l'utilisateur peut entrer ses informations personnelles pour pouvoir générer une preuve ZK :



The screenshot shows the zk-KYC web application interface. At the top, a green header bar displays "Connected : 0x9191...afba". Below the header, the navigation bar includes "zk-KYC", "KYC", "Verify proof", "Whitelist", "Compliance tool", and a red "Disconnect" button. The main content area is titled "Personal Information" and contains several input fields: "Full Name" (Farold Adoukonou), "Date of Birth" (25/05/2001), "Country" (Benin), and "ID Number" (12352184855854). A green "Generated Proof" box displays a JSON object: {"success": true, "proof": {}}. A blue "Submit KYC" button is located at the bottom of the form.

FIGURE 4.2 – Vérification d'identité et génération de preuve

L'utilisateur peut maintenant whitelister ses autres adresses Ethereum avec le exceptedHash obtenu et une adresse déjà whitelisted.



The screenshot shows the zk-KYC web application interface for the "Whitelist" section. The header bar displays "Connected : 0x4648...8a0d". The navigation bar includes "zk-KYC", "KYC", "Verify proof", "Whitelist", "Compliance tool", and a red "Disconnect" button. The main content area is titled "Whitelist address" and "Whitelist your Ethereum addresses". A green success message box states: "Address 0xF5cdc0f9f6d637CDB0681308A1ea08cEEaf58eE8 has been whitelisted successfully". Below this, the "Whitelist an Ethereum address" form contains two input fields: "Hash" (2708520981237975141632143262814852336789711091644803483594731250716483093398) and "Ethereum address" (0xF5cdc0f9f6d637CDB0681308A1ea08cEEaf58eE8). A blue "Whitelist Address" button is located at the bottom of the form.

FIGURE 4.3 – Whiteliste d'adresse avec un hash valide

4.1.2 Mixeur de transactions (Phantom ETH)

L'interface du mixeur Phantom ETH est conçue pour être simple, intuitive et accessible à tous les utilisateurs. Pour pouvoir utiliser la plateforme, l'utilisateur connecte son portefeuille Metamask. Voici ci-dessous la page d'accueil de notre plateforme de mixeur de transactions :

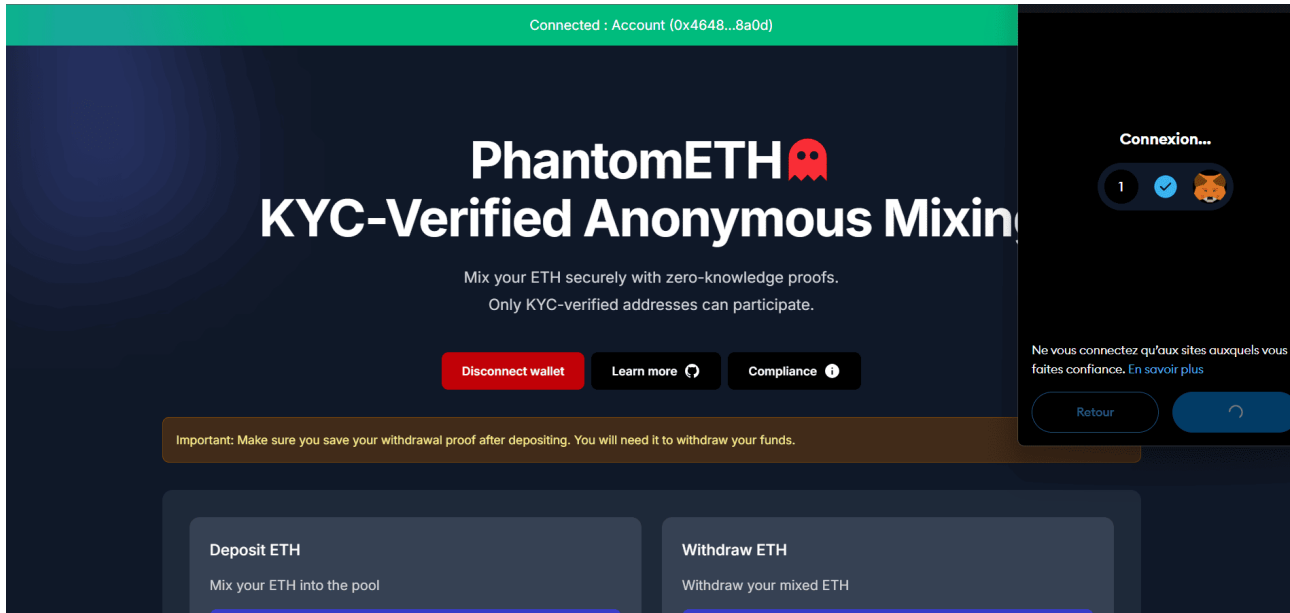


FIGURE 4.4 – Connexion avec Metamask à Phantom ETH

Une fois connecté, l'utilisateur peut maintenant effectuer des dépôts et des retraits sur la plateforme.

4.1.2.1 Processus de dépôt sur Phantom ETH

Le dépôt est le processus par lequel les utilisateurs transfèrent des fonds de leurs portefeuilles Ethereum (dans ce cas, Metamask) vers le smart contract du pool qu'ils ont sélectionné. Voici ci-dessous le formulaire de dépôt :

FIGURE 4.5 – Formulaire de dépôt sur Phantom ETH

Pour effectuer le dépôt, l'utilisateur sélectionne le pool du montant qu'il veut déposer et soumet le formulaire. Une fenêtre de Metamask s'ouvre pour qu'il valide le transfert des fonds vers le smart contract. Une fois la requête confirmée, le système traite le dépôt et génère la note secrète à l'utilisateur. L'utilisateur reçoit une notification confirmant le dépôt des fonds et la note est affichée et aussi sauvegardée en local dans un fichier texte.

FIGURE 4.6 – Note secrète après dépôt sur Phantom ETH

On peut constater dans la figure ci-dessous que lorsque l'utilisateur est connecté avec une adresse non whitelistée, le dépôt échoue.

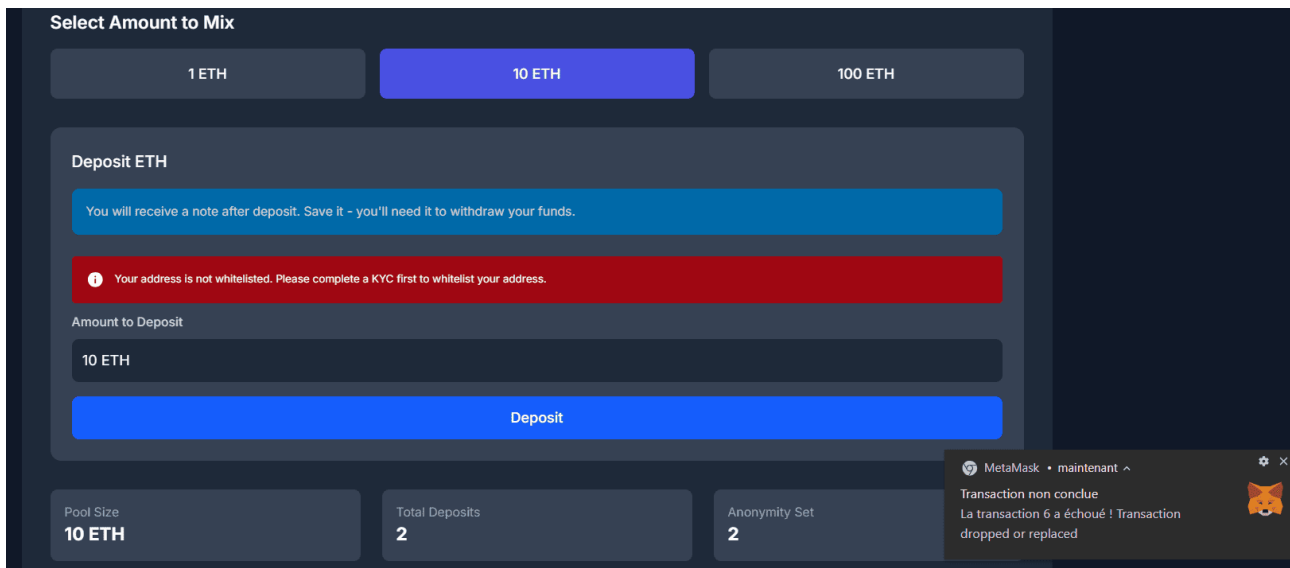


FIGURE 4.7 – Erreur de dépôt avec une adresse non whitelisted

4.1.2.2 Processus de retrait sur Phantom ETH

Le retrait permet aux utilisateurs de retirer les fonds déposés sur un des pools du smart contract vers une adresse externe. Lorsque l'utilisateur entre sa note secrète et l'adresse de réception des fonds, la demande de retrait est soumise au serveur backend qui génère une preuve de la validité de la note et l'envoie au relayeur. Il est à noter ici qu'aucune requête vers Metamask n'est faite pour la confirmation du retrait. En effet, le relayeur s'en charge. Voici ci-dessous le formulaire de retrait :

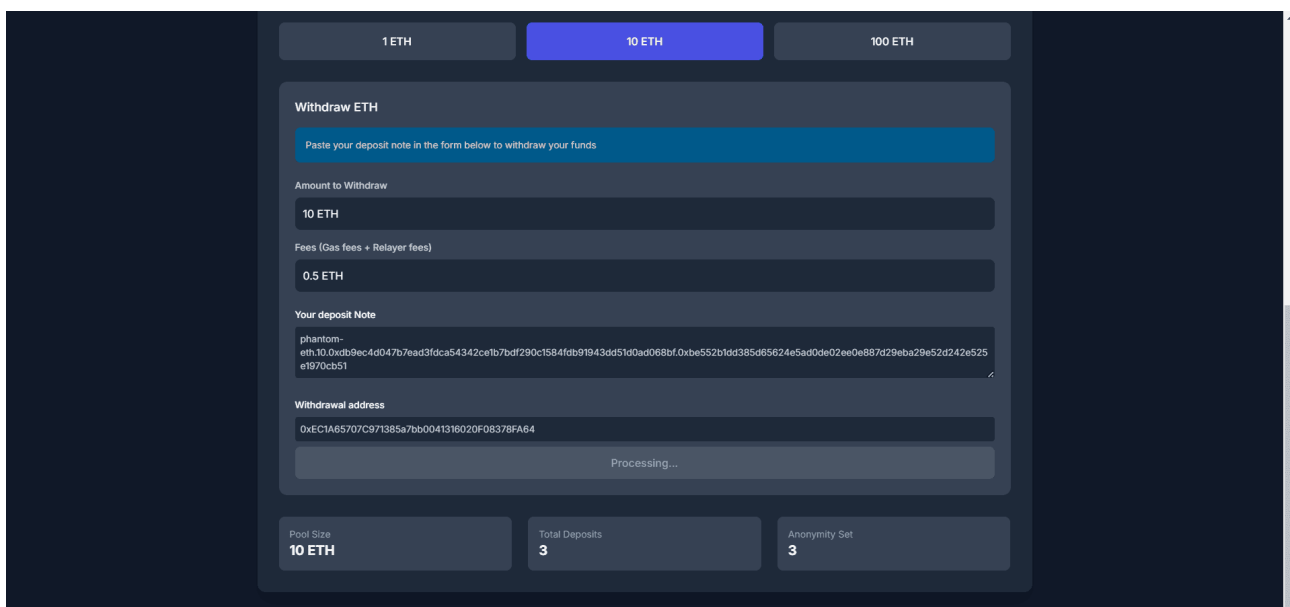


FIGURE 4.8 – Soumission de la demande de retrait sur Phantom ETH

Lorsque l'adresse de réception des fonds n'est pas whitelisted, le système avorte le processus de retrait et envoie une erreur. Une fois le retrait effectué, le système renvoie le hash de la transaction à l'utilisateur.

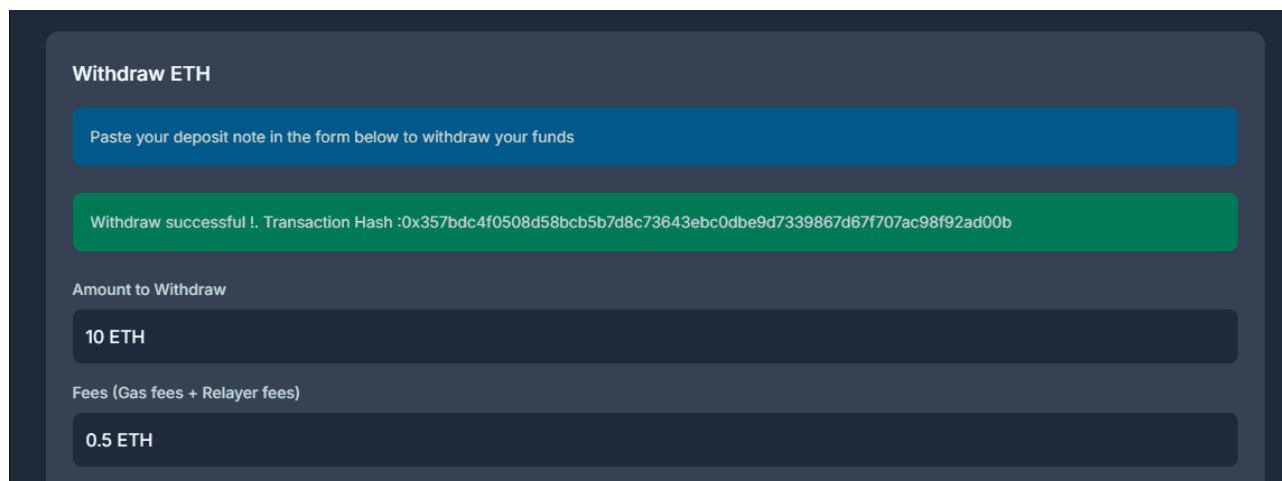


FIGURE 4.9 – Fonds retirés avec succès sur Phantom ETH

4.2 Discussions

4.2.1 Le projet Phantom ETH

Le projet Phantom ETH est né d’une réflexion sur la nécessité de concilier la confidentialité des transactions sur Ethereum avec les exigences réglementaires croissantes auxquelles sont confrontées les plateformes de finance décentralisée. En s’inspirant du célèbre mixeur Tornado Cash, Phantom ETH propose une approche équilibrée entre l’anonymat des utilisateurs et la conformité réglementaire.

La particularité de notre solution réside dans son intégration native du whitelist d’adresses avant le mixage basé sur un système KYC à divulgation nulle de connaissance. Cette approche permet de répondre aux exigences KYC/AML sans compromettre la confidentialité des utilisateurs. L’architecture du projet reflète cette approche, avec une séparation claire entre la vérification d’identité et le mécanisme de mixage.

4.2.2 Comparatif de notre solution avec Tornado Cash

Voici ci-dessous un comparatif de notre solution d’anonymisation avec Tornado Cash :

4.2.3 Défis rencontrés

Au cours du développement de notre solution, nous avons fait face à plusieurs défis techniques et conceptuels à savoir :

- **Implémentation des circuits ZK** : Le développement des circuits ZK avec Circom a constitué l’un des défis les plus complexes
- **Implémentation des smart contracts** : Le développement du smart contract de l’arbre de Merkle et l’optimisation de la consommation de gas des smart contracts.
- **Les tests et déploiement** : Les phases de test et de déploiement ont révélé des défis comme la mise en place d’un environnement de développement et la validation des preuves ZK dans différents scénarios.

Caractéristiques	Phantom ETH	Tornado Cash
Fonctionnalités		
Anonymisation des transactions	✓	✓
Vérification KYC intégrée	✓	×
Whitelisting en cascade	✓	×
Support multi-denominations	×	✓
Technique		
Utilisation de zk-SNARK	✓	✓
Arbre de Merkle on-chain	✓	✓
Relayeurs décentralisés	×	✓
Nullifier hash	✓	✓
Conformité		
Conformité réglementaire	✓	×
Traçabilité KYC	✓	×
Protection vie privée	✓	✓
Expérience Utilisateur		
Interface simplifiée	✓	✓
Gestion des notes	✓	✓

TABLE 4.1 – Comparaison entre Phantom ETH et Tornado Cash

4.2.4 Limitations actuelles de la solution

Notre solution présente encore plusieurs limitations malgré son fonctionnement. Tout d’abord, le coût en gas sur Ethereum reste élevé, surtout à cause des vérifications de preuves on chain, ce qui impacte significativement l’efficacité économique des transactions. Ensuite, une file d’attente n’a pas été implémentée pour le traitement des dépôts et retraits concurrents, ce qui peut engendrer des délais, surtout en période de forte activité. Enfin, le système repose actuellement sur un relayeur centralisé pour exécuter les retraits. Ce qui va à l’encontre du principe de décentralisation .

Pour améliorer la robustesse et la scalabilité, il serait nécessaire de déployer un nombre plus important de relayeurs. Ces défis devront être résolus afin d’optimiser l’accessibilité et la performance globale de la solution.

4.2.5 Perspectives d’évolution du projet

Notre projet Phantom ETH, bien que fonctionnel dans son état actuel, présente de nombreuses opportunités d’amélioration et d’évolution. Nous avons, entre autres :

- **Support cross-chain** : Le système pourrait être étendu pour supporter d’autres types d’actifs et de blockchains.
- **Support de dénominations personnalisés** : Pour le cas de notre projet, nous avons travaillé avec des pools de transactions fixes (1 ETH, 10 ETH, 100 ETH). L’idéal serait d’avoir un pool de transaction qui prends des montant définis par l’utilisateur.
- **Décentralisation du Relayeur** : Notre relayeur est hébergé sur notre serveur backend, ce qui introduit un point de confiance et va à l’encontre des principes de décentralisation. Nous pouvons y remédier en déployant un réseau de relayeurs décentralisés.

- **Coût en Gaz :** Dans le système, les opérations sur les arbres de Merkle et les vérification de preuves on chain sont coûteuses en gas.
- **Audit et Sécurité :** Faire auditer les contrats et les circuits zk-SNARKs par des experts en sécurité et par la communauté.

Conclusion

Dans ce chapitre, nous avons présenté les résultats obtenus après la mise en place de notre solution. Les résultats de nos tests démontrent que notre solution Phantom ETH atteint son objectif principal : concilier l'anonymisation des transactions sur Ethereum avec les exigences de conformité réglementaire. L'intégration du système KYC basé sur les preuves à divulgation nulle de connaissance s'est révélée particulièrement efficace, offrant un équilibre unique entre confidentialité et conformité.

Néanmoins, certaines limitations ont été identifiées, notamment en termes de coûts de transaction sur Ethereum et de support limité des dénominations. Ces points d'amélioration, ainsi que les perspectives d'évolution identifiées, ouvrent la voie à de futures versions plus complètes et optimisées de la solution.

Conclusion Générale

Dans ce mémoire, nous avons exploré en profondeur la problématique de la confidentialité des transactions sur la blockchain Ethereum, en proposant une solution qui concilie anonymat et conformité réglementaire. Notre système, basé sur les preuves à divulgation nulle de connaissance, permet aux utilisateurs de protéger leur vie privée tout en respectant les exigences KYC/AML essentielles au secteur financier.

Notre prototype se compose de plusieurs éléments interdépendants : les smart contracts pour la gestion du KYC et du mixage, les circuits ZK pour la génération des preuves, et les interfaces utilisateur pour une expérience fluide et intuitive. Les tests réalisés sur Ganache ont démontré la viabilité et l'efficacité de cette architecture.

Les perspectives d'évolution comprennent diverses pistes d'amélioration, telles que la réduction des coûts de gas, une meilleure gestion des files d'attente des transactions et la mise en place de nombreux relayeurs pour rendre le système plus rapide et efficace. Ces avancées seront cruciales pour améliorer l'accessibilité et la performance de notre solution.

En définitive, ce projet démontre qu'il est possible de construire des solutions préservant la confidentialité sur Ethereum tout en respectant le cadre réglementaire, ouvrant ainsi la voie à une utilisation plus privée et responsable de la blockchain.

Bibliographie

- [1] Roberto Infante. Building ethereum dapps : Decentralized applications on the ethereum blockchain. *Building Ethereum Dapps : Decentralized applications on the Ethereum blockchain*, 2019.

Webographie

- [2] Aml : How to comply with online anti-money laundering regulations. <https://fastercapital.com/fr/contenu/AML---Comment-se-conformer-aux-reglementations-anti-blanchiment-d-argent-en-ligne.html>. ,consulté le 01 Novembre 2024.
- [3] <https://www.futura-sciences.com/tech/definitions/cryptomonnaie-blockchain-18277>. ,consulté le 17 Septembre 2024.
- [4] Crypto mixers and tumblers, what are they? <https://preciousruby.medium.com/crypto-mixers-tumblers-what-are-they-579eb624d885>. ,consulté le 15 Novembre 2024.
- [5] Quelles sont les différences entre un custodial / non custodial wallet? <https://cryptoast.fr/quelles-sont-les-differences-entre-un-custodial-non-custodial-wallet/>. ,consulté le 19 Octobre 2024.
- [6] What is defi? <https://www.coinbase.com/fr/learn/crypto-basics/what-is-defi>. ,consulté le 15 Octobre 2024.
- [7] What is dexts? <https://trustwallet.com/fr/blog/what-is-a-dex-beginners-guide>. ,consulté le 05 Novembre 2024.
- [8] What is ethereum 2.0 and why does it matter? <https://academy.binance.com/fr/articles/what-is-ethereum-2-0-and-why-does-it-matter>. ,consulté le 19 Septembre 2024.
- [9] https://www.ic3.gov/AnnualReport/Reports/2023_IC3Report.pdf. ,consulté le 05 Novembre 2024.
- [10] Gaz et frais. <https://ethereum.org/fr/developers/docs/gas/>. ,consulté le 15 Octobre 2024.
- [11] Groth16. <https://docs.sui.io/guides/developer/cryptography/groth16>. ,consulté le 27 Novembre 2024.
- [12] Heuristique de fiat-shamir. https://fr.wikipedia.org/wiki/Heuristique_de_Fiat-Shamir. ,consulté le 05 Décembre 2024.
- [13] Kyc : definition, procedure and regulations. <https://www.signicat.com/fr/blog/kyc-know-your-customer-importance>. ,consulté le 23 Octobre 2024.

-
- [14] North korea's lazarus hackers behind recent crypto heists : Fbi. <https://therecord.media/north-korea-lazarus-behind-crypto-heists>. ,consulté le 17 Novembre 2024.
- [15] Les mécanismes de consensus. <https://coinacademy.fr/academie/differents-algorithmes-consensus-blockchain/>. ,consulté le 19 Septembre 2024.
- [16] How are merkle trees used in blockchains? <https://docs.alchemy.com/docs/merkle-trees-in-blockchains>. ,consulté le 07 Décembre 2024.
- [17] Merkle tree in blockchain : What is it, how does it work and benefits. <https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain>. ,consulté le 07 Décembre 2024.
- [18] Mimc. <https://byt3bit.github.io/primesym/mimc/>. ,consulté le 15 Janvier 2025.
- [19] Combining on-chain and off-chain analysis : A primer. <https://www.soliduslabs.com/post/off-chain-and-on-chain-analysis>. ,consulté le 19 Octobre 2024.
- [20] What is open-source intelligence? <https://www.sans.org/blog/what-is-open-source-intelligence/>. ,consulté le 18 Octobre 2024.
- [21] What are privacy coins? [https://www.nervos.org/knowledge-base/what_are_%20privacy_coins_\(explainCKBot\)](https://www.nervos.org/knowledge-base/what_are_%20privacy_coins_(explainCKBot)). ,consulté le 20 Novembre 2024.
- [22] Identité : Quelle est la différence entre le pseudonymat et l'anonymat? <https://start-in-blockchain.fr/pseudonymat-anonymat-definition-difference/>. ,consulté le 18 Octobre 2024.
- [23] Reentrancy attacks in smart contracts explained. <https://www.binance.com/en/square/post/8177762116393>. ,consulté le 07 Février 2025.
- [24] Solidity documentation. <https://docs.soliditylang.org/en/latest/>. ,consulté le 15 Janvier 2025.
- [25] Victory for tornado cash as court rules sanctions were unlawful. <https://www.bakerlaw.com/insights/victory-for-tornado-cash-as-court-rules-sanctions-were-unlawful/>. ,consulté le 30 Novembre 2024.
- [26] What is web3.js? <https://how.dev/answers/what-is-web3js>. ,consulté le 15 Janvier 2025.
- [27] The knowledge complexity of interactive proof systems. https://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Proof%20Systems/The_Knowledge_Complexity_Of_Interactive_Proof_Systems.pdf. ,consulté le 02 Décembre 2024.
- [28] Zero-knowledge proof (zkp) : Working mechanism and use cases. <https://medium.com/@centicio/zero-knowledge-proof-zkp-working-mechanism-and-use-cases-cc77afc98b08>. ,consulté le 02 Décembre 2024.
- [29] Zero-knowledge proof – how it works. <https://hacken.io/discover/zero-knowledge-proof/>. ,consulté le 05 Décembre 2024.

Table des matières

Dédicace	ii
Remerciements	iii
Résumé	iv
Abstract	v
Liste des figures	vi
Liste des tableaux	vii
Sigles et Abréviations	viii
Glossaire	ix
Introduction	1
1 Etat de l'art	3
Introduction	3
1.1 Généralités sur la blockchain	3
1.1.1 Définition	3
1.1.2 Les types de blockchain	4
1.1.3 Architecture et fonctionnement de la blockchain	4
1.1.3.1 Architecture de la blockchain	4
1.1.3.2 Fonctionnement de la blockchain	5
1.1.3.3 Mécanismes de consensus	6
1.1.4 La blockchain Ethereum	7
1.1.4.1 Définition	7
1.1.4.2 Ethereum Virtual Machine	7
1.1.4.3 Le gaz	8
1.1.4.4 Les contrats Intelligents (smarts contracts)	8
1.1.4.5 Les dApp	9
1.2 Anonymat et Confidentialité sur la blockchain Ethereum	9
1.2.1 Différence entre pseudonymat et anonymat réel	9
1.2.1.1 Introduction des concepts	9
1.2.1.2 Implication pour les utilisateurs	10
1.3 Régulations et Conformité : KYC et AML	10
1.3.1 Introduction aux régulations financières	10

1.3.2	Blanchiment d'argent via les cryptomonnaies : Une tendance ces dernières années	11
1.3.3	Objectifs des régulations KYC et AML	11
1.3.4	KYC : Know Your Customer	11
1.3.5	AML : Anti Money Laundering	12
1.3.6	Impact des régulations sur les transactions financières sur la blockchain	12
1.3.6.1	Conformité et adhésion aux normes	12
1.3.6.2	Sécurisation des transactions	13
1.4	Techniques d'anonymisation des transactions sur la blockchain	13
1.4.1	Importance de l'anonymisation dans le contexte de la blockchain Ethereum	13
1.4.2	Solutions Traditionnelles d'Anonymisation : Mixers et Tumblers	13
1.4.2.1	Définition	13
1.4.2.2	Fonctionnement	13
1.4.2.3	Limitations et préoccupations	14
1.4.3	Les cryptomonnaies axées sur la confidentialité (Privacy coins)	15
1.4.3.1	Définition	15
1.4.3.2	Principe de fonctionnement	15
1.4.3.3	Exemples de privacy coins	15
1.4.4	Technologies avancées pour l'anonymisation	16
1.4.4.1	Zero-Knowledge Proofs (ZKP) : principes et applications.	16
1.4.4.2	Les zk-SNARKs	18
	zk-SNARK (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge)	18
1.4.4.3	Les arbres de Merkle	18
1.4.5	Etude de cas de projet existant : Le Fléau Tornado Cash	20
1.4.5.1	Introduction	20
1.4.5.2	Fonctionnement	20
1.4.5.3	Le processus de dépôt	20
1.4.5.4	Le processus de retrait	21
	Conclusion	22
2	Matériels et Méthodes	23
	Introduction	23
2.1	Matériels	23
2.1.1	Environnement de développement blockchain	23
2.1.1.1	Langage Solidity	23
2.1.1.2	Truffle Suite et Ganache	24
	La Truffle Suite	24
	Ganache	24
2.1.1.3	Circom et SnarkJS	24
	Circom (Circuit Compiler)	24
	SnarkJS	24
2.1.1.4	Interface utilisateur avec Nuxt.js	24
2.1.1.5	Web3.js	24
2.1.1.6	OpenZeppelin	24
2.2	Architecture du système	25

Conclusion	26
3 Modélisation et Développement	27
Introduction	27
3.1 Modélisation du Système	27
3.1.1 Flux de fonctionnement du système	27
3.1.2 Diagramme de composant du système	28
3.1.2.1 La couche Frontend	28
La couche blockchain	28
Enfin, la couche Backend	28
3.1.3 Diagramme de flux de données	29
3.1.3.1 Le processus de vérification KYC	29
3.1.3.2 Le processus de dépôt	30
3.1.3.3 Le processus de retrait	31
3.2 Développement des Circuits ZK avec Circom	33
3.2.1 Conception des Circuits	33
3.2.1.1 Le circuit <code>kyc_verifier.circom</code>	33
3.2.1.2 Le circuit <code>merkle_tree.circom</code>	33
3.2.1.3 Le circuit <code>withdraw.circom</code>	34
3.2.2 Compilation des circuits et génération des preuves	34
3.3 Modélisation des Smart Contracts	35
3.3.1 Smart Contract Zero Knowledge KYC	35
3.3.2 Smart Contract MerkleTreeWithHistory	37
3.3.3 Smart Contract Mixer	38
3.3.3.1 La gestion des dépôts :	39
3.3.3.2 La gestion des retraits :	39
3.3.3.3 Le schéma de nullification :	40
3.3.4 Déploiement des smart contracts sur Ganache	40
3.3.4.1 Déploiement du contrat zk-KYC	41
Déploiement du contrat du mixer	41
3.4 Développement des interfaces utilisateur	42
3.4.1 L'interface du KYC	42
3.4.1.1 Composants principaux de l'interface	43
3.4.2 L'interface du Mixer (Phantom ETH)	43
3.4.2.1 Fonctionnalités principales de l'interface du mixer	44
3.5 Mise en place du relayeur de transactions	46
3.5.1 Architecture du relayeur	46
3.5.2 Fonctionnement	46
Conclusion	47
4 Résultats et Discussion	48
Introduction	48
4.1 Résultats obtenus	48
4.1.1 Vérification d'identité ZK	48
4.1.2 Mixeur de transactions (Phantom ETH)	50

4.1.2.1	Processus de dépôt sur Phantom ETH	50
4.1.2.2	Processus de retrait sur Phantom ETH	52
4.2	Discussions	53
4.2.1	Le projet Phantom ETH	53
4.2.2	Comparatif de notre solution avec Tornado Cash	53
4.2.3	Défis rencontrés	53
4.2.4	Limitations actuelles de la solution	54
4.2.5	Perspectives d'évolution du projet	54
	Conclusion	55
	Conclusion	56
	Bibliographie	57
	Webographie	58
	Table des matières	60
	Annexes	64
A.1	Code du projet	64

Annexes

A.1 Code du projet

Le code source complet de Phantom ETH est disponible sur GitHub. Ce dépôt contient l'ensemble des composants du système : les smart contracts, les circuits ZK, les interfaces utilisateur et le code du relayeur.

Dépôt GitHub : [Lien du dépôt](https://github.com/cybfar/anonymous-tx) (https ://github.com/cybfar/anonymous-tx)

Le projet est organisé en plusieurs sous-dossiers :

- **zkkyc-ui/**
 - Interface utilisateur KYC
 - Composants Vue réutilisables
 - Smart contracts Solidity
 - Tests et scripts de déploiement
 - Circuits ZK-SNARK
- **mixer-ui/**
 - Interface utilisateur Phantom ETH
 - Composants Vue réutilisables
 - Smart contracts Solidity
 - Tests et scripts de déploiement
 - Circuits ZK-SNARK
- **proof-server/**
 - Code du relayeur
 - Service de génération de preuves
 - API REST

Les instructions d'installation, de configuration et de déploiement sont détaillées dans les fichiers README.md de chaque sous-dépôt. Les contributions sont les bienvenues.
