Zero-Knowledge Identity Verification

Rapport de Projet Crypto 2025

Alban Naulin & Christophe De Pontac & Térence Miralves & Ugo Majer

30 juin 2025

Table des matières

1 Vue d'ensemble des preuves à divulgation nulle de connaissance (ZKP)

Les Zero-Knowledge Proofs, ou preuves à divulgation nulle de connaissance, sont un pilier fondamental de la cryptographie moderne. Elles permettent à un individu (appelé prouveur) de démontrer à un autre (le vérificateur) qu'il connaît une information sans en révéler le contenu exact. Cette propriété, à la fois contre-intuitive et puissante, repose sur des constructions mathématiques élaborées qui garantissent à la fois la confidentialité des données et la possibilité de vérification.

Il existe plusieurs familles de protocoles ZKP. Parmi les plus populaires, on trouve les zk-SNARKs (Succinct Non-interactive Arguments of Knowledge), qui sont des preuves très compactes, vérifiables rapidement, mais nécessitent une phase initiale dite de "trusted setup". Les zk-STARKs (Scalable Transparent ARguments of Knowledge), eux, éliminent le besoin de cette phase mais au prix de preuves plus longues et de vérifications plus coûteuses. Enfin, les Bulletproofs offrent une alternative sans setup de confiance mais sont plus limités dans leur efficacité lorsqu'il s'agit de preuves complexes.

Les applications des ZKP sont nombreuses. Dans le domaine de la finance, la cryptomonnaie Zcash utilise les zk-SNARKs pour permettre des transactions anonymes vérifiables publiquement. Dans le domaine électoral, ces preuves sont envisagées pour garantir des scrutins secrets mais vérifiables. Elles sont également cruciales pour les systèmes d'identification numérique respectueux de la vie privée, ou encore dans les systèmes d'accès conditionné à certains services, produits ou zones géographiques.

2 Protocole ZKP choisi et architecture du système

Dans le cadre de ce projet, nous avons choisi d'utiliser le protocole zk-SNARK (Succinct Non-interactive Argument of Knowledge), l'un des protocoles de preuve à divulgation nulle de connaissance les plus utilisés dans les systèmes cryptographiques modernes. Ce protocole permet de produire des preuves de validité très compactes, qui peuvent être vérifiées rapidement et sans interaction avec le prouveur, tout en garantissant la confidentialité des données sensibles.

Nous avons mis en œuvre zk-SNARK à l'aide de deux outils principaux : `circom`, utilisé pour la définition des circuits de contrainte, et `snarkjs`, qui gère le trusted setup, la génération des preuves et leur vérification. `circom` permet de construire des circuits arithmétiques exprimant des relations logiques entre des entrées privées et publiques. Ces circuits sont ensuite compilés en contraintes de type R1CS (Rank-1 Constraint System), base du fonctionnement de zk-SNARK. `snarkjs` facilite le traitement de ces contraintes : il gère l'initialisation des paramètres, la génération de preuves et leur vérification dans divers contextes (navigateur, ligne de commande, etc.).

L'un des avantages majeurs de zk-SNARK réside dans la taille réduite de ses preuves (quelques centaines d'octets) et dans la vitesse de vérification (quelques millisecondes), ce qui le rend parfaitement adapté à des environnements web ou mobiles. En revanche, son principal inconvénient est la nécessité d'un trusted setup, c'est-à-dire une phase initiale de génération de paramètres communs, qui doit être menée avec rigueur pour éviter toute faille de sécurité. Si cette phase est corrompue, un attaquant pourrait générer des preuves fausses valides.

Dans notre projet, le trusted setup a été réalisé en deux temps via le protocole `powers of tau`. La première phase, universelle, permet de générer des paramètres partagés pour des circuits de taille arbitraire. Nous avons choisi un niveau de sécurité correspondant à une taille de circuit de 2^{18}, assurant une marge suffisante pour notre application. La seconde phase est spécifique au circuit que nous avons conçu ; elle fixe définitivement les paramètres utilisés pour la génération et la vérification des preuves. Ce processus a abouti à la production d'une clé de preuve (`proving key`) et d'une clé de vérification (`verification key`).

L'architecture fonctionnelle de notre système repose sur une séparation claire entre les rôles. L'utilisateur interagit avec une interface et saisit ses informations personnelles (âge, type de permis...). Ces données sont ensuite converties en entiers ou chaînes encodées de manière compatible avec `circom`. Elles sont introduites comme entrées privées dans le circuit. Le circuit vérifie alors que certaines propriétés sont vraies — par exemple, que l'âge fourni est supérieur ou égal à 18 ans, ou que le code du permis correspond à la catégorie A — sans jamais exposer la valeur exacte. Une fois la preuve générée, elle est transmise à un vérificateur (qui peut être un serveur, une autorité ou même un navigateur distant), lequel peut valider la preuve sans jamais apprendre les données originales.

Cette architecture respecte les principes fondamentaux des ZKP : confidentialité des données privées, vérifiabilité publique, et non-interactivité. Elle a l'avantage d'être simple à mettre en œuvre tout en étant relativement robuste. Toutefois, elle hérite également des limitations structurelles des zk-SNARKs : la difficulté de modification des circuits une fois les paramètres générés, la nécessité de maintenir l'intégrité des clés de preuve/vérification, et l'absence par défaut de certaines fonctionnalités de sécurité comme l'expiration temporelle ou l'ancrage externe (signature ou hash sur une blockchain).

En résumé, le choix de zk-SNARK, couplé à `circom` et `snarkjs`, nous a permis de construire un système efficace de vérification d'identité basé sur la confidentialité. Ce choix technologique, malgré quelques contraintes, présente un bon compromis entre performances, simplicité de déploiement et respect de la vie privée.

3 Choix des bibliothèques et des outils

Le choix des outils et bibliothèques utilisés pour la réalisation de ce projet a été guidé par plusieurs critères fondamentaux : la maturité des technologies, leur compatibilité avec les protocoles zk-SNARK, la disponibilité de la documentation et leur intégration dans des écosystèmes modernes de développement. À l'issue de cette analyse, nous avons retenu principalement `circom` et `snarkjs` pour la conception et l'exécution des preuves, ainsi que JavaScript pour l'interface utilisateur.

`circom` est un langage de description de circuits arithmétiques conçu spécifiquement pour la construction de preuves zk-SNARK. Il est aujourd'hui l'une des solutions les plus répandues pour modéliser des circuits de vérification dans des contextes variés. Sa syntaxe, bien que rigide, permet d'exprimer des relations complexes entre données privées et publiques de manière concise et déterministe. Un des avantages de `circom` réside dans la clarté de son paradigme fonctionnel : chaque circuit est une fonction pure, ce qui facilite les tests, la composition modulaire et la réutilisabilité du code. L'outil s'intègre naturellement à `snarkjs`, et bénéficie d'un support actif de la communauté, notamment grâce à son rôle central dans des projets blockchain tels que Tornado Cash ou Semaphore.

`snarkjs` est un outil en ligne de commande développé pour accompagner `circom` dans la manipulation des preuves. Il propose une panoplie de fonctions permettant de compiler les circuits, exécuter le trusted setup (`powers of tau`), générer des preuves à partir d'inputs, vérifier leur validité et exporter les données nécessaires à une vérification côté client (notamment via des fichiers JSON ou Solidity). Cette boîte à outils est essentielle pour déployer des preuves dans un environnement de développement complet. Un atout non négligeable de `snarkjs` réside dans sa capacité à générer des verifiers en JavaScript ou Solidity, ce qui ouvre la voie à des vérifications locales dans un navigateur ou dans un smart contract.

Nous avons également fait le choix de JavaScript pour la mise en place de l'interface utilisateur. Ce langage, largement adopté pour le développement web, permet d'interfacer facilement les fichiers générés par `snarkjs` avec une application interactive. Les entrées utilisateur sont collectées via CLI, puis transformées pour correspondre au format attendu par les circuits (e.g. chaînes ASCII converties en entiers). Cette flexibilité facilite le prototypage rapide tout en maintenant

une compatibilité directe avec les outils de preuve.

Enfin, nous avons utilisé des scripts Node.js pour automatiser le pipeline de génération : compilation des circuits, génération de la preuve, export du vérificateur, etc. Cela permet d'enchaîner efficacement les différentes étapes de production et de test.

Ce choix d'outillage, bien que centré sur l'écosystème `circom`/`snarkjs`, nous a offert une grande souplesse dans le développement du projet, tout en restant cohérent avec les standards actuels du domaine des ZKP. Les principaux inconvénients observés concernent la complexité de débogage dans `circom`, les limitations sur les chaînes de caractères (taille fixe, encodage), et le manque de benchmarks intégrés. Ces limites sont toutefois compensées par la transparence des outils, leur documentation et la possibilité d'intégration dans des systèmes plus vastes à terme.

4 Expérimentations et cas d'usage simulés

Deux scénarios ont été implémentés et testés : la vérification de la majorité légale, et la preuve de possession d'un permis de conduire. Ces cas représentent deux situations fréquentes dans lesquelles une entité (boîte de nuit, plateforme de vote, force de l'ordre) souhaite s'assurer d'une information sans en demander plus que nécessaire.

L'utilisateur interagit avec une application JS en CLI dans laquelle il saisit ses données. Ces données sont ensuite utilisées pour générer une preuve cryptographique. À aucun moment les données sensibles ne sont stockées sur un serveur ou transmises à un tiers, ce qui garantit une confidentialité totale.

Nous n'avons pas encore finalisé l'intégration du système de preuve dans un format exploitable de bout en bout (ex. téléchargement de fichier de preuve, affichage JSON). De même, aucun test de performance approfondi n'a été réalisé. Toutefois, l'infrastructure actuelle permet de simuler l'ensemble du processus avec des résultats conformes aux attentes.

5 Contraintes et limitations du système

Malgré les qualités intrinsèques du protocole zk-SNARK et la pertinence de l'implémentation choisie, notre système présente plusieurs contraintes et limitations structurelles, fonctionnelles et techniques.

Une première limitation concerne les circuits eux-mêmes. En effet, `circom` impose des contraintes fortes sur le traitement des chaînes de caractères, qui doivent être encodées sous forme d'entiers. Cela réduit la flexibilité dans la gestion des entrées textuelles, en particulier pour des champs tels que le nom ou le prénom de l'utilisateur. Ces données doivent être transformées au préalable et respecter des longueurs fixes, ce qui peut entraîner des problèmes de compatibilité ou d'ergonomie côté utilisateur.

Par ailleurs, une limitation importante est l'absence d'un mécanisme natif d'expiration temporelle des preuves. Une fois générée, une preuve reste valide indéfiniment, sauf si une vérification externe est introduite (e.g. horodatage ou jeton temporel). Cette lacune pose des questions évidentes de sécurité dans le cas où des attributs sont susceptibles de changer (comme l'âge ou la validité d'une autorisation). Nous considérons donc l'ajout d'un tel mécanisme comme une évolution nécessaire dans les travaux futurs.

Une autre contrainte tient au manque de flexibilité des circuits une fois compilés et liés à un trusted setup. Toute modification du circuit nécessite de relancer entièrement la phase 2 du trusted setup, ce qui représente une opération coûteuse et délicate. Cette rigidité complique l'ajout de nouvelles règles de vérification ou l'adaptation à d'autres types de justificatifs (e.g. nationalité, résidence, statut professionnel...).

Nous avons également identifié une absence de mécanismes de gestion d'identité persistante. Le système repose sur des entrées ponctuelles, ce qui empêche toute forme de session utilisateur

ou de continuité d'identité dans le temps, sauf à reconstruire un protocole plus complexe de gestion des credentials. Cette limitation restreint les cas d'usage à des vérifications ponctuelles.

Enfin, bien que nous ayons porté une attention particulière à la clarté du code et à la simplicité d'usage de l'interface, celle-ci reste sommaire. Elle mériterait d'être renforcée en matière d'ergonomie, de sécurité (prévention des attaques XSS, gestion des entrées malformées, etc.) et d'intégration dans des systèmes plus larges (comme des portails d'accès ou des API sécurisées).

En résumé, ces limitations ne remettent pas en cause la validité cryptographique de notre approche, mais elles en restreignent le champ d'application. Elles constituent autant de pistes concrètes d'amélioration et d'extension à considérer pour faire évoluer notre système vers une solution plus robuste, adaptable et prête à l'intégration en production.

6 Applications pratiques et potentialités d'usage

Le système que nous avons conçu trouve des applications dans de nombreux domaines. La vérification de la majorité peut servir dans les bars, les discothèques, les plateformes de jeux en ligne ou les services soumis à des restrictions d'âge. La preuve de détention d'un permis de type A est utile lors de contrôles routiers automatisés, de location de véhicules spécifiques ou de démarches administratives.

Plus généralement, le cadre de notre solution peut être étendu à d'autres types de preuves : preuve de nationalité, de résidence, de revenu, de statut professionnel. Dans tous les cas, le principe reste le même : démontrer une propriété d'une donnée privée sans révéler cette donnée.

Les systèmes KYC (Know Your Customer) pourraient bénéficier de cette approche, notamment dans les domaines bancaire, assurantiel ou immobilier, où la vérification d'identité est obligatoire mais ne devrait pas impliquer une exposition inutile des données personnelles.

7 Perspectives et évolutions futures

Plusieurs axes d'amélioration sont envisagés pour faire évoluer notre prototype vers une solution robuste et déployable en production. Tout d'abord, l'interface sera finalisée afin d'intégrer la génération complète de la preuve, son export, son affichage et éventuellement son partage sécurisé.

Ensuite, nous comptons introduire des mécanismes d'authentification des preuves via des signatures numériques, afin de garantir que la preuve provient bien de l'individu qu'elle concerne. L'utilisation d'une blockchain publique pour enregistrer les preuves ou vérifier leur authenticité est également envisagée.

Un autre point fondamental à traiter est l'introduction d'une date d'expiration dans les preuves générées. Cette fonctionnalité permettrait de limiter la réutilisation abusive d'une preuve vieille de plusieurs mois ou années, et pourrait être incluse directement dans le circuit comme contrainte vérifiable.

Enfin, des tests de performance doivent être menés pour évaluer la charge, la scalabilité, et la robustesse de l'ensemble du système. Cela permettra d'envisager une mise en production, sous forme d'API publique ou d'outil intégré à une infrastructure existante.