RÉPUBLIQUE DU SÉNÉGAL
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR, DE LA RECHERCHE ET DE L'INNOVATION
UNIVERSITÉ GASTON BERGER



L'EXCELLENCE AU SERVICE DU DÉVELOPPEMENT.

UFR : Sciences Appliquées et de Technologie
DÉPARTEMENT : Mathématiques
OPTION : Cryptographie, Codage, Sécurité et Applications (CCA)

MÉMOIRE

présenté par

Amadou FAYE

pour l'obtention du diplôme de

Master en Cryptographie

SUJET :

ÉTUDE DES MÉCANISMES DE PREUVE À DIVULGATION NULLE DE CONNAISSANCE EN CRYPTOGRAPHIE

Soutenu à *Saint-Louis, le 22 Juin 2026* devant le jury composé de :

Président	**: Pr Augustin P.SARR**	**UFR SAT UGB**
Examinateur	**: Pr. Albert DIOMPY**	**UFR SAT UGB**
Encadrant	**: Pr. Cherif DIALLO**	**UFR SAT UGB**

Année académique : 2024-2025

À mes parents,
pour m'avoir appris que le savoir
est la seule richesse que l'on ne perd jamais.

À ma femme,
pour sa patience sans faille,
sa confiance dans les moments de doute,
et sa présence lumineuse à chaque étape de ce travail.

À ma famille toute entière,
dont le soutien silencieux et constant
a été le socle sur lequel ce mémoire s'est construit.

« La connaissance est une lumière qui éclaire
celui qui la porte et ceux qui l'entourent. »

REMERCIEMENTS

J'exprime ma profonde gratitude envers Dieu, le Tout-Puissant, qui m'a accordé la santé, la patience, la force et la persévérance nécessaires pour mener à bien ce travail.

Je tiens à remercier chaleureusement mon directeur de mémoire, **Pr. Cherif DIALLO**, pour son accompagnement constant, sa grande disponibilité, sa rigueur scientifique et ses conseils toujours éclairés. Son encadrement précieux a été déterminant dans l'aboutissement de ce projet de recherche.

Mes remerciements les plus respectueux vont également au président du jury, **Pr. Augustin P.SARR**, pour l'honneur qu'il nous fait en acceptant de présider cette soutenance. J'adresse aussi ma reconnaissance au **Pr. Albert DIOMPY**, pour avoir accepté de siéger parmi les membres du jury et d'examiner mon travail.

Je tiens à exprimer ma vive gratitude envers l'ensemble du personnel et le corps enseignant de l'Université Gaston Berger, en particulier ceux du département de Mathématiques et Applications : **Pr. Mouhamed Ben MAAOUIA, Pr. Madicke DIOP, Pr. Amadou TALL et Pr. Jean Marie DEMBÉLÉ**, pour la haute qualité de leur enseignement, leur professionnalisme et leur dévouement tout au long de ma formation. Un remerciement tout particulier à **Mme Bernadette FAYE**, pour la rigueur et la bienveillance de son enseignement qui m'ont guidé tout au long de ce travail.

Je remercie de tout cœur ma famille, en particulier ma mère, **Boury DIENG**, et mon père, **Moussa FAYE**, dont l'amour inconditionnel et les prières quotidiennes ont tracé mon chemin. À mes parents, mes frères et mes sœurs : **Lamine FAYE, Cheikh FAYE, Cheikh LO FAYE, Ibrahima FAYE, Ndeye FAYE, Djilane FAYE, Yacine SARR, Diayenaba FAYE, Amy DIONE, Yatouti KOYATÉ, Mame Diarra FAYE, Khoudia DARAMÉ, Khady NDIAYE et Aicha DIONE**, merci pour votre soutien constant, votre immense patience et vos encouragements sans faille.

Je témoigne toute mon affection à mes amis, **Bassirou CISSÉ, Amdy Moustapha**

SEYE, Abdoulaye FAYE, Taib FALL, Cheikh LEYE et Diaba SYLLA, pour leur présence fidèle et leur soutien indéfectible.

J'exprime également ma gratitude à mes camarades de promotion et collègues, **Djiby CAMARA, Cheikh Omar DIARRA, Abdou Fatim DIOP, Cheikh Sidaty DIOP, Ababacar Sadikh DIOUF, Djagly FALL, Adama SENE et Abdou Karim NDIOUR**, pour la qualité de notre collaboration, leur esprit d'entraide et leur camaraderie.

J'adresse un merci bienveillant à mes compagnons de chambre, **Serigne Mbacké SÈYE, Augustin MANGA, Oumar DIOUF, Assane DIOUF et Sohibou DEMBLÉ**. Un hommage spirituel et fraternel à **Dieuwrine Gora DIOP**, un frère particulier qui m'a toujours guidé avec sagesse sur le droit chemin, ainsi qu'à tous mes collaborateurs.

Enfin, j'adresse mes remerciements les plus sincères à toutes celles et ceux qui, de près ou de loin, ont contribué à la réalisation de ce mémoire. Un merci infiniment tendre à mon épouse, **Soxna SAKHO**, qui n'a ménagé aucun effort pour me soutenir, m'épauler et m'inspirer dans les moments de doute. C'est avec une immense fierté et un amour profond que je lui dédie ce travail.

$$\Pi_{ZK} : \mathsf{Prove}(x \in L) \wedge \neg\mathsf{Reveal}(x)$$

Ce mémoire est dédié à ma famille entière,
dont le soutien sans faille est le plus beau fruit de ce travail.

RÉSUMÉ

Les preuves à divulgation nulle de connaissance (*Zero-Knowledge Proofs*, ZKP) constituent une primitive cryptographique fondamentale. Elles permettent de prouver la validité d'une assertion sans révéler aucune information sur le témoin (secret) qui la justifie. Ce mémoire propose une étude complète de cette technique, depuis ses fondements mathématiques jusqu'à la validation expérimentale d'un protocole original.

Les quatre premiers chapitres établissent les bases nécessaires : structures algébriques et problèmes difficiles (logarithme discret, Diffie-Hellman, factorisation), formalisation des trois propriétés fondamentales (complétude, robustesse, divulgation nulle), analyse comparative des protocoles interactifs (Schnorr, Feige-Fiat-Shamir) et non-interactifs (zk-SNARKs, zk-STARKs, Bulletproofs), puis mise en perspective des applications et limites actuelles.

Le cinquième chapitre présente la contribution originale de ce travail : Π_{VoteZK}, un protocole de vote électronique à divulgation nulle de connaissance. Construit sur quatre primitives complémentaires (Schnorr–Fiat-Shamir, engagement de Pedersen, OR-proof CDS et arbre de Merkle). Il satisfait cinq propriétés formelles : éligibilité, unicité, validité binaire des bulletins, confidentialité et vérifiabilité. Les preuves de sécurité sont conduites dans le modèle de l'oracle aléatoire. L'implémentation en Python, validée par dix tests unitaires et des benchmarks empiriques, confirme des complexités $\mathcal{O}(N)$ pour la génération et la vérification, et $\mathcal{O}(\sqrt{N})$ pour le dépouillement.

Mots-clés : preuves à divulgation nulle de connaissance, logarithme discret, protocole de Schnorr, transformation de Fiat-Shamir, engagement de Pedersen, OR-proof CDS, arbre de Merkle, vote électronique, oracle aléatoire.

ABSTRACT

Zero-Knowledge Proofs (ZKPs) are a fundamental cryptographic primitive that allow one party to prove the validity of a statement to another without revealing any information beyond the truth of that statement. This thesis provides a comprehensive study of this technique, from its mathematical foundations to the experimental validation of an original protocol.

The first four chapters establish the necessary background : algebraic structures and hard problems (discrete logarithm, Diffie-Hellman, factorization), formalization of the three fundamental properties (completeness, soundness, zero-knowledge), comparative analysis of interactive protocols (Schnorr, Feige-Fiat-Shamir) and non-interactive ones (zk-SNARKs, zk-STARKs, Bulletproofs), followed by a discussion of practical applications and current limitations.

The fifth chapter presents the original contribution of this work : Π_{VoteZK}, an electronic voting protocol with zero-knowledge properties. Built on four complementary primitives (Schnorr-Fiat-Shamir, Pedersen commitment, CDS OR-proof and Merkle tree), it satisfies five formal properties : eligibility, uniqueness, binary validity of ballots, confidentiality and verifiability. Security proofs are conducted in the random oracle model. The Python implementation, validated by ten unit tests and empirical benchmarks, confirms $\mathcal{O}(N)$ complexity for proof generation and verification, and $\mathcal{O}(\sqrt{N})$ for tallying.

Keywords : zero-knowledge proofs, discrete logarithm, Schnorr protocol, Fiat-Shamir transform, Pedersen commitment, CDS OR-proof, Merkle tree, electronic voting, random oracle model.

NOTATIONS ET ABRÉVIATIONS

Abréviations

BSGS	*Baby-Step Giant-Step* (algorithme de Shanks)
CDH	*Computational Diffie-Hellman Problem*
CDS	Cramer–Damgård–Schoenmakers (OR-proof)
DDH	*Decisional Diffie-Hellman Problem*
DLP	*Discrete Logarithm Problem*
FFS	Feige–Fiat–Shamir
IZKP	*Interactive Zero-Knowledge Proof*
LCI	Loi de Composition Interne
MTD	Machine de Turing Déterministe
NFS	*Number Field Sieve*
NIZKP	*Non-Interactive Zero-Knowledge Proof*
NIST	*National Institute of Standards and Technology*
OWF	*One-Way Function* (fonction à sens unique)
PPT	Machine de Turing Probabiliste Polynomiale
ROM	*Random Oracle Model*
SHA	*Secure Hash Algorithm*
ZKP	*Zero-Knowledge Proof*
zk-SNARK	*Zero-Knowledge Succinct Non-Interactive Argument of Knowledge*
zk-STARK	*Zero-Knowledge Scalable Transparent Argument of Knowledge*

Notations mathématiques

$\mathbb{G}$	Groupe cyclique
$\mathbb{Z}_q$	Anneau des entiers modulo q
$\mathbb{Z}_p^*$	Groupe multiplicatif des entiers modulo p
$\mathbb{F}_p$	Corps fini à p éléments
g	Générateur du groupe cyclique
q	Ordre du sous-groupe premier
p	Premier sûr, $q = (p-1)/2$
λ	Paramètre de sécurité
$\mathsf{negl}(\lambda)$	Fonction négligeable en λ
$x \xleftarrow{\$} S$	x tiré uniformément au hasard dans S
$\log_g h$	Logarithme discret de h en base g
$H(\cdot)$	Fonction de hachage cryptographique
$\|$	Concaténation d'octets
Π_{VoteZK}	Protocole de vote électronique ZKP
pp	Paramètres publics $(p, q, g, h, \varepsilon)$
sk, pk	Clé secrète et clé publique de l'électeur
nul	Nullifieur $(H(\mathsf{sk}\|\varepsilon))$
C	Engagement de Pedersen du bulletin
π	Preuve ZKP complète
(t, c, s)	Triplet engagement–défi–réponse
$\mathcal{P}$, $\mathcal{V}$, $\mathcal{S}$	Prouveur, Vérificateur, Simulateur
$\approx_c$	Indistinguabilité computationnelle
$\top$ / $\bot$	Accepté / Rejeté

TABLE DES MATIÈRES

Remerciements **ii**

Résumé **iv**

Abstract **v**

Notations et abréviations **vi**

Introduction Générale **1**

1 Fondements Mathématiques et Primitives Cryptographiques **4**

Introduction 4

1.1 Structures algébriques fondamentales 4

1.1.1 Loi de composition interne 4

1.1.2 Groupes 5

1.1.3 Groupes cycliques 5

1.1.4 Corps 5

1.1.5 Courbes elliptiques 6

1.1.6 Couplages bilinéaires 7

1.2 Problèmes algorithmiques difficiles 8

1.2.1 La famille du logarithme discret (DLP) 8

1.2.2 Le problème de Diffie-Hellman 9

1.2.3 La factorisation 10

1.3 Théorie de la complexité calculatoire 10

1.3.1 Modèles de calcul 10

1.3.2 Classes de complexité 12

1.4 Primitives cryptographiques 14

1.4.1 Fonctions à sens unique 14

1.4.2 Modèle de l'oracle aléatoire (ROM) 15
1.4.3 Fonctions de hachage cryptographiques 15
1.4.4 Schémas d'engagement cryptographique 17
1.4.5 Σ-protocoles 18
Conclusion 19

2 Fondements Théoriques et Mécanismes des ZKPs 20
Introduction 20
2.1 Analogie pédagogique : La grotte d'Ali Baba 20
2.1.1 Le protocole en trois phases 20
2.1.2 Analyse de la fiabilité 22
2.2 Propriétés fondamentales des ZKPs 22
2.2.1 Complétude (*Completeness*) 22
2.2.2 Robustesse (*Soundness*) 23
2.2.3 Divulgation nulle (*Zero-Knowledge*) 23
2.3 Définition formelle des ZKPs 23
2.4 Processus de construction des ZKPs 24
2.4.1 Entités et rôles 24
2.4.2 Formalisation du problème 24
2.4.3 Choix du paradigme 24
2.4.4 Schéma en trois tours 25
2.4.5 Preuve des trois propriétés 25
Conclusion 25

3 Modèles, Hypothèses de sécurité et Analyse comparative des ZKPs 26
Introduction 26
3.1 Le modèle interactif (IZKP) 26
3.1.1 Le protocole d'identification de Schnorr 27
3.1.2 Le protocole de Feige–Fiat–Shamir (FFS) 27
3.2 Le modèle non-interactif (NIZKP) 28
3.2.1 Les protocoles zk-SNARKs 29
3.2.2 Les zk-STARKs 29
3.2.3 Bulletproofs 29
3.3 Hypothèses de sécurité des protocoles ZKP 29
3.3.1 Protocoles ZKP interactifs 29
3.3.2 Protocoles ZKP non interactifs 30
3.4 Analyse comparative des protocoles ZKP 32
3.4.1 IZKPs contre NIZKPs 32
3.4.2 Analyse comparative des IZKPs 32
3.4.3 Analyse comparative des protocoles NIZKP modernes 33

Conclusion 34

4 Applications, Limites, Perspectives et Ouvertures des ZKPs 36

Introduction 36

4.1 Applications des ZKPs 36

4.1.1 Transactions financières sensibles 36

4.1.2 Audits et vérifications 37

4.1.3 Partage de données médicales 37

4.1.4 Sécurité des mots de passe 37

4.1.5 Stockage et partage de fichiers 37

4.2 Limites des ZKPs 37

4.2.1 La charge computationnelle et le coût de génération 38

4.2.2 La configuration de confiance (*Trusted Setup*) 38

4.2.3 Les compromis de vérification et de stockage 38

4.2.4 La menace de l'informatique quantique 38

4.3 Perspectives d'évolution des ZKPs 39

4.3.1 Vote électronique et démocratie numérique : le contexte sénégalais 39

4.3.2 Gouvernance résiliente et transparente 42

4.3.3 Sécurisation de l'Internet des Objets (IoT) 42

4.3.4 Applications dans le domaine de l'éducation 42

4.4 Ouverture des ZKPs 42

4.4.1 Standardisation et interopérabilité des protocoles 42

4.4.2 L'émergence de l'Identité Numérique Décentralisée 43

4.4.3 Défis éducatifs et souveraineté technologique 43

4.5 Conclusion 43

5 Π_{VoteZK} : de la Formalisation à la Validation Expérimentale 45

Introduction 45

5.1 Spécification formelle du problème de vote électronique 46

5.1.1 Modèle d'adversaire et propriétés requises 46

5.1.2 Relation de preuve formelle 47

5.2 Conception du protocole Π_{VoteZK} 47

5.2.1 Choix de conception et justification 47

5.2.2 Architecture du protocole 48

5.2.3 Phase 1 — Setup : génération des paramètres 48

5.2.4 Phase 2 — Registration : inscription des électeurs 49

5.2.5 Phase 3 — Voting : génération de la preuve 49

5.2.6 Phase 4 — Tallying : vérification et décompte 50

5.3 Implémentation en Python 51

5.3.1 Architecture du code 51

5.3.2 Module `params.py` . . . 52
5.3.3 Module `merkle.py` . . . 54
5.3.4 Module `or_proof.py` . . . 55
5.3.5 Module `prover.py` . . . 57
5.3.6 Module `verifier.py` . . . 58
5.3.7 Module `tally.py` . . . 59
5.4 Tests expérimentaux et analyse de performance . . . 60
5.4.1 Tests unitaires . . . 61
5.4.2 Module `test_votezk.py` . . . 61
5.4.3 Protocole de mesure . . . 66
5.4.4 Analyse et interprétation . . . 68
5.5 Analyse de sécurité . . . 69
5.5.1 Complétude . . . 69
5.5.2 Robustesse . . . 70
5.5.3 Validité binaire — OR-proof CDS . . . 73
5.5.4 Divulgation nulle . . . 74
5.5.5 Résistance aux attaques spécifiques au vote . . . 76
5.6 Limites du protocole et perspectives d'amélioration . . . 76
5.6.1 Limites identifiées . . . 77
5.6.2 Perspectives d'amélioration . . . 77
Conclusion . . . 77

Conclusion générale **79**

TABLE DES FIGURES

2.1 Phase d'engagement dans le protocole interactif [15]. 21
2.2 Émission du défi par le vérificateur [15]. 21
2.3 Vérification de la réponse [15]. 22

3.1 Architecture d'un protocole ZKP interactif [20]. 27
3.2 Architecture d'un protocole ZKP non interactif [20]. 28

$$\Pi_{ZK} : \mathsf{Prove}(x \in L) \wedge \neg\mathsf{Reveal}(x)$$

LISTE DES TABLEAUX

1.1 Courbes elliptiques utilisées dans les principaux protocoles ZKP 7
1.2 Fonctions de hachage utilisées dans les protocoles ZKP 16

4.1 Analyse comparative des solutions de vote électronique et apport de Π_{VoteZK}. 41

5.1 Primitives du protocole Π_{VoteZK} et leurs justifications 48
5.2 Suite de tests unitaires du protocole Π_{VoteZK} (10 tests). 61
5.3 Résultats bruts du benchmark des performances (Π_{VoteZK}). 67
5.4 Décomposition de la taille théorique et validation empirique de $|\pi|$. . . 68
5.5 Résistance de Π_{VoteZK} aux attaques du vote électronique. 76

TABLE DES CODES

5.1 `params.py` — génération des paramètres publics `pp`. 52
5.2 `merkle.py` — arbre de Merkle et vérification de chemin. 54
5.3 `or_proof.py` — génération et vérification de la OR-proof CDS [30]. . . 55
5.4 `prover.py` — génération de la preuve π. 57
5.5 `verifier.py` — vérification de la preuve en 6 étapes. 58
5.6 `tally.py` — décompte homomorphe et algorithme BSGS. 60
5.7 `test_votezk.py` — suite de dix tests unitaires couvrant les cinq propriétés requises de Π_{VoteZK}. 61
5.8 Sortie `python test_votezk.py` — 10 tests PASSED. 65
5.9 Script de benchmark unifié — avec OR-proof intégrée. 66

INTRODUCTION GÉNÉRALE

La cryptographie moderne ne se limite plus à chiffrer des messages. Pendant des décennies, sa mission première était simple : transformer un texte clair en un texte illisible pour protéger la confidentialité d'une communication. Mais aujourd'hui, les enjeux du monde numérique exigent une approche bien plus subtile : *comment prouver qu'une information est vraie sans jamais la révéler ?*

Ce paradoxe est au cœur d'une transformation majeure de nos infrastructures numériques. À l'ère de la collecte massive de données et de la surveillance, la protection de la vie privée est devenue un enjeu politique et sociétal de premier plan [19]. Qu'il s'agisse des réglementations comme le RGPD ou de l'essor des systèmes décentralisés, le constat reste le même : il est désormais nécessaire de concevoir des outils capables de vérifier des propriétés logiques ou mathématiques sans avoir à exposer les données sensibles sous-jacentes.

C'est précisément le rôle des *preuves à divulgation nulle de connaissance* (*Zero-Knowledge Proofs*, ZKP), introduites par Goldwasser, Micali et Rackoff en 1985 [36]. Ce puissant concept permet à un prouveur de convaincre un vérificateur de la validité d'une affirmation (la connaissance d'un secret, l'appartenance à une liste, la validité d'une transaction) sans lui transmettre la moindre information. Leurs applications vont bien au-delà du cadre académique : authentification sans mot de passe, transactions financières privées, identité numérique souveraine et, au centre de ce travail, le **vote électronique confidentiel**.

Le vote en ligne cristallise parfaitement ces tensions. Un scrutin numérique doit impérativement garantir deux exigences qui semblent pourtant contradictoires : l'anonymat absolu de l'électeur et la transparence totale du décompte pour éviter toute fraude. Le verrou scientifique que ce mémoire cherche à lever est donc le suivant : *comment concevoir un protocole de vote électronique simultanément confidentiel, valide, unique et vérifiable, sans qu'aucune de ces garanties ne suppose la divulgation du vote lui-même ?*

Plus précisément, notre système doit satisfaire cinq propriétés formelles et distinctes.

L'éligibilité assure que seuls les électeurs inscrits peuvent voter, sans pour autant révéler leur identity. L'unicité garantit que chaque électeur ne vote qu'une seule fois, sans possibilité d'être tracé. La validité binaire contraint le bulletin à appartenir strictement à $\{0, 1\}$ pour représenter un choix binaire, ce qui est prouvé cryptographiquement sans ouvrir l'enveloppe. Enfin, la confidentialité rend le contenu opaque pour tout adversaire, y compris pour l'autorité de scrutin, tandis que la vérifiabilité permet à n'importe quel observateur externe de valider l'exactitude du calcul final.

Ces dernières années, des technologies ZKP génériques et très efficaces ont émergé, comme les *zk-SNARKs* [16] (preuves ultra-courtes), les *zk-STARKs* [12] (résistants au calcul quantique et sans tiers de confiance) ou les *Bulletproofs* [5] (adaptés aux plages de données). Cependant, ces solutions présentent des limites pour un contexte de vote : les SNARKs requièrent une phase d'initialisation risquée (*trusted setup*), les STARKs génèrent des preuves trop volumineuses en bande passante, et les Bulletproofs s'appuient sur le logarithme discret traditionnel, vulnérable à long terme face à l'algorithme de Shor [29].

Ce travail propose de combler cet espace en concevant Π_{VoteZK}, un protocole de vote électronique sur mesure. Son architecture repose sur l'articulation de quatre primitives éprouvées : le **protocole de Schnorr** combiné à la **transformation de Fiat-Shamir** [6, 1] pour la non-interactivité ; l'**engagement de Pedersen** [37] pour ses propriétés d'homomorphisme additif permettant le dépouillement à l'aveugle ; la **OR-proof CDS** [30] pour valider le choix binaire ; et enfin un **arbre de Merkle avec SHA-256** pour authentifier l'éligibilité à moindre coût.

La contribution de ce mémoire est double. Sur le plan **théorique**, nous formalisons la relation de preuve et démontrons trois théorèmes essentiels : complétude, robustesse et divulgation nulle dans le modèle de l'oracle aléatoire [23]. Sur le plan **pratique**, le protocole a été entièrement codé en Python. Une série de dix tests unitaires et des benchmarks empiriques confirment la viabilité de l'approche et la cohérence entre la complexité théorique et les performances réelles observées.

Pour guider le lecteur à travers ce travail, le manuscrit est organisé selon une progression logique en cinq chapitres, articulés autour de la transition entre théorie fondamentale et application pratique.

Le premier chapitre pose le socle algébrique indispensable à la sécurité de nos protocoles. Nous y étudions les structures mathématiques fondamentales, le problème du logarithme discret, les fonctions de hachage et les schémas d'engagement.

Le deuxième chapitre formalise les propriétés essentielles des preuves à divulgation nulle de connaissance. Nous y définissons les notions de complétude, de robustesse et de simulation, tout en jetant les bases formelles des Σ-protocoles.

Le troisième chapitre analyse le paysage des constructions ZKP de la littérature. Il oppose les modèles interactifs aux approches non-interactives, avant de dresser un

comparatif critique des grandes familles modernes que sont les SNARKs, les STARKs et les Bulletproofs.

Le quatrième chapitre élargit la perspective aux applications et aux limites pratiques des ZKP. Nous y explorons leur impact concret dans la gestion de l'identité, de la finance et de la santé, tout en soulevant les défis liés à la charge de calcul et à la menace quantique.

Enfin, le cinquième chapitre constitue le cœur original de ce mémoire. Il détaille la spécification algorithmique de notre protocole Π_{VoteZK}, ses preuves formelles de sécurité, son implémentation en Python ainsi que les résultats de nos évaluations expérimentales.

$$\Pi_{ZK} : \mathsf{Prove}(x \in L) \wedge \neg\mathsf{Reveal}(x)$$

CHAPITRE 1

FONDEMENTS MATHÉMATIQUES ET PRIMITIVES CRYPTOGRAPHIQUES

Introduction

Les preuves à divulgation nulle de connaissance ne surgissent pas du néant. Elles reposent sur un édifice mathématique et cryptographique dont chaque brique a été posée avec rigueur. Ce chapitre met en évidence les fondations : structures algébriques, problèmes difficiles, théorie de la complexité, primitives cryptographiques. Il introduit également les outils transversaux (modèle de l'oracle aléatoire et Σ-protocoles) qui seront mobilisés dès les chapitres suivants.

1.1 Structures algébriques fondamentales

Garantir les propriétés de complétude et de robustesse des ZKP exige un cadre mathématique précis. Les travaux fondateurs [36, 27] le montrent clairement : les interactions cryptographiques tirent leur force de propriétés algébriques spécifiques, en particulier celles des groupes cycliques et des corps finis. Ces structures fournissent le terrain nécessaire à la construction de problèmes dits « difficiles », dont la résolution sans information privilégiée est statistiquement improbable [40].

1.1.1 Loi de composition interne

Définition 1.1 (Loi de composition interne)**.** Soit E un ensemble. On appelle *loi de composition interne* (LCI) sur E une application qui à tout couple (a, b) d'éléments de E associe un unique élément c appartenant également à E.

Formellement, si on note $*$ cette loi, elle est définie par :

$$* : E \times E \to E$$

$$(a, b) \mapsto a * b$$

1.1.2 Groupes

Définition 1.2 (Groupe)**.** Un *groupe* est un couple $(\mathbb{G}, *)$ où $\mathbb{G}$ est un ensemble non vide et $*$ est une loi de composition interne satisfaisant :

i. **Associativité :** $(a * b) * c = a * (b * c)$ pour tout $a, b, c \in \mathbb{G}$.

ii. **Élément neutre :** $\exists\, e \in \mathbb{G}$, $a * e = e * a = a$ pour tout a.

iii. **Symétrique :** $\forall a \in \mathbb{G}$, $\exists\, a^{-1} \in \mathbb{G}$, $a * a^{-1} = e$.

Si de plus $a * b = b * a$ pour tout a, b, le groupe est dit *abélien* (commutatif).

1.1.3 Groupes cycliques

Définition 1.3 (Groupe cyclique)**.** Un groupe $\mathbb{G}$ est dit *cyclique* s'il existe un élément $g \in \mathbb{G}$, appelé *générateur*, tel que [18] :

$$\mathbb{G} = \langle g \rangle = \{g^0, g^1, g^2, \ldots, g^{|\mathbb{G}|-1}\}$$

L'entier $|\mathbb{G}|$ est l'*ordre* du groupe. Si cet ordre est un nombre premier q, tout élément non-neutre de $\mathbb{G}$ est un générateur.

Exemple 1.1 (Groupe multiplicatif $\mathbb{Z}_p^*$)**.** Pour un nombre premier p, l'ensemble $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ muni de la multiplication modulo p est un groupe cyclique d'ordre $p-1$ [40]. Si p est choisi tel que $q = (p-1)/2$ soit également premier (nombre premier sûr), le sous-groupe d'ordre q engendré par un générateur g est le cadre habituel des protocoles de Schnorr [6].

1.1.4 Corps

Définition 1.4 (Corps)**.** Un *corps* est un ensemble $\mathbb{F}$ muni de deux lois de composition internes, notées $+$ (addition) et $\cdot$ (multiplication), satisfaisant les conditions suivantes [33] :

i. $(\mathbb{F}, +)$ **est un groupe abélien** : l'addition est associative, commutative, possède un élément neutre 0, et tout élément possède un opposé.

ii. $(\mathbb{F} \setminus \{0\}, \cdot)$ **est un groupe** : la multiplication est associative, possède un élément neutre $1 \neq 0$, et tout élément non nul possède un inverse.

iii. **Distributivité** : la multiplication est distributive par rapport à l'addition :

$$\forall a, b, c \in \mathbb{F}, \quad a \cdot (b + c) = a \cdot b + a \cdot c$$

Si l'ensemble $\mathbb{F}$ contient un nombre fini d'éléments, on parle de **corps fini** (ou corps de Galois). On le note alors généralement $\mathbb{F}_q$ où q est la puissance d'un nombre premier.

Exemple 1.2 (Corps fini $\mathbb{F}_p$)**.** Pour un nombre premier p, le *corps fini* $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ est l'ensemble $\{0, 1, \ldots, p-1\}$ muni de l'addition et de la multiplication modulo p. $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$ est un groupe cyclique d'ordre $p - 1$.

Les corps finis sont le terrain naturel de l'*arithmétique modulaire*, sur laquelle repose l'ensemble des protocoles étudiés dans ce mémoire. L'opération centrale est l'exponentiation modulaire :

$$\mathrm{ExpMod}(g, x, p) = g^x \bmod p \tag{1.1.1}$$

calculable en $O(\log x)$ multiplications par l'algorithme des carrés successifs, mais dont l'inversion (retrouver x à partir de $g^x \bmod p$) est supposée difficile.

1.1.5 Courbes elliptiques

Les groupes multiplicatifs classiques ne sont pas les seuls terrains de jeu. Les courbes elliptiques offrent une alternative séduisante : des paramètres de sécurité bien plus compacts pour un niveau de protection équivalent [8]. L'idée est simple. On change de structure et on gagne en efficacité.

Définition 1.5 (Courbe elliptique sur $\mathbb{F}_p$)**.** Une *courbe elliptique* sur un corps fini $\mathbb{F}_p$ est l'ensemble des points $(x, y) \in \mathbb{F}_p^2$ satisfaisant l'équation de Weierstrass courte :

$$y^2 \equiv x^3 + ax + b \pmod{p}, \quad \text{avec } 4a^3 + 27b^2 \not\equiv 0 \pmod{p}$$

augmenté d'un point à l'infini $\mathcal{O}$ servant d'élément neutre.

L'ensemble des points d'une courbe elliptique forme un groupe abélien pour une loi d'addition géométrique. Deux familles méritent une attention particulière dans le contexte des ZKP : les courbes adaptées aux signatures, comme `Curve25519` [8], et les courbes *pairing-friendly* essentielles aux SNARKs, comme `BN254` et `BLS12-381` [41]. Le tableau 1.1 synthétise leurs caractéristiques et domaines d'application :

TABLE 1.1 – Courbes elliptiques utilisées dans les principaux protocoles ZKP

Courbe	**Équation**	**Usage**	**Réf.**
Curve25519	$y^2 = x^3 + 486662x^2 + x$	Ed25519 (Schnorr sur CE), X25519	[8]
BN254	$y^2 = x^3 + b$	zk-SNARKs (Groth16)	[16]
BLS12-381	$y^2 = x^3 + 4$	zk-SNARKs, Ethereum 2.0	[41]

Proposée par Bernstein [8], Curve25519 est particulièrement adaptée aux implémentations pratiques de signatures numériques de type Schnorr sur courbes elliptiques (Ed25519) et à l'échange de clés Diffie-Hellman (X25519) (son premier $p = 2^{255} - 19$ permet des opérations extrêmement rapides sur les processeurs modernes). À noter que le protocole de Schnorr *modulaire classique* [6], présenté au chapitre suivant, opère dans le groupe multiplicatif $\mathbb{Z}_p^*$ avec un sous-groupe d'ordre premier q et non directement sur cette courbe.

1.1.6 Couplages bilinéaires

Les couplages bilinéaires sont la brique spécifique aux zk-SNARKs [16]. Pas de couplages, pas de SNARKs. Leur rôle est de vérifier des relations multiplicatives dans l'exposant sans les révéler.

Définition 1.6 (Couplage bilinéaire)**.** Un *couplage bilinéaire* est une application :

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

où $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$ sont des groupes cycliques d'ordre premier q, satisfaisant :

i. **Bilinéarité :** $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ pour tout $a, b \in \mathbb{Z}_q$.

ii. **Non-dégénérescence :** $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.

iii. **Calculabilité efficace :** e est calculable en temps polynomial.

Remarque 1.1 (Construction pratique des couplages). Les couplages utilisés en pratique (Weil, Tate, Ate optimal) nécessitent des courbes dites *pairing-friendly* [41]

dont les groupes $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ sont soigneusement choisis pour que e soit calculable efficacement. La vérification d'un zk-SNARK Groth16 via $e(A, B) = e([\alpha]_1, [\beta]_2) \cdot \ldots$ est une conséquence directe de la bilinéarité appliquée aux éléments de la clé de vérification [16].

L'équation de vérification d'un zk-SNARK dans la construction de Groth [16] s'exprime sous la forme :

$$e(A, B) = e([\alpha]_1,\, [\beta]_2) \cdot e(L_{\mathsf{pub}},\, [\gamma]_2) \cdot e(C,\, [\delta]_2)$$

où $[\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_2$ sont des éléments de la clé de vérification, A, B, C les éléments de la preuve, et L_{pub} une combinaison linéaire des entrées publiques dans $\mathbb{G}_1$, calculée par le vérificateur.

1.2 Problèmes algorithmiques difficiles

La sécurité d'un système cryptographique ne repose pas sur le secret de l'algorithme. Elle repose sur la *difficulté supposée* de résoudre certains problèmes mathématiques dans un temps raisonnable [26]. Pour les protocoles étudiés ici, trois familles structurent ces défis : la famille du logarithme discret, les problèmes de Diffie-Hellman, et la factorisation.

1.2.1 La famille du logarithme discret (DLP)

Le logarithme discret est la pierre angulaire des preuves à divulgation nulle modernes. Sa force tient à une asymétrie radicale : construire une preuve par exponentiation modulaire est trivial ; retrouver le secret original sans information supplémentaire est hors de portée.

Définition 1.7 (Problème du logarithme discret — DLP)**.** Soient $\mathbb{G}$ un groupe cyclique d'ordre premier q et g un générateur de $\mathbb{G}$. Étant donné un élément $h \in \mathbb{G}$, le *problème du logarithme discret* consiste à trouver l'unique entier $x \in \mathbb{Z}_q$ tel que :

$$g^x = h$$

On note cette opération $x = \log_g h$.

Note sur les niveaux de sécurité réels. Dans le groupe multiplicatif $\mathbb{Z}_p^*$, l'algorithme *Number Field Sieve* (NFS) résout le DLP en temps sous-exponentiel $L_p[1/3, c]$ avec $c \approx 1.923$. Un module p de 128 bits n'offre donc **pas** 128 bits de

sécurité effective : la sécurité tombe à quelques dizaines de bits seulement. Le NIST recommande des modules d'au moins **2048 bits** pour un niveau de sécurité de 112 bits dans $\mathbb{Z}_p^*$ [40, 25]. En pratique, $\lambda = 128$ bits dans ce mémoire est un paramètre de *prototype pédagogique uniquement* ; toute mise en production doit utiliser $p \geq 2048$ bits ou migrer vers des courbes elliptiques (Curve25519, § 1.1.3).

Exemple 1.3 (Asymétrie du DLP dans $\mathbb{F}_{11}$)**.** Considérons le corps fini $\mathbb{F}_{11} = \{0, 1, \ldots, 10\}$ et le groupe multiplicatif associé $\mathbb{F}_{11}^*$ d'ordre $q = 10$. Soit $g = 2$ un générateur de ce groupe.

- **Sens direct (Facile) :** Pour un secret $x = 6$, le calcul de l'élément public $h = g^x$ se fait par exponentiation modulaire :

$$2^6 = 64 \equiv 9 \pmod{11}$$

  Grâce à l'algorithme des carrés successifs, cette opération reste quasi-instantanée même pour des exposants de 256 bits.

- **Sens inverse (Difficile) :** Si l'on donne à un adversaire le résultat $h = 9$, celui-ci doit résoudre $2^x \equiv 9 \pmod{11}$. Sans méthode analytique, il est contraint de tester les puissances successivement :

$$2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8,$$
$$2^4 = 5, \quad 2^5 = 10, \quad 2^6 = 9 \equiv \mathbf{h}$$

L'adversaire trouve $x = 6$ après une recherche exhaustive. L'asymétrie réside dans le fait qu'en augmentant la taille de p (ex : 256 bits), le calcul direct nécessite environ 512 opérations, tandis que l'inversion naïve en nécessiterait 2^{256}, rendant le problème physiquement insoluble.

1.2.2 Le problème de Diffie-Hellman

Le DLP protège un secret individuel. Les problèmes de type Diffie-Hellman vont plus loin. Ils permettent de vérifier des relations entre plusieurs secrets sans en révéler aucun.

Définition 1.8 (Problème Diffie-Hellman computationnel — CDH)**.** Étant donné le triplet $(g, g^a, g^b) \in \mathbb{G}^3$, le problème CDH consiste à calculer l'élément g^{ab}.

Exemple 1.4 (Problème Diffie-Hellman computationnel — CDH)**.** Le problème de Diffie-Hellman illustre comment deux entités établissent un secret commun sur un canal non sécurisé. Reprenons $\mathbb{F}_{11}$ avec $g = 2$.

❖ **Échange de clés :**
Alice choisit un secret $a = 3$ et envoie $A = g^a = 2^3 = 8 \pmod{11}$.
Bob choisit un secret $b = 4$ et envoie $B = g^b = 2^4 = 5 \pmod{11}$.

❖ **Calcul du secret partagé (S) :**
Alice calcule $S = B^a = 5^3 = 125 \equiv 4 \pmod{11}$.
Bob calcule $S = A^b = 8^4 = 4096 \equiv 4 \pmod{11}$.

Difficulté (CDH) : Un adversaire intercepte les valeurs publiques $(g, g^a, g^b) = (2, 8, 5)$. Le *Computational Diffie-Hellman Problem* consiste à retrouver $g^{ab} = 4$. Si la taille du groupe q est de 256 bits, l'adversaire fait face à un espace de recherche de 2^{256} combinaisons : le secret partagé reste hors de portée sans résoudre au préalable le logarithme discret.

Définition 1.9 (Problème Diffie-Hellman décisionnel — DDH)**.** Étant donné (g, g^a, g^b, g^c), décider si $c = ab \bmod q$ ou si c est aléatoire.

1.2.3 La factorisation

Historiquement, la factorisation a ouvert la voie aux premières preuves ZK. Elle repose sur la structure de l'anneau $\mathbb{Z}/n\mathbb{Z}$ où n est un produit de grands nombres premiers.

Définition 1.10 (Problème de factorisation)**.** Étant donné un entier composé $n = p \cdot q$, trouver les facteurs premiers p et q.

Ce problème est directement lié à la difficulté de décider si un nombre est un carré modulo n : c'est la base du protocole de Feige-Fiat-Shamir.

1.3 Théorie de la complexité calculatoire

La cryptographie moderne repose sur l'existence de problèmes *faciles à poser mais difficiles à résoudre*. La théorie de la complexité fournit les outils formels pour définir cette asymétrie avec précision [36].

1.3.1 Modèles de calcul

La sécurité des systèmes cryptographiques contemporains repose sur le concept de *sécurité computationnelle*. Contrairement à la sécurité informationnelle, elle suppose que l'adversaire dispose de ressources limitées. Goldwasser et al. [36] formalisent ce cadre autour de deux notions clés.

Définition 1.11 (Machine de Turing probabiliste polynomiale (PPT))**.** Une *machine de Turing probabiliste polynomiale* (PPT) est une machine de Turing disposant d'un ruban d'aléas, dont le temps d'exécution est borné par un polynôme en la taille de l'entrée, pour toute valeur du ruban d'aléas [34].

Définition 1.12 (Machine de Turing déterministe (MTD))**.** Une *machine de Turing déterministe* est un modèle de calcul composé d'un ruban infini de cellules, d'une tête de lecture/écriture et d'un registre d'état. Elle est définie par un 7-uplet $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ où :

- Q est un ensemble fini d'états ;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ est la fonction de transition.

Le caractère *déterministe* implique que pour chaque configuration (état actuel et symbole lu), la fonction δ définit **une unique action possible**. Aucun hasard, aucun choix alternatif lors de l'exécution.

Définition 1.13 (Fonction négligeable)**.** Une fonction $f : \mathbb{N} \to \mathbb{R}$ est dite *négligeable* si pour tout $c \in \mathbb{R}^+$, il existe un entier $n_0 \in \mathbb{N}$ tel que pour tout $n > n_0$:

$$|f(n)| < \frac{1}{n^c}$$

On note généralement cette propriété par $f(n) = \mathrm{negl}(n)$ [18].

Exemple 1.5 (Fonctions négligeables)**.** Les fonctions 2^{-n}, $n^{-\log n}$ et $2^{-\sqrt{n}}$ sont négligeables. La fonction n^{-5} ne l'est pas (elle est l'inverse d'un polynôme).

Le modèle PPT est la définition formelle d'un *adversaire efficace* en cryptographie. Toutes les propriétés de sécurité des ZKP s'y réfèrent : un protocole est sûr si aucun adversaire PPT ne peut le casser, sauf avec probabilité négligeable [36]. Une probabilité de $\mathsf{negl}(n)$ est en pratique nulle pour des paramètres de sécurité n suffisamment grands (typiquement $n = 128$ ou 256 bits).

Définition 1.14 (Transcription d'un protocole interactif)**.** Soit $(\mathcal{P}, \mathcal{V})$ un protocole interactif en trois tours. Une *transcription* (ou *vue*) est le triplet (t, c, s) de tous les messages échangés lors d'une exécution : l'engagement t envoyé par $\mathcal{P}$, le défi c choisi par $\mathcal{V}$, et la réponse s de $\mathcal{P}$. Deux transcriptions (t, c_1, s_1) et (t, c_2, s_2) partagent le même engagement si elles diffèrent uniquement sur leur défi et leur

réponse [9].

Définition 1.15 (Rembobinage (*Rewinding*))**.** Le *rembobinage* d'un algorithme probabiliste $\mathcal{A}$ est la technique qui consiste à exécuter $\mathcal{A}$ jusqu'à un certain point, à enregistrer son état interne (ruban d'aléas, état de calcul), puis à le relancer depuis cet état en lui fournissant une entrée différente. Le rembobinage permet à un extracteur de provoquer deux exécutions de $\mathcal{A}$ partageant le même préfixe aléatoire mais recevant des défis distincts [36, 9].

Définition 1.16 (Extracteur de témoin (*Knowledge Extractor*))**.** Soit $(\mathcal{P}, \mathcal{V})$ un protocole ZKP pour une relation R. Un *extracteur de témoin* $\mathcal{E}$ est un algorithme PPT qui, en ayant accès en boîte noire à un prouveur $\mathcal{P}^*$ (éventuellement malhonnête) capable de convaincre $\mathcal{V}$ avec probabilité non-négligeable, produit un témoin w tel que $(x, w) \in R$ avec probabilité non-négligeable. L'extracteur peut rembobiner $\mathcal{P}^*$ autant de fois que nécessaire [26, 18].

1.3.2 Classes de complexité

La sécurité des protocoles ZKP repose sur une distinction fondamentale : il est difficile de *résoudre* un problème, mais facile d'en *vérifier* la solution. Les classes $\mathcal{P}$ et $\mathcal{NP}$ formalisent cette hiérarchie.

Définition 1.17 (Problème de décision)**.** Un **Problème de décision** est un problème dont la réponse est $\mathcal{OUI}$ ou $\mathcal{NON}$.

Définition 1.18 (Réduction polynomiale de Karp)**.** Un problème A se *réduit polynomialement* à un problème B (noté $A \leq_p B$) s'il existe une fonction f calculable en temps polynomial telle que :

$$x \in A \iff f(x) \in B$$

Intuitivement, toute instance de A peut être transformée efficacement en une instance de B. Si B est facile à résoudre, alors A l'est aussi [34, 35].

Définition 1.19 (Classe $\mathcal{P}$)**.** La classe $\mathcal{P}$ est l'ensemble des problèmes de décision résolubles en temps polynomial.

Exemple 1.6 (Calcul du PGCD par l'algorithme d'Euclide)**.** Le problème consiste à trouver le plus grand entier qui divise deux nombres a et b. Pour $a = 252$ et $b = 105$:

- **Étape 1 :** $252 = 2 \times 105 + 42$
- **Étape 2 :** $105 = 2 \times 42 + 21$
- **Étape 3 :** $42 = 2 \times 21 + 0$

Le dernier reste non nul est 21, donc $PGCD(252, 105) = 21$.

Pourquoi ce problème appartient-il à $\mathcal{P}$?

- **Rapidité d'exécution :** Même avec des nombres comportant des centaines de chiffres, la solution est trouvée quasi instantanément.
- **Absence de tâtonnement :** L'ordinateur suit un chemin direct et unique. Pas de combinaisons à tester au hasard, contrairement aux problèmes de la cryptographie.

Définition 1.20 (Classe $\mathcal{NP}$)**.** La classe $\mathcal{NP}$ est l'ensemble des problèmes de décision pour lesquels la réponse $\mathcal{OUI}$ peut être vérifiée en temps polynomial moyennant une information supplémentaire appelée "certificat" [35].

Exemple 1.7 (Le problème du Sudoku)**.** Considérons une grille de Sudoku de grande taille ($n \times n$). Ce problème appartient à la classe $\mathcal{NP}$ pour les raisons suivantes :

- **Difficulté de résolution :** Pour une grille très complexe, aucun algorithme déterministe simple ne permet de trouver la solution directement. On explore de nombreuses combinaisons et donc le temps peut devenir exponentiel.
- **Facilité de vérification :** Si quelqu'un fournit une grille déjà remplie (le *témoin*), il est facile et rapide (en temps polynomial) de vérifier qu'aucune erreur n'a été commise sur chaque ligne, colonne et carré.

Définition 1.21 ($\mathcal{NP}$-complétude)**.** La classe $\mathcal{NP}$-*Complet* est l'ensemble des problèmes de décision pour lesquels la réponse $\mathcal{OUI}$ peut être vérifiée en temps polynomial, et pour lesquels tout autre problème de $\mathcal{NP}$ peut être *réduit polynomialement* (au sens de la Définition 1.18) à ce dernier [34].

Définition 1.22 ($\mathcal{NP}$-dur)**.** Un problème Q est dit *$\mathcal{NP}$-dur* si tout problème de $\mathcal{NP}$ se réduit polynomialement à Q. Un problème $\mathcal{NP}$-dur n'appartient pas nécessairement lui-même à $\mathcal{NP}$: il peut ne pas admettre de certificat vérifiable efficacement [34, 35].

Exemple 1.8 (NP-complétude : le Circuit Hamiltonien)**.** Imaginons un réseau de villes reliées par des routes. Le problème : existe-t-il un itinéraire qui passe par chaque ville exactement une fois et revient à la ville de départ ?

- **Vérification (Classe $\mathcal{NP}$)** : Si un itinéraire (le témoin) est fourni, sa validité se vérifie en temps polynomial.
- **Difficulté ($\mathcal{NP}$-dur)** : Trouver cet itinéraire est complexe car le temps de calcul explose avec le nombre de villes (croissance exponentielle).
- **Universalité** : Une solution efficace à ce problème permettrait, par réduction polynomiale, de résoudre tout autre problème de $\mathcal{NP}$.

Remarque 1.2. La définition de $\mathcal{NP}$ ne stipule pas que le certificat peut être *trouvé* en temps polynomial. Elle énonce simplement que, pour toute instance dont la réponse est **OUI**, il existe un témoin qui permet à un vérificateur de *confirmer* cette réponse en temps polynomial.

1.4 Primitives cryptographiques

Un protocole ZKP n'est sûr que s'il s'appuie sur des fonctions mathématiques dont l'inversion est jugée impossible dans un modèle de calcul réaliste. Ces primitives constituent le langage de base : elles traduisent des énoncés logiques en preuves mathématiques vérifiables. Trois primitives structurent cette section : les fonctions à sens unique, les fonctions de hachage et les schémas d'engagement.

1.4.1 Fonctions à sens unique

Définition 1.23 (Fonction à sens unique — OWF)**.** On dit qu'une fonction $f : \{0,1\}^* \to \{0,1\}^*$ est une fonction *à sens unique* si [26] :

i. **Calculabilité :** f est calculable en temps polynomial.

ii. **Irréversibilité :** pour tout adversaire PPT $\mathcal{A}$, pour tout $x \xleftarrow{\$} \{0,1\}^n$:

$$\Pr\Big[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))\Big] \leq \mathsf{negl}(n)$$

Exemple 1.9 (Fonction à sens unique : Multiplication et Factorisation)**.** Un exemple classique repose sur la multiplication de deux grands nombres premiers p et q :

- **Sens direct (Facile) :** Le calcul de $n = p \times q$ est réalisable en temps polynomial par n'importe quelle machine.
- **Sens inverse (Difficile) :** Retrouver p et q à partir de n (problème de la factorisation) est computationnellement impossible pour de grandes valeurs de n.

La fonction $f(p, q) = p \times q$ satisfait les critères d'une OWF : facile à calculer, hors de portée à inverser pour un adversaire PPT.

1.4.2 Modèle de l'oracle aléatoire (ROM)

Définition 1.24 (Oracle aléatoire et modèle ROM)**.** Un *oracle aléatoire* $H : \{0,1\}^* \to \{0,1\}^n$ est une fonction choisie uniformément au hasard parmi toutes les fonctions de même signature. Ses sorties sont uniformes et imprévisibles : pour toute entrée x jamais interrogée, $H(x)$ est indépendant de toutes les sorties précédentes.

Dans le *modèle de l'oracle aléatoire* (ROM) [23], toute fonction de hachage cryptographique H (par exemple SHA-256) est modélisée comme un oracle aléatoire : toutes les parties (prouveur, vérificateur, adversaire) y accèdent uniquement par requêtes, et ne peuvent pas calculer H autrement.

Portée et limites du ROM. Le ROM est un modèle *idéalisé* : les fonctions de hachage réelles (SHA-256, BLAKE2s) ne sont pas de vrais oracles aléatoires. Des protocoles prouvés sûrs dans le ROM peuvent, en théorie, être rendus non sûrs par des fonctions de hachage concrètes [23]. Cependant, en pratique, le ROM est le standard pour les preuves de sécurité impliquant la transformation de Fiat-Shamir, et aucun contre-exemple pratique sur SHA-256 n'est connu à ce jour. La sécurité dans le *modèle standard* (sans idéalisation de H) reste une question ouverte pour les protocoles de ce mémoire.

1.4.3 Fonctions de hachage cryptographiques

Définition 1.25 (Fonction de hachage cryptographique)**.** Une *fonction de hachage cryptographique* est une fonction $H : \{0,1\}^* \to \{0,1\}^n$ satisfaisant [18] :

i. **Résistance à la préimage :** étant donné h, il est difficile de trouver x tel que $H(x) = h$.

ii. **Résistance à la seconde préimage :** étant donné x, il est difficile de

trouver $x' \neq x$ tel que $H(x') = H(x)$.

iii. **Résistance aux collisions :** il est difficile de trouver (x, x') avec $x \neq x'$ et $H(x) = H(x')$.

Exemple 1.10 (Fonctions de hachage cryptographiques)**.** Considérons une fonction de hachage H comme le SHA-256 :

- **Préimage :** Empêche un attaquant de retrouver un mot de passe à partir de son empreinte stockée en base de données.
- **Seconde préimage :** Garantit qu'un utilisateur ne peut pas modifier le contenu d'un document signé sans que l'empreinte ne change.
- **Collisions :** Assure l'unicité des empreintes ; deux documents distincts ne peuvent produire le même condensé.

Dans les ZKP, les fonctions de hachage jouent deux rôles distincts :

i. **Oracle aléatoire.** Dans la transformation de Fiat-Shamir [1], la fonction de hachage *simule* un vérificateur externe : elle génère le défi de manière déterministe mais imprévisible :

$$c = H(g \parallel y \parallel t) \bmod q$$

où g est le générateur du groupe, $y = g^x \bmod p$ la clé publique du prouveur, $t = g^r \bmod p$ l'engagement, et q l'ordre du sous-groupe. Cette notation est maintenue uniformément dans l'ensemble du mémoire.

ii. **Engagement.** Dans les zk-STARKs [12], les arbres de Merkle construits à partir de fonctions de hachage servent d'engagements polynomiaux (ils remplacent les couplages bilinéaires des SNARKs).

TABLE 1.2 – Fonctions de hachage utilisées dans les protocoles ZKP

Fonction	Sortie	Sécurité	Post-quantique	Utilisations
SHA-256	256 bits	128 bits	64 bits	Fiat-Shamir, STARKs
SHA-3-256	256 bits	128 bits	64 bits	STARKs (alternative)
BLAKE2s	256 bits	128 bits	64 bits	Protocoles légers
Poseidon	variable	variable	variable	ZKP natifs en corps fini

Remarque 1.3 (Optimisation de Poseidon). À la différence des fonctions classiques (SHA-256), Poseidon est conçue pour minimiser le nombre de contraintes arithmétiques (R1CS). La génération de preuves ZK devient nettement plus efficace. C'est

la fonction de hachage de référence pour les protocoles ZKP natifs en corps fini — systèmes basés sur PLONK ou Groth16 [16] en particulier.

1.4.4 Schémas d'engagement cryptographique

Un *schéma d'engagement* permet au prouveur de se *lier* à une valeur secrète sans la révéler, pour la révéler plus tard de façon vérifiable. Il est au cœur du mécanisme d'engagement dans tous les Σ-protocoles [14].

Définition 1.26 (Schéma d'engagement)**.** Un *schéma d'engagement* est un couple d'algorithmes PPT $(\mathsf{Com}, \mathsf{Open})$ définis sur un espace de messages $\mathcal{M}$ tel que [26] :

i. **Engagement :** $c \leftarrow \mathsf{Com}(v;\, r)$ produit un engagement c à partir d'une valeur $v \in \mathcal{M}$ et d'un aléa $r \xleftarrow{\$} \{0,1\}^\lambda$.

ii. **Ouverture :** $\mathsf{Open}(c, v, r) \in \{1, 0\}$ vérifie l'intégrité de l'engagement : il retourne 1 si $c = \mathsf{Com}(v; r)$ et 0 sinon.

Le schéma doit satisfaire les deux propriétés suivantes :

- **Dissimulation (*Hiding*) :** Pour tout adversaire, les distributions des engagements de deux messages distincts sont indiscernables (le contenu du "coffre" est invisible).
- **Liaison (*Binding*) :** Un adversaire ne peut pas trouver un triplet (c, v, r) et (v', r') tels que $v \neq v'$ et $\mathsf{Open}(c, v, r) = \mathsf{Open}(c, v', r') = 1$ (le "coffre" ne peut pas être modifié après coup).

L'engagement de Pedersen

L'engagement de Pedersen [37] est la référence dans les ZKP. Sa propriété d'homomorphisme additif est ce qui rend les Bulletproofs possibles [5].

Définition 1.27 (Engagement de Pedersen [37])**.** Soient $\mathbb{G}$ un groupe cyclique d'ordre premier q et $g, h \in \mathbb{G}$ deux générateurs indépendants (c'est-à-dire $\log_g h$ est inconnu). L'engagement de Pedersen est défini par :

$$\mathsf{Com}(v;\, r) = g^v \cdot h^r \bmod p, \quad v \in \mathbb{Z}_q,\ r \xleftarrow{\$} \mathbb{Z}_q$$

Il satisfait :

- **Dissimulation parfaite :** la distribution de c est uniforme dans $\mathbb{G}$, indépendamment de v.

- ❖ **Liaison computationnelle :** ouvrir c sur deux valeurs différentes implique de calculer $\log_g h$.
- ❖ **Homomorphisme additif :** $\mathsf{Com}(v_1; r_1) \cdot \mathsf{Com}(v_2; r_2) = \mathsf{Com}(v_1 + v_2;\ r_1 + r_2)$

Vérification de la propriété d'homomorphisme.

$$\mathsf{Com}(v_1; r_1) \cdot \mathsf{Com}(v_2; r_2) = g^{v_1} h^{r_1} \cdot g^{v_2} h^{r_2} = g^{v_1+v_2} \cdot h^{r_1+r_2} = \mathsf{Com}(v_1 + v_2;\ r_1 + r_2).$$

La propriété d'homomorphisme est ce qui rend Pedersen remarquable : elle permet de *vérifier des relations arithmétiques sur des valeurs engagées* sans jamais les révéler. Pour prouver que $v \in [0, 2^n - 1]$, on décompose v en bits $b_0, \ldots, b_{n-1}$, on s'engage sur chaque bit séparément, puis on prouve que chaque engagement est bien 0 ou 1 grâce à l'homomorphisme [5]. C'est le mécanisme fondamental des range proofs des Bulletproofs.

Engagement par hachage

Une alternative plus simple, utilisée dans les zk-STARKs [12], est l'engagement par hachage :

$$\mathsf{Com}(v;\ r) = H(v \,\|\, r)$$

Il satisfait la dissimulation computationnelle (sous résistance à la préimage) et la liaison computationnelle (sous résistance aux collisions), mais *n'est pas homomorphe*. Son avantage principal est la résistance post-quantique. Il ne repose sur aucune hypothèse de difficulté algébrique (logarithme discret, factorisation).

1.4.5 Σ-protocoles

Les Σ-protocoles structurent une grande famille de protocoles ZKP interactifs.

Définition 1.28 (Σ-protocole)**.** Soit R une relation binaire. Un Σ-*protocole* pour R est un protocole interactif $(\mathcal{P}, \mathcal{V})$ en trois tours (engagement t, défi $c \in \mathcal{C}$, réponse s) satisfaisant [14] :

i. **Complétude :** pour tout $(x, w) \in R$, le vérificateur honnête accepte avec probabilité 1.

ii. **Robustesse spéciale (*special soundness*) :** il existe un extracteur PPT $\mathcal{E}$ tel que, données deux transcriptions acceptantes (t, c, s) et (t, c', s') pour la même instance x avec $c \neq c'$, $\mathcal{E}$ calcule un témoin w tel que $(x, w) \in R$ en temps polynomial.

iii. **Divulgation nulle parfaite pour le vérificateur honnête :** il existe un simulateur PPT $\mathcal{S}$ qui, donné x et un défi c arbitraire, produit une transcription (t, c, s) statistiquement indiscernable d'une vraie interaction.

Remarque 1.4 (Place des Σ-protocoles dans ce mémoire). Le protocole de Schnorr (§ 3.1.1) est le Σ-protocole de référence pour la preuve de connaissance d'un logarithme discret. La OR-proof CDS (§ 5.3.4) est une composition de Σ-protocoles permettant de prouver la vérité d'une disjonction sans révéler laquelle des branches est vraie. Cette composition est possible grâce à la robustesse spéciale et à la simulabilité indépendante des branches [30, 14].

Conclusion

Ce chapitre a posé les jalons mathématiques indispensables à la sécurité des ZKP. La sécurité de ces protocoles ne repose pas sur une confiance aveugle. Elle repose sur des piliers précis : structure des groupes cycliques, asymétrie calculatoire du logarithme discret, formalisation de la limite d'un adversaire par la théorie de la complexité. Articulées avec les primitives que sont les fonctions de hachage et les schémas d'engagement, ces bases constituent la boîte à outils nécessaire pour aborder l'étude détaillée des protocoles ZKP. Ce chapitre a également introduit deux outils transversaux indispensables : le modèle de l'oracle aléatoire (Définition 1.24) et les Σ-protocoles (Définition 1.28), dont les chapitres suivants feront un usage constant.

CHAPITRE 2

FONDEMENTS THÉORIQUES ET MÉCANISMES DES ZKPS

Introduction

Comment prouver qu'on connaît un secret sans jamais le révéler ? C'est la question fondatrice des preuves à divulgation nulle de connaissance (*Zero-Knowledge Proofs*). Formulée par Goldwasser, Micali et Rackoff en 1985 [36], elle semble paradoxale. Ce chapitre montre qu'elle ne l'est pas.

Nous partons d'une analogie pédagogique (la grotte d'Ali Baba) pour rendre la mécanique intuitive, puis nous formalisons les trois propriétés qui définissent tout protocole ZKP : complétude, robustesse, divulgation nulle. La définition formelle et le processus de construction terminent ce chapitre.

2.1 Analogie pédagogique : La grotte d'Ali Baba

L'analogie de la grotte d'Ali Baba [17] est la plus pédagogique qui existe pour introduire les ZKP. Le scénario met en scène une cavité circulaire disposant d'une entrée unique se scindant en deux couloirs, désignés par A et B. Ces deux couloirs sont séparés en leur point de jonction par une porte verrouillée. Alice prétend connaître le mot secret qui ouvre cette porte. Elle veut en convaincre Bob sans lui révéler le mot.

2.1.1 Le protocole en trois phases

Phase 1 — Engagement. Bob reste à l'extérieur. Alice entre et choisit secrètement l'un des deux couloirs (A ou B) Bob ignore où elle est.

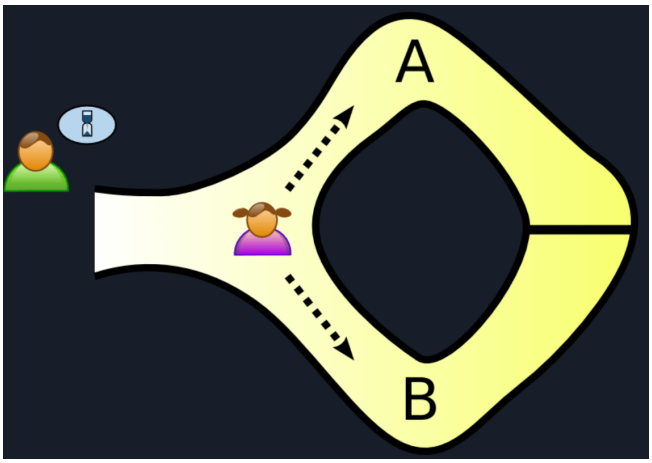


FIGURE 2.1 – Phase d'engagement dans le protocole interactif [15].

Phase 2 — Défi. Bob entre jusqu'au point de bifurcation et ordonne : « Ressors par le côté A ! » Il ne sait pas où Alice se trouve.

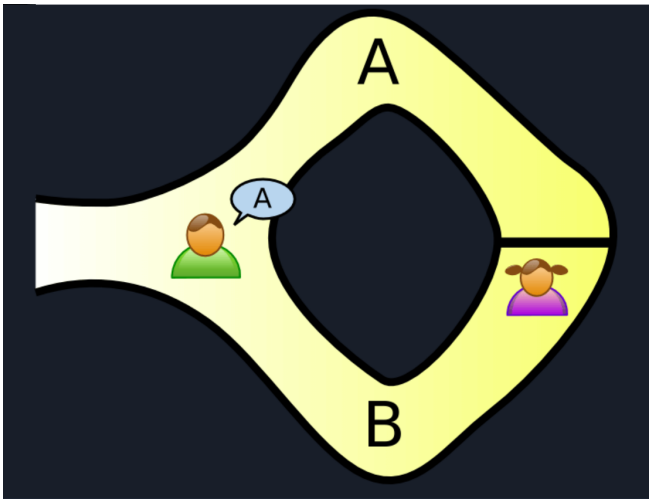


FIGURE 2.2 – Émission du défi par le vérificateur [15].

Phase 3 — Réponse. Deux cas sont possibles :

- **Alice connaît le secret :** elle peut traverser la porte depuis n'importe quel couloir et ressortir par celui demandé par Bob.
- **Alice ment :** elle ne réussit que si le hasard l'a placée d'emblée dans le bon couloir (probabilité $\frac{1}{2}$).

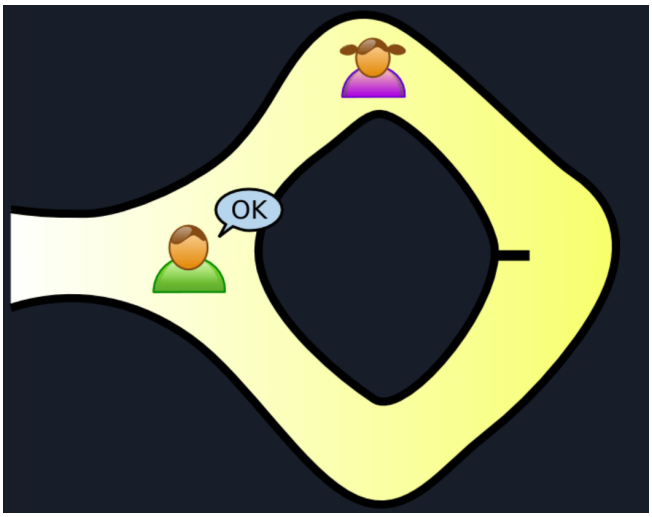


FIGURE 2.3 – Vérification de la réponse [15].

2.1.2 Analyse de la fiabilité

Si Alice ment, sa probabilité de tromper Bob à chaque tour est $\frac{1}{2}$. Sur n tours :

$$P(\text{succès imposteur}) = \frac{1}{2^n} \tag{2.1.1}$$

Dix tours suffisent à ramener cette probabilité sous 0,1%. Cent tours la rendent négligeable. Bob acquiert une certitude statistique sans jamais entendre le mot secret.C'est l'essence des ZKP : convaincre par itération, sans transmission d'information.

L'analogie de la grotte permet de saisir intuitivement le concept. La mise en œuvre de cette confiance sans révélation nécessite un cadre formel rigoureux, reposant sur des propriétés mathématiques vérifiables que nous détaillons ci-après.

2.2 Propriétés fondamentales des ZKPs

Un protocole ZKP est valide si et seulement s'il satisfait les trois propriétés définies par Goldwasser, Micali et Rackoff [36].

2.2.1 Complétude (*Completeness*)

Si l'assertion est vraie et que le prouveur est honnête, le vérificateur honnête accepte toujours [36].

Exemple : Dans l'analogie, si Alice connaît le mot, elle réussit chaque défi de Bob sans exception.

2.2.2 Robustesse (*Soundness*)

Si l'assertion est fausse, aucun prouveur malhonnête ne peut convaincre le vérificateur, sauf avec une probabilité négligeable [36]. La probabilité de succès de l'imposteur décroît exponentiellement avec le nombre de tours (voir l'équation (2.1.1)).
Exemple : Si Alice ment, ses chances de succès diminuent de façon exponentielle à chaque itération.

2.2.3 Divulgation nulle (*Zero-Knowledge*)

Le vérificateur n'apprend rien d'autre que la véracité de l'assertion [36]. Pas le secret, pas d'information partielle, rien. Bob sait juste qu'Alice connaît le mot, il ne peut pas déduire ce mot, même après cent tours.
Exemple : Si Bob observe plusieurs fois Alice réussir à ressortir par le bon chemin, il ne parvient jamais à déduire le mot magique.

Formellement, on distingue trois variantes de cette propriété selon la force de l'indistinguabilité requise [36] :

- **Divulgation nulle parfaite** (*perfect ZK*) : les distributions sont identiques.
- **Divulgation nulle statistique** (*statistical ZK*) : les distributions sont statistiquement indiscernables (distance statistique négligeable).
- **Divulgation nulle computationnelle** (*computational ZK*) : les distributions sont computationnellement indiscernables pour tout adversaire PPT.

2.3 Définition formelle des ZKPs

Définition 2.1 (Preuve à Divulgation Nulle de Connaissance)**.** Soit $R \subseteq \{0,1\}^* \times \{0,1\}^*$ une relation de preuve et $L = \{x \mid \exists\, w,\ (x,w) \in R\}$ le langage associé. Un protocole interactif (P, V) est une *preuve à divulgation nulle de connaissance (ZKP) pour* R s'il satisfait [36] :

i. **Complétude :** pour tout $(x,w) \in R$, V accepte avec probabilité 1.

ii. **Robustesse :** pour tout prouveur malveillant P^* et tout $x \notin L$, la probabilité que V accepte est $\mathsf{negl}(\lambda)$.

iii. **Divulgation nulle :** il existe un simulateur PPT $\mathcal{S}$ tel que, pour tout $x \in L$: $\left\{\mathcal{S}(x)\right\} \approx_c \left\{\langle P(x,w),\, V(x)\rangle\right\}$ où $\approx_c$ désigne l'indiscernabilité computationnelle et $\left\{\mathcal{X}\right\}$ la distribution de la trace produite par $\mathcal{X}$.

La propriété de divulgation nulle via simulation est la clé. Elle dit : tout ce que V peut calculer après l'interaction, il pourrait le calculer sans elle. La trace ne contient aucune information sur w.

2.4 Processus de construction des ZKPs

La construction d'un protocole ZKP suit une démarche formelle qui articule modélisation, choix de paradigmes et analyses de sécurité [18].

2.4.1 Entités et rôles

Quatre acteurs interviennent :

- **Le prouveur $\mathcal{P}$:** possède le témoin w tel que $(x, w) \in R$. Veut convaincre $\mathcal{V}$ que $x \in L$ sans révéler w.
- **Le vérificateur $\mathcal{V}$:** reçoit la preuve et décide d'accepter ou de rejeter selon une procédure formelle.
- **Le simulateur $\mathcal{S}$:** machine PPT qui génère une trace indiscernable de l'interaction réelle sans connaître w [18]. Son existence est la définition opérationnelle de la divulgation nulle.
- **L'adversaire PPT :** modèle un prouveur ou vérificateur malveillant cherchant à violer la robustesse ou la divulgation nulle [18].

2.4.2 Formalisation du problème

La première étape consiste à décrire l'assertion à prouver sous forme d'une relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$. L'instance publique x et le témoin secret w vérifient $(x, w) \in R$. Le prouveur doit montrer que $x \in L$ en gardant w opaque.

Ce cadre place naturellement le problème dans $\mathcal{NP}$. Goldreich, Micali et Wigderson ont montré que tout langage $\mathcal{NP}$ admet un protocole ZKP [27], résultat fondateur qui légitime l'universalité de l'approche.

2.4.3 Choix du paradigme

On distingue principalement les deux paradigmes suivants [18] :

- **ZKP interactif :** plusieurs échanges entre $\mathcal{P}$ et $\mathcal{V}$, typiquement en trois tours. Ce modèle est le cadre initial introduit par Goldwasser, Micali et Rackoff [36].
- **ZKP non interactif :** une seule preuve autonome, construite via l'heuristique de Fiat-Shamir ou dans le modèle de l'oracle aléatoire [14].

Le choix détermine la structure du jeu de sécurité et la définition opérationnelle de la divulgation nulle.

2.4.4 Schéma en trois tours

La conception d'un schéma de preuve se fait généralement sous la forme d'un schéma à trois tours de communication [14] :

- **Engagement (*commitment*) :** $\mathcal{P}$ envoie une valeur masquée t, dérivée de w via une fonction à sens unique.
- **Défi (*challenge*) :** $\mathcal{V}$ (ou un oracle aléatoire) génère c dont la distribution contrôle la probabilité de succès d'un adversaire.
- **Réponse (*response*) :** $\mathcal{P}$ calcule s depuis w et c, tel que la vérification relie s à la validité de $(x, w) \in R$.

2.4.5 Preuve des trois propriétés

Une fois le schéma défini, il faut démontrer formellement :

- **Complétude :** si $(x, w) \in R$, les équations de vérification sont satisfaites par construction [36].
- **Robustesse :** si $x \notin L$, tout adversaire PPT ne peut convaincre $\mathcal{V}$ qu'avec probabilité $\mathsf{negl}(\lambda)$ [36]. Pour Schnorr, cela revient à résoudre le DLP depuis deux transcriptions acceptantes — le *forking lemma* [9].
- **Divulgation nulle :** construire explicitement $\mathcal{S}(x)$ qui produit une trace indiscernable sans accès à w [36].

Conclusion

Ce chapitre a posé les bases conceptuelles et formelles des ZKP. La grotte d'Ali Baba traduit l'intuition ; la définition de Goldwasser, Micali et Rackoff [36] la formalise. Trois propriétés (complétude, robustesse, divulgation nulle) et un simulateur : c'est tout ce dont un protocole a besoin pour être qualifié de ZKP. La notation (t, c, s) est établie ; elle revient dans chaque protocole du document. Le chapitre suivant analyse les différents modèles qui instancient ce cadre : de Schnorr aux STARKs.

CHAPITRE 3

MODÈLES, HYPOTHÈSES DE SÉCURITÉ ET ANALYSE COMPARATIVE DES ZKPS

Introduction

Un protocole ZKP n'existe pas en un seul modèle. Selon la contrainte à traiter (latence réseau, architecture décentralisée, ressources matérielles limitées) la construction change radicalement. Deux grandes familles structurent le paysage : les **modèles interactifs**, qui reposent sur un dialogue en temps réel entre prouveur et vérificateur, et les **modèles non-interactifs**, où le prouveur produit une preuve autonome que n'importe qui peut vérifier, à tout moment, sans échange supplémentaire [36].

Ce chapitre présente les principaux représentants de chacune de ces familles, analyse leurs hypothèses de sécurité, puis les confronte sur quatre axes : mécanisme d'échange, transférabilité, efficacité et hypothèses de sécurité.

3.1 Le modèle interactif (IZKP)

Le modèle interactif s'articule autour d'un *Sigma-protocole* (Σ-protocol) structuré en trois passes : l'engagement (*commitment*), le défi aléatoire (*challenge*) et la réponse (*response*). La Figure 3.1 illustre cette dynamique.

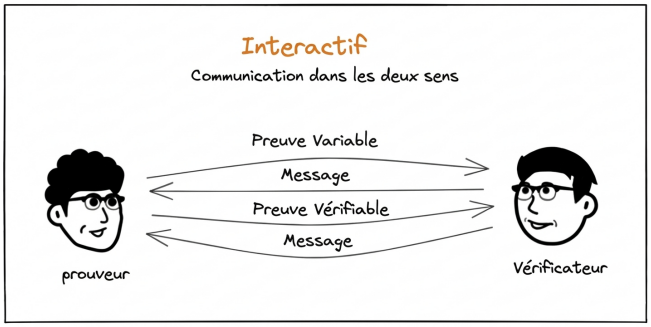


FIGURE 3.1 – Architecture d'un protocole ZKP interactif [20].

La force du modèle tient à une asymétrie simple : le prouveur ne connaît pas le défi à l'avance, donc il ne peut pas préparer une réponse frauduleuse. Sa limite est symétrique. Il exige une connexion synchrone entre les deux parties, ce qui le rend difficilement déployable dans des systèmes asynchrones ou décentralisés. Parmi les protocoles qui incarnent ce modèle, le protocole de Schnorr en reste l'illustration la plus nette.

3.1.1 Le protocole d'identification de Schnorr

Le protocole de Schnorr [6] constitue la preuve de référence pour la connaissance d'un logarithme discret. Soit un groupe cyclique $\mathbb{G}$ d'ordre premier q, engendré par g sur un corps fini $\mathbb{F}_p$. L'objectif est de prouver la connaissance du secret $x \in \mathbb{Z}_q$ tel que $y = g^x \pmod{p}$, sans jamais divulguer x.

- **Engagement.** $r \xleftarrow{\$} \mathbb{Z}_q, \quad t = g^r \pmod{p}$ et t transmis au vérificateur.
- **Défi.** $c \xleftarrow{\$} \mathbb{Z}_q$ et envoyé par le vérificateur au prouveur.
- **Réponse.** $s = (r + c \cdot x) \bmod q$, envoyé par le prouveur.
- **Vérification.** $g^s \equiv t \cdot y^c \pmod{p}$.

La correction est immédiate : $g^s = g^{r+cx} = t \cdot y^c$. Extraire x depuis s sans connaître r revient à résoudre le DLP. L'aléa r joue le rôle de masque cryptographique. Sans lui, $s = cx \bmod q$ permettrait de retrouver $x = s \cdot c^{-1} \pmod{q}$ [1].

3.1.2 Le protocole de Feige–Fiat–Shamir (FFS)

Le schéma FFS [38] répond à des contraintes que Schnorr n'adresse pas. Cartes à puce, dispositifs IoT, architectures sans coprocesseur cryptographique : partout où l'exponentiation devient un luxe, FFS remplace les calculs lourds par des multiplications modulaires. Sa sécurité repose sur la difficulté d'extraire des racines carrées modulo un entier composé $n = p \cdot q$, problème équivalent à factoriser n [10].

- **Génération des clés.** Le prouveur choisit k secrets $a_1, \ldots, a_k$ et publie la valeur $y_i = a_i^{-2} \bmod n$.

- **Engagement.** $r \xleftarrow{\$} \mathbb{Z}_n^*, \quad t = r^2 \pmod{n}$ est transmis au vérificateur.
- **Défi.** $(c_1, \ldots, c_k) \xleftarrow{\$} \{0,1\}^k$ envoyé par le vérificateur.
- **Réponse.** $s = r \cdot \prod_{i=1}^{k} a_i^{c_i} \pmod{n}$.
- **Vérification.** $s^2 \cdot \prod_{i=1}^{k} y_i^{c_i} \equiv t \pmod{n}$. La probabilité d'erreur est : 2^{-kt} [38].

Ces deux protocoles partagent la structure engagement-défi-réponse. Mais ils restent prisonniers du canal synchrone. C'est précisément pour briser cette contrainte que Fiat et Shamir ont proposé dès 1986 de remplacer le vérificateur par une fonction de hachage [1] d'où la naissance du modèle non-interactif.

3.2 Le modèle non-interactif (NIZKP)

À la différence du modèle interactif qui exige une synchronisation du canal, le modèle non-interactif (Figure 3.2) permet au prouveur de générer une preuve π de manière unilatérale.

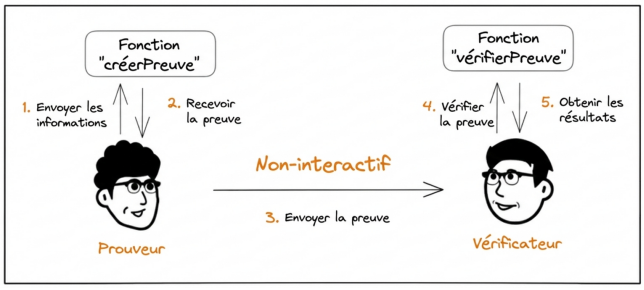


FIGURE 3.2 – Architecture d'un protocole ZKP non interactif [20].

Le gain est immédiat. Plus de latence liée aux allers-retours, plus de synchronisation entre les parties. La preuve devient un objet autonome qui est publiable, archivable, vérifiable par n'importe quel observateur. Ce basculement repose entièrement sur la transformation de Fiat-Shamir.

L'idée est d'une simplicité redoutable. Dans un protocole interactif, le vérificateur envoie un défi aléatoire c que le prouveur ne peut pas anticiper. La transformation de Fiat-Shamir remplace ce défi par [1] :

$$c = H(g \parallel y \parallel t) \bmod q,$$

où H est une fonction de hachage cryptographique. Le prouveur calcule lui-même son défi. Comme H se comporte comme une boîte noire déterministe (un **oracle aléatoire** dans le modèle formel) il ne peut pas choisir t pour orienter c à sa guise. Le défi reste imprévisible et non manipulable.
La transformation de Fiat-Shamir a ouvert la voie aux protocoles modernes :

3.2.1 Les protocoles zk-SNARKs

Un zk-SNARK (*Zero-Knowledge Succinct Non-Interactive Argument of Knowledge*) produit une preuve de quelques centaines d'octets, vérifiable en temps quasi-constant $\mathcal{O}(1)$, quelle que soit la complexité du calcul prouvé [2]. Son point faible est que sa construction demande une phase de *trusted setup* dont la compromission permet de forger des preuves indétectables [19].

3.2.2 Les zk-STARKs

Un zk-STARK (*Zero-Knowledge Scalable Transparent Argument of Knowledge*) élimine le trusted setup en s'appuyant exclusivement sur des fonctions de hachage résistantes aux collisions et des techniques d'interpolation polynomiale [12]. Ce qui définit sa transparence totale et sa résistance quantique native. Son point faible est que ce protocole produit une preuve de plusieurs dizaines de kilooctets.

3.2.3 Bulletproofs

Les Bulletproofs [5] sont des arguments courts spécialisés dans les preuves de plage (*range proofs*) : prouver que $v \in [0, 2^k - 1]$ sans révéler v. Pas de trusted setup. La sécurité repose sur le DLP sur courbes elliptiques et sur l'homomorphisme additif des engagements de Pedersen $C = g^v h^r \bmod p$ [1].

3.3 Hypothèses de sécurité des protocoles ZKP

Chaque protocole ZKP repose sur un socle d'hypothèses précises. Rompre l'une d'elles (trouver un algorithme efficace pour le logarithme discret, casser une fonction de hachage, compromettre un trusted setup) suffit à annuler toutes les garanties formelles. Savoir distinguer une *garantie mathématique inconditionnelle* d'une condition pratique (confiance envers un tiers, résistance quantique, modèle de l'oracle aléatoire) est important, c'est le préalable à tout déploiement sérieux [36].

3.3.1 Protocoles ZKP interactifs

Les IZKPs constituent le cadre originel dans lequel les notions de soundness et de zéro-connaissance ont été formalisées par Goldwasser, Micali et Rackoff [36]. Parmi les nombreuses familles identifiées dans la littérature, nous nous concentrons sur les deux plus représentatives : les protocoles classiques (Schnorr, Feige–Fiat–Shamir) et les Σ-protocoles [14].

A — Protocoles interactifs classiques (Schnorr, Feige–Fiat–Shamir)

Ces protocoles partagent une structure commune : un échange en trois tours, une hypothèse de difficulté algorithmique, un adversaire modélisé en PPT. Leur sécurité repose sur deux piliers :

- **Difficulté algorithmique.** La robustesse (soundness) de Schnorr repose sur la difficulté du **logarithme discret** [6]. Celle de Feige–Fiat–Shamir repose sur la difficulté du **calcul de racines carrées modulo un entier composé**, problème équivalent à la factorisation [38]. Si ces problèmes deviennent faciles (via l'algorithme de Shor sur un ordinateur quantique, par exemple) la sécurité s'effondre.
- **Modèle de l'adversaire.** Les preuves de sécurité supposent un adversaire PPT et des défis tirés de façon totalement indépendante de l'engagement [36]. Si le prouveur peut prédire ou biaiser ces défis, la propriété de zéro-connaissance disparaît.

B — Σ-protocoles

Les Σ-protocoles généralisent la structure de Schnorr à n'importe quelle relation arithmétique : logarithme discret, racine carrée, égalités linéaires de la forme $g^x h^y = c$. Même construction en trois tours. Mêmes hypothèses, même fragilité face aux ordinateurs quantiques.

- **Randomisation honnête.** Le défi est tiré de façon aléatoire et indépendante de l'engagement. Toute tentative de simulation par le prouveur reste détectable [14]. C'est la condition minimale de la robustesse (soundness).
- **Difficulté du problème de base.** La sécurité reste conditionnée à la difficulté de la relation cachée. Le cadre des Σ-protocoles est théoriquement élégant ; sa robustesse pratique hérite des limites du problème sous-jacent [14].

3.3.2 Protocoles ZKP non interactifs

Les NIZKPs garantissent robustesse (soundness) et zéro-connaissance sans échange en temps réel. Ce gain de commodité est favorisé par une entrée en jeu de nouvelles hypothèses de sécurité (oracle aléatoire, trusted setup, résistance quantique) qui n'ont pas d'équivalent dans le modèle interactif [24]. Nous allons étudier les trois protocoles dominant de cette famille (la transformation de Fiat–Shamir, les zk-SNARKs, les zk-STARKs et les Bulletproofs) [1, 16, 12].

A — Transformation de Fiat–Shamir

La transformation de Fiat–Shamir convertit un protocole interactif en preuve autonome en remplaçant le défi du vérificateur par le résultat d'une fonction de hachage. Sa sécurité repose sur différentes hypothèses :

- **Hypothèse de l'oracle aléatoire.** La sécurité tient à la condition que la fonction de hachage se comporte comme un oracle aléatoire [1]. Une collision ou une attaque en pré-image sur H compromet directement l'imprévisibilité des défis et donc toutes les preuves construites via cette transformation.
- **Difficulté du problème de base.** La robustesse (soundness) de la version non interactive hérite de celle du protocole d'origine. Si le DLP devient soluble, la transformation de Fiat–Shamir ne compense rien [1].

B — zk-SNARKs

Les zk-SNARKs sont remarquables car elles produisent des preuves très courtes (quelques centaines d'octets) vérifiables quasi instantanément. Cette efficacité dépend de deux hypothèses exigeantes :

- **Difficulté bilinéaire.** Les zk-SNARKs s'appuient sur des hypothèses de type *bilinear hardness* (ex. q-SDH) dans des groupes munis de couplages bilinéaires [16]. Ces hypothèses ne résistent pas à un ordinateur quantique efficace, ce qui constitue un point de vulnérabilité structurelle à long terme [12].
- **Trusted setup.** La phase d'initialisation génère des paramètres publics secrets. Si ces paramètres sont compromis (par une implémentation défaillante ou une entité malveillante) un adversaire peut forger des preuves valides indétectables, même si le modèle PPT est par ailleurs respecté [12, 16].

C — zk-STARKs

Les zk-STARKs répondent à une question précise : peut-on obtenir les performances des SNARKs sans leurs deux vulnérabilités structurelles ? La réponse est oui, partiellement.

- **Sécurité par hachage.** Les STARKs combinent preuves polynomiales, codes correcteurs et fonctions de hachage robustes qui assurent la robustesse (soundnesse) sans bilinéarité [12]. Si les fonctions de hachage sont cassables, la sécurité est affaiblie.

D — Bulletproofs

Les Bulletproofs [5] occupent un positionnement singulier. Sans trusted setup, taille de preuve intermédiaire.

- **Logarithme discret sur courbes elliptiques.** La sécurité repose sur la difficulté du DLP dans des groupes de courbes elliptiques [5]. Robuste aujourd'hui, fragile face à l'algorithme de Shor demain.
- **Modèle PPT.** La probabilité de forger une preuve valide reste négligeable pour tout adversaire PPT, y compris en contexte de collusion [5]. C'est cette garantie qui les rend utilisables dans les systèmes financiers anonymes.

3.4 Analyse comparative des protocoles ZKP

Comprendre un protocole en isolation ne suffit pas. Ce qui compte pour un déploiement réel, c'est le compromis : quelle hypothèse de confiance accepte-t-on ? Quelle taille de preuve supporte-t-on ? Quelle résistance quantique exige-t-on ? Cette section confronte les protocoles étudiées sur ces axes.

3.4.1 IZKPs contre NIZKPs

La transition du modèle interactif vers le non-interactif n'est pas une simple optimisation de performance, c'est un changement d'hypothèses [36]. Les deux paradigmes divergent sur quatre dimensions.

- **Mécanisme d'échange.** Les IZKPs exigent plusieurs tours synchrones : engagement, défi, réponse. Les NIZKPs condensent tout cela en un seul message autonome, généré une fois pour toutes, vérifiable sans interaction [1].
- **Transférabilité.** Une preuve interactive est liée au vérificateur avec qui le prouveur a communiqué ; le transcript n'est pas réutilisable directement [36]. Une preuve non interactive est un objet public : n'importe quel observateur peut la vérifier, de façon asynchrone, sans solliciter le prouveur [16].
- **Hypothèses de sécurité.** Les IZKPs reposent sur la qualité de l'aléa et la structure de l'échange challenge/réponse. Les NIZKPs introduisent des hypothèses supplémentaires (oracle aléatoire, trusted setup) sans équivalent dans le modèle interactif [1, 2].
- **Efficacité.** Les IZKPs paient le prix de la latence de synchronisation : chaque tour d'échange a un coût. Les NIZKPs produisent une preuve compacte et statique, vérifiable à tout moment. Le coût computationnel se déplace vers le prouveur au moment de la génération.

3.4.2 Analyse comparative des IZKPs

Schnorr, Feige–Fiat–Shamir et les Σ-protocoles partagent la même architecture en trois tours. Leurs différences portent sur l'hypothèse de difficulté et sur la généralité

des relations qu'ils peuvent prouver [36, 14].

A — Hypothèses de sécurité

- **Schnorr** est ancré dans le *logarithme discret* [6]. Simple, bien analysé, vulnérable à Shor.
- **Feige–Fiat–Shamir** repose sur la *factorisation* et les racines carrées modulaires [38]. Plus lourd structurellement, mais il contourne le DLP.
- Les **Σ-protocoles** offrent un cadre générique pour des relations arithmétiques variées (log, $\sqrt{}$, relations linéaires) avec des hypothèses proches de Schnorr et Feige–Fiat–Shamir selon le problème sous-jacent [14].

B — Structure et nombre de tours

Trois protocoles, trois tours chacun. La distinction est dans la généralité : Schnorr et FFS ciblent des relations spécifiques ; les Σ-protocoles permettent de les combiner et de les composer [6, 38, 14].

C — Résistance quantique

Aucun des trois ne tient face à un ordinateur quantique efficace. DLP et factorisation sont tous deux solubles par l'algorithme de Shor avec un ordinateur quantique [14, 36]. Pour des applications à horizon décennal, cette limite est structurelle : pas un détail à ignorer.

3.4.3 Analyse comparative des protocoles NIZKP modernes

zk-SNARKs, zk-STARKs, Bulletproofs : trois familles, trois compromis distincts. Elles visent toutes la confidentialité ; elles divergent sur tout le reste [13]. Trois critères structurent la comparaison.

A — Configuration de confiance (Trusted Setup)

- **zk-SNARKs.** Ils exigent un trusted setup. C'est leur talon d'Achille : des paramètres compromis permettent de forger des preuves valides sans jamais être détecté [16].
- **zk-STARKs et Bulletproofs.** Transparents. Aucun secret initial. Les paramètres proviennent de fonctions de hachage publiques ou de courbes elliptiques standards [5]. N'importe qui peut auditer le système depuis le premier jour.

B — Efficience computationnelle

- **zk-STARKs.** Les plus scalables. Leur efficience croît avec la complexité de l'énoncé : plus le calcul à prouver est grand, plus le rapport preuve/calcul est avantageux [12].
- **zk-SNARKs.** Vérification quasi constante : quelques millisecondes, quelle que soit la complexité du calcul prouvé. Idéal pour les smart contracts et les terminaux à ressources limitées [16].
- **Bulletproofs.** Efficaces pour des preuves simples (plages, valeurs bornées). Le temps de vérification croît linéairement avec la taille des données. Pour des calculs complexes, ils sont surpassés par les deux autres [5].

C — Compacité des preuves

- **zk-SNARKs.** Champions de la compacité. Environ 288 octets, quelle que soit la complexité de l'énoncé [16]. C'est ce qui les rend dominants dans les protocoles blockchain où chaque octet a un coût.
- **Bulletproofs.** Taille intermédiaire : 1 à 2 Ko, bien optimisée pour les transactions confidentielles et les preuves de plage [5].
- **zk-STARKs.** La contrepartie de leur transparence et de leur résistance quantique : des preuves de plusieurs dizaines de kilooctets [12]. Acceptable là où la bande passante est abondante et inutilisable ailleurs.

Aucun protocole ne domine sur tous les critères. Le choix dépend du contexte : contraintes de bande passante, horizon temporel de sécurité, tolérance à un trusted setup, complexité des énoncés.

Conclusion

Ce chapitre a cartographié le paysage des protocoles ZKP selon deux axes : les modèles (interactif vs non-interactif) et leurs hypothèses de sécurité. La trajectoire est lisible. De Schnorr et Feige–Fiat–Shamir, ancrés dans des problèmes arithmétiques classiques, jusqu'aux STARKs qui s'affranchissent de toute hypothèse algébrique lourde, chaque génération a repoussé une limite différente (interactivité, confiance, résistance quantique, compacité).

L'analyse comparative le confirme : il n'existe pas de protocole universel. Chaque famille résout un problème et en laisse un autre ouvert. Ce n'est pas un échec ; c'est la structure normale d'un domaine vivant. Ces bases théoriques et comparatives étant

posées, le chapitre suivant met en évidence les applications, limites et perspectives des ZKPs.

$$\Pi_{ZK} : \mathsf{Prove}(x \in L) \wedge \neg\mathsf{Reveal}(x)$$

CHAPITRE 4

APPLICATIONS, LIMITES, PERSPECTIVES ET OUVERTURES DES ZKPS

Introduction

Les chapitres précédents ont posé les fondements théoriques et comparé les architectures. Ce chapitre change d'angle : à quoi servent réellement les ZKP, qu'est-ce qui freine leur déploiement, et vers quoi pointent-ils ? Les cas d'usage couvrent des secteurs aussi différents que la finance, la santé et le vote électronique [22]. Les limites sont tout aussi réelles : coût computationnel, trusted setup, fragilité quantique à l'horizon [29, 16, 12]. Les réponses à ces questions ont des implications directes, pas seulement techniques, mais politiques et souveraines [19].

4.1 Applications des ZKPs

Les ZKPs ne sont pas qu'un objet théorique. Ils s'imposent aujourd'hui comme une primitive cryptographique pivot pour concilier la transparence des systèmes décentralisés et la confidentialité des données privées. En s'appuyant notamment sur la classification proposée par **le cabinet de cybersécurité LockSelf**[22], cette section synthétise les cas d'usage majeurs qui traversent plusieurs secteurs critiques.

4.1.1 Transactions financières sensibles

Le protocole ZKP obscurcit les flux financiers tout en garantissant leur intégrité. Il atteste de la validité d'une opération (vérification de la solvabilité, signature numérique) sans compromettre les variables sensibles : montants exacts, adresses des portefeuilles impliqués. La preuve est là mais le secret reste caché.

4.1.2 Audits et vérifications

Dans un cadre organisationnel, cette technologie permet à une entité de démontrer la conformité de ses calculs ou le respect de ses engagements contractuels sans révéler ses données opérationnelles ni ses paramètres témoins (*witnesses*).Cette approche assure une vérifiabilité totale pour les régulateurs tout en préservant l'étanchéité du secret industriel.

4.1.3 Partage de données médicales

Le ZKP facilite l'agrégation de statistiques cliniques ou le partage d'indicateurs pathologiques sans exposition des données brutes des patients. Les chercheurs collaborent sur des preuves de résultats. Les données individuelles ne bougent pas. Ainsi ils garantissaient une confidentialité mathématique absolue, conforme aux exigences de protection de la vie privée.

4.1.4 Sécurité des mots de passe

Le ZKP transforme les mécanismes d'authentification : l'utilisateur prouve la possession d'un secret sans que celui-ci, ni son condensat (*hash*), ne transite sur le réseau. Le vérificateur valide une preuve générée localement. Interception impossible et attaque par rejeu neutralisée. Considérant que la majorité des cyberattaques documentées exploitent des identifiants compromis selon les rapports annuels sur la sécurité [21], ce paradigme renforce substantiellement la résilience des accès.

4.1.5 Stockage et partage de fichiers

Lors de l'échange de documents, une preuve cryptographique atteste de l'intégrité ou de propriétés spécifiques du fichier sans en divulguer le contenu exhaustif. Dans un environnement *cloud*, le fournisseur prouve la persistance et l'exactitude des données stockées sans disposer de la capacité technique de les lire. Assurant ainsi une architecture de confiance distribuée, sans accès centralisé.

4.2 Limites des ZKPs

Les ZKPs offrent des garanties de confidentialité remarquables. Mais leur mise en œuvre pratique se heurte à plusieurs contraintes structurelles : puissance de calcul, confiance initiale, scalabilité. Ces limites ne sont pas secondaires. Elles conditionnent directement les choix de déploiement.

4.2.1 La charge computationnelle et le coût de génération

La principale contrainte des *ZKP* est la puissance de calcul que le prouveur doit mobiliser. Générer une preuve mathématique, particulièrement pour des transactions complexes, consomme des ressources CPU et une mémoire vive considérables. Comme le montre l'analyse de [13], un arbitrage constant s'impose entre la taille de la preuve et le temps nécessaire pour la produire. Dans le cas des *Bulletproofs*, la preuve est légère mais le temps de calcul reste supérieur à celui des solutions traditionnelles non cryptographiées. Ce qui freine l'adoption sur des terminaux à ressources limitées [5].

4.2.2 La configuration de confiance (*Trusted Setup*)

Une limite critique propre aux *zk-SNARKs* : la nécessité d'une phase d'initialisation sécurisée. Ce processus génère des paramètres publics à partir de secrets qui doivent impérativement être détruits. Si ces « déchets toxiques » subsistent, un attaquant peut théoriquement forger de fausses preuves [16]. Cette dépendance à une honnêteté initiale est un point de vulnérabilité structurel. C'est précisément pourquoi des protocoles comme les *Bulletproofs* [5] cherchent à s'en affranchir pour atteindre une sécurité dite « sans confiance » (*trustless*).

4.2.3 Les compromis de vérification et de stockage

La scalabilité des réseaux blockchain est directement impactée par la nature des preuves choisies. Les protocoles interactifs multiplient les échanges pour atteindre une probabilité d'erreur négligeable $1/2^n$: impraticable pour un registre distribué [36]. En mode non interactif, les *zk-STARKs* génèrent des preuves volumineuses qui saturent l'espace de bloc [12]. Les *Bulletproofs* optimisent l'espace de stockage, mais imposent un temps de vérification plus long pour les nœuds du réseau (un nouveau goulot d'étranglement) lors de la validation de blocs massifs [5]. Chaque famille déplace le problème, aucune ne le supprime.

4.2.4 La menace de l'informatique quantique

Une limite prospective pèse sur l'ensemble du domaine : la pérennité des hypothèses de sécurité. La majorité des *ZKP* actuels (*Bulletproofs*, *zk-SNARKs*) reposent sur la difficulté du logarithme discret sur les courbes elliptiques. L'algorithme de Shor [29] rendrait ces preuves caduques : les secrets deviendraient retrouvables en temps polynomial [12]. Un ordinateur quantique suffisamment puissant changerait les règles du jeu. Seuls les protocoles basés sur le hachage (*hash-based*), comme les *zk-STARKs*, résistent à cette menace future [12].

4.3 Perspectives d'évolution des ZKPs

La robustesse démontrée des protocoles ZKP laisse présager une extension majeure de leur usage au sein de nos infrastructures numériques. Cette section s'appuie sur l'analyse prospective menée par **LockSelf**[22] pour explorer comment cette transition impacte les piliers fondamentaux de nos sociétés numériques, notamment à travers quatre axes majeurs : le vote électronique, la gouvernance décentralisée, la gestion de l'identité numérique et l'éducation.

4.3.1 Vote électronique et démocratie numérique : le contexte sénégalais

Le vote électronique incarne la tension la plus aiguë entre les exigences de la démocratie et les contraintes de la cryptographie. Un scrutin numérique doit simultanément garantir l'anonymat absolu de l'électeur et la transparence totale du décompte. Deux exigences qui paraissent contradictoires mais ne le sont pas, pourvu que les outils cryptographiques soient bien choisis.

Dans de nombreux pays africains, dont le Sénégal, la numérisation des processus électoraux est perçue comme un levier de modernisation de l'État et de renforcement de la confiance citoyenne. Cependant, les solutions déployées restent confrontées à un dilemme structurel. Soit elles garantissent la traçabilité au détriment de l'anonymat, soit elles assurent la confidentialité mais ouvrent la porte à des contestations sur l'intégrité du décompte. La blockchain couplée aux ZKP offre une perspective concrète : un décompte exact et public des voix, avec un anonymat absolu de l'électeur [3, 36].

A — Acteurs institutionnels sénégalais

Deux institutions pilotent la gestion électorale au Sénégal :

- **La Direction Générale des Élections (DGE)**, rattachée au Ministère de l'Intérieur, est responsable de l'organisation matérielle des scrutins nationaux. Elle gère le fichier électoral biométrique national, qui regroupe les données d'inscription, les empreintes digitales et les informations de la carte nationale d'identité [7, 32].
- **La Commission Électorale Nationale Autonome (CENA)**, créée par la loi n° 2005-07 du 11 mai 2005 [31], est l'organe indépendant de supervision et de contrôle des élections. Structure permanente dotée de la personnalité juridique et de l'autonomie financière [31], elle contrôle et supervise l'ensemble des opérations électorales et référendaires, et veille à leur régularité, transparence et sincérité.

À ce jour, le Sénégal n'a pas déployé de système de vote en ligne pour les élections nationales : le vote se déroule en bureau physique, le bulletin papier restant la norme [32].

Des systèmes numériques sont cependant utilisés pour la transmission des procès-verbaux, la consultation des listes électorales et la délivrance des cartes d'électeur biométrique [7].

Note de contexte — réforme institutionnelle en cours (2026). Un avant-projet de loi portant nouveau Code électoral du Sénégal, examiné en 2026, prévoit de remplacer la CENA et la DGE par une Commission Électorale Nationale Indépendante (CENI) et d'introduire une dématérialisation progressive des opérations électorales (parrainage numérique, transmission automatisée des résultats). Ce contexte renforce la pertinence de travaux comme Π_{VoteZK}, qui fournit les fondements cryptographiques nécessaires à un tel déploiement.

B — Plateformes de vote électronique dans les institutions sénégalaises

À l'échelle des universités et institutions publiques, plusieurs expériences de vote électronique ont été menées pour les élections étudiantes et les votes de conseils. Les solutions utilisées appartiennent à deux catégories :

- Formulaires en ligne génériques (Google Forms, Microsoft Forms) : aucune garantie cryptographique. L'opérateur de la plateforme a un accès complet aux données. L'anonymat de l'électeur dépend exclusivement de la politique de confidentialité du prestataire, hors de contrôle de l'institution.
- Systèmes de vote en ligne commerciaux : ces plateformes proposent un dépouillement automatisé. Certaines intègrent des mécanismes d'authentification par OTP. Cependant, le secret du scrutin repose sur une *confiance aveugle* dans le serveur central : l'administrateur de la plateforme peut, en principe, accéder aux bulletins individuels [3].

C — Helios Voting : la référence académique internationale

Helios Voting [3] est le premier système de vote électronique académique à audit ouvert, publiquement accessible. Son architecture repose sur :

- Un chiffrement homomorphe des bulletins (ElGamal) permettant un décompte sans déchiffrement individuel [3].
- Des preuves à divulgation nulle (Σ-protocoles, Définition 1.28) attestant de la validité de chaque bulletin chiffré.
- Un tableau de bord public permettant à n'importe quel observateur de vérifier le décompte [3].

En mars 2009, l'Université catholique de Louvain a élu son Président par un déploiement personnalisé d'Helios [4] : sur 25 000 électeurs potentiels, 5 000 se sont

inscrits et près de 4 000 ont voté à chaque tour. La précision du système s'est révélée cruciale : au premier tour, le leader n'a manqué la victoire que de 2 voix [4].

Cependant, Cortier et Smyth [39] ont identifié une vulnérabilité permettant à un adversaire de compromettre la confidentialité des bulletins par rejeu. La vulnérabilité exploite l'absence d'*indépendance des bulletins* (*ballot independence*) dans Helios et permet, en rejouant le bulletin d'un électeur ou une variante de celui-ci, d'amplifier sa contribution au décompte de façon à inférer le vote individuel [39]. Les auteurs proposent une correction formelle satisfaisant une définition de sécurité en calcul des processus appliqué [39].

Au-delà de cette vulnérabilité corrigée dans les versions ultérieures, trois limites structurelles subsistent dans le contexte sénégalais :

- **Registre d'éligibilité non ZKP :** la vérification de l'inscription repose sur une liste nominative en clair. Le lien entre le compte utilisateur et le bulletin constitue une surface d'attaque pour la désanonymisation [3].
- **Déchiffrement centralisé :** la phase finale requiert une autorité détenant la clé de déchiffrement. Si cette clé est compromise, l'anonymat de l'ensemble des électeurs est perdu rétrospectivement [4].
- **Taille de preuve :** les preuves ElGamal croissent avec le nombre de candidats ; pour des réseaux à faible bande passante, cela peut constituer une contrainte pratique.

D — Tableau comparatif et apport de Π_{VoteZK}

TABLE 4.1 – Analyse comparative des solutions de vote électronique et apport de Π_{VoteZK}.

Solution	**Anonymat**	**Vérif. pub.**	**Registre ZKP**	**Anti-double vote**
Formulaires génériques	×	×	×	×
Solutions commerciales	Partielle	×	×	Serveur
Helios Voting [3, 39]	Partielle[a]	✓	×	Liste en clair
Π_{VoteZK} (ce travail)	✓ ZKP	✓	✓ Merkle	✓ Nullifieur

[a] Vulnérabilité de rejeu identifiée par Cortier & Smyth [39] ; corrigée dans les versions ultérieures.

Le Tableau 4.1 fait apparaître un manque commun : **aucune solution existante ne fournit simultanément l'anonymat cryptographique, la vérifiabilité publique, un registre d'éligibilité à divulgation nulle et une protection anti-double-vote sans tiers de confiance**. C'est précisément ce verrou que Π_{VoteZK} lève au Chapitre 5.

4.3.2 Gouvernance résiliente et transparente

Les ZKP permettent de concilier confidentialité et redevabilité au sein des organes de décision. Une délibération peut être prouvée conforme aux règles établies sans révéler les détails sensibles des débats internes. La gouvernance gagne en équité et la responsabilité reste traçable.

4.3.3 Sécurisation de l'Internet des Objets (IoT)

Face à l'explosion des cyberattaques visant les objets connectés, dont les rapports sur les menaces informatiques font état d'une forte croissance annuelle [42], le ZKP propose une alternative aux solutions EDR (*Endpoint Detection and Response*) classiques :

- **Authentification :** Les appareils prouvent leur identité sur le réseau sans échanger de clés d'accès ou d'identifiants vulnérables.
- **Intégrité des données :** Un capteur (par exemple médical) prouve la véracité des données collectées sans révéler leur contenu brut, limitant ainsi les risques de manipulation.

4.3.4 Applications dans le domaine de l'éducation

Le ZKP peut transformer la gestion des examens en ligne et des processus de candidature, en luttant contre la fraude et la discrimination :

- **Évaluations :** Garantir l'authenticité des résultats d'un candidat tout en préservant son anonymat vis-à-vis des correcteurs.
- **Candidatures :** Permettre aux étudiants de prouver l'obtention de diplômes ou de bourses sur la base du mérite, sans exposer d'autres informations personnelles sensibles : réduisant les biais cognitifs et les risques de fuites de données.

4.4 Ouverture des ZKPs

Malgré les défis techniques réels, les ZKPs ouvrent des perspectives majeures pour la sécurisation des infrastructures numériques de demain. Trois axes de développement structurent cette ouverture.

4.4.1 Standardisation et interopérabilité des protocoles

Le foisonnement actuel des solutions (*zk-SNARKs, STARKs, Bulletproofs*) appelle à une standardisation accrue. Comme l'indique l'analyse de [13], chaque protocole répond à des besoins spécifiques de stockage ou de vitesse. L'avenir réside probablement dans l'interopérabilité de ces systèmes, permettant par exemple d'utiliser la légèreté

des *Bulletproofs* pour les transactions mobiles, tout en s'appuyant sur la résistance quantique des *STARKs* pour l'archivage de données d'État à long terme [12].

4.4.2 L'émergence de l'Identité Numérique Décentralisée

L'application la plus prometteuse hors du secteur purement financier concerne l'identité numérique (*Self-Sovereign Identity*). En permettant de prouver des attributs (citoyenneté, diplômes, solvabilité) sans exposer les données brutes, les *ZKP* offrent une réponse concrète aux exigences du RGPD et des lois sur la protection des données personnelles [19]. Pour un pays comme le Sénégal, cela représente une opportunité de moderniser les services publics (état civil, bourses universitaires) tout en garantissant la vie privée des citoyens.

4.4.3 Défis éducatifs et souveraineté technologique

Enfin, l'ouverture doit être académique et structurelle. La complexité mathématique des preuves à divulgation nulle nécessite une formation spécialisée des ingénieurs et auditeurs système. Comme le souligne [19], la compréhension de ces outils est indispensable pour ne pas subir les technologies importées. La maîtrise de la cryptographie avancée est, à terme, un levier de souveraineté permettant de construire des solutions locales adaptées aux réalités socio-économiques du continent, garantissant ainsi une indépendance numérique durable.

4.5 Conclusion

Ce chapitre a permis de mettre en perspective le potentiel transformateur des preuves à divulgation nulle de connaissance. Si les limites actuelles liées à la charge computationnelle et à la menace quantique sont réelles, les avancées constantes dans la standardisation des protocoles ouvrent la voie à une adoption massive. Que ce soit pour sécuriser l'Internet des Objets, garantir l'intégrité des processus démocratiques ou instaurer une identité numérique souveraine, le ZKP s'affirme comme le chaînon manquant entre sécurité absolue et respect de la vie privée.

En définitive, la maîtrise de ces outils cryptographiques dépasse le simple cadre technique pour devenir un enjeu de confiance numérique globale, essentiel au développement d'infrastructures résilientes. C'est précisément pour concrétiser ces opportunités et répondre à ces défis théoriques que le chapitre suivant sera consacré à l'implémentation pratique d'un protocole ZKP spécifique, nommé Π_{VoteZK}.

$$\Pi_{\mathsf{ZK}} : \mathsf{Prove}(x \in L) \ \wedge \ \neg\mathsf{Reveal}(x)$$

CHAPITRE 5

Π_{VoteZK} : DE LA FORMALISATION À LA VALIDATION EXPÉRIMENTALE

Introduction

Les chapitres précédents ont établi les fondements mathématiques, formalisé les propriétés des preuves à divulgation nulle de connaissance, analysé leurs modèles de sécurité, et identifié le vote électronique comme application majeure [36, 27]. Ce cinquième chapitre concrétise cette trajectoire. Il construit, prouve et valide Π_{VoteZK}, un protocole de vote électronique original garantissant qu'un électeur puisse prouver la validité de son vote (binaire ($\mathsf{vote} \in \{0, 1\}$), provenant d'un inscrit, non soumis en double) *sans jamais en révéler le contenu.*

Le protocole repose sur quatre primitives : la signature de Schnorr [6] transformée par Fiat-Shamir [1] pour la non-interactivité, les engagements de Pedersen [37] pour la confidentialité et le décompte homomorphe, la OR-proof CDS [30] pour la vérification cryptographique de la binarité des bulletins, et les arbres de Merkle pour l'éligibilité sans révélation d'identité. Toutes les preuves de sécurité sont conduites dans le **modèle de l'oracle aléatoire** [23] (défini au § 1.4.2), hypothèse standard pour la transformation de Fiat-Shamir. La sécurité dans le modèle standard reste une question ouverte.

Ce chapitre s'articule en six sections : spécification formelle, conception du protocole, implémentation Python, validation expérimentale, analyse de sécurité, puis limites et perspectives.

5.1 Spécification formelle du problème de vote électronique

Construire un protocole correct commence par poser précisément le problème qu'il résout. Cette section définit le modèle d'adversaire, les cinq propriétés requises du système Π_{VoteZK}, et la relation de preuve formelle qui les traduit en contraintes mathématiquement vérifiables.

5.1.1 Modèle d'adversaire et propriétés requises

Un système de vote électronique sécurisé doit garantir simultanément des propriétés qui semblent *a priori* contradictoires : la transparence du décompte et l'anonymat de l'électeur. Nous considérons le modèle d'adversaire suivant :

Modèle d'adversaire. Trois types d'adversaires PPT sont considérés :

i. **Électeur malveillant :** tente de voter plusieurs fois (attaque de double-vote) ou de soumettre un bulletin invalide ($\mathsf{vote} \notin \{0, 1\}$).

ii. **Scrutateur passif malveillant :** observe les preuves publiées et tente d'en déduire le contenu des bulletins. Il ne modifie pas les preuves.

iii. **Observateur externe :** intercepte les communications et tente de lier une preuve à un électeur identifiable.

Le scrutateur est supposé *honnête mais curieux* : il respecte le protocole mais tente d'extraire de l'information des preuves reçues. Un scrutateur malveillant actif (capable de forger des preuves) est hors périmètre de ce prototype ; son traitement nécessiterait un schéma de chiffrement homomorphe complet [27].

Conformément à la littérature fondatrice [36, 27], nous spécifions les obligations formelles suivantes.

Propriétés requises du système Π_{VoteZK}

1. **Éligibilité :** seul un électeur enregistré peut soumettre un bulletin. L'appartenance au registre est prouvée sans révéler l'identité.
2. **Unicité :** chaque électeur ne peut voter qu'une seule fois. Le nullifieur (*nullifier*) détecte les doublons sans désanonymiser l'électeur.
3. **Validité du vote :** chaque bulletin satisfait $\mathsf{vote} \in \{0, 1\}$, prouvé cryptographiquement par OR-proof CDS [30] sans révéler la valeur.
4. **Confidentialité :** le contenu du bulletin reste opaque pour tout adversaire

PPT, y compris l'autorité de scrutin.

5. **Vérifiabilité :** tout observateur peut vérifier la validité des preuves et l'exactitude du décompte.

5.1.2 Relation de preuve formelle

La relation de preuve que le protocole doit satisfaire est :

$$\mathcal{R}_{\mathsf{Vote}} = \left\{ \Big(\underbrace{(C, \mathsf{nul}, \mathsf{root})}_{\text{instance publique}}, \underbrace{(\mathsf{vote}, r, \mathsf{sk}, \mathsf{path})}_{\text{témoin secret}} \Big) \;\middle|\; (1)\text{–}(4) \text{ ci-dessous} \right\} \tag{5.1.1}$$

où les quatre conditions sont :

$$(1) \quad \mathsf{vote} \in \{0, 1\}, \tag{5.1.2}$$

$$(2) \quad C = g^{\mathsf{vote}} \cdot h^{r} \pmod{p}, \tag{5.1.3}$$

$$(3) \quad \mathsf{nul} = H(\mathsf{sk} \| \varepsilon), \tag{5.1.4}$$

$$(4) \quad \mathsf{MerkleVerify}\big(\mathsf{root}, H(\mathsf{pk}), \mathsf{path}\big) = \top. \tag{5.1.5}$$

Ici $\mathsf{pk} = g^{\mathsf{sk}} \bmod p$ est la clé publique de l'électeur, $r \xleftarrow{\$} \mathbb{Z}_q$ est l'aléa de l'engagement de Pedersen, path est le chemin de Merkle attestant l'inscription au registre, ε est l'identifiant d'élection public, et root est la racine publique de l'arbre.

Remarque 5.1. La condition (4) utilise $H(\mathsf{pk})$, non $H(\mathsf{sk})$: c'est la clé *publique* $\mathsf{pk} = g^{\mathsf{sk}}$ qui est insérée dans l'arbre de Merkle lors de l'enregistrement, conformément à l'implémentation dans `merkle.py`.

5.2 Conception du protocole Π_{VoteZK}

Les spécifications étant posées, cette section décrit les choix de conception de Π_{VoteZK} : quelles primitives, pourquoi, et comment elles s'articulent. Le protocole est non-interactif et s'organise en quatre phases.

5.2.1 Choix de conception et justification

Π_{VoteZK} repose sur quatre primitives dont les propriétés ont été établies dans les chapitres précédents (Tableau 5.1).

TABLE 5.1 – Primitives du protocole Π_{VoteZK} et leurs justifications

Primitive	Propriété exploitée	Justification
Schnorr [6] + Fiat-Shamir [1]	Non-interactivité, preuve DLP	Prouve la possession de sk sans interaction.
Engagement de Pedersen [37]	Hiding parfait + homomorphisme	Cache le bulletin, décompte sans déchiffrement.
OR-proof CDS [30]	Preuve disjonctive $v \in \{0, 1\}$	Propriété 3 : validité binaire cryptographiquement prouvée.
Arbre de Merkle + SHA-256	Appartenance sans révélation	Prouve l'inscription sans divulguer l'identité.

Note sur la distinction preuve vs argument. Un protocole offre une *preuve* (*proof*) si la robustesse est inconditionnelle, indépendamment de la puissance de l'adversaire. Il offre un *argument* si la robustesse n'est garantie que contre un adversaire PPT. Π_{VoteZK}, comme tout protocole fondé sur le DLP et la transformation de Fiat-Shamir, est un *argument* : sa robustesse est computationnelle, non inconditionnelle [26]. Cette distinction importe en contexte post-quantique : un argument fondé sur le DLP ne résisterait pas à un adversaire disposant d'un ordinateur quantique efficace exécutant l'algorithme de Shor [29].

5.2.2 Architecture du protocole

Π_{VoteZK} est non-interactif (NIZKP). Quatre phases le structurent : Setup, Registration, Voting et Tallying.

5.2.3 Phase 1 — Setup : génération des paramètres

L'autorité de scrutin génère les paramètres publics du système :

- Un premier sûr p tel que $q = (p-1)/2$ soit premier ($\lambda = 128$ bits pour le prototype ; ≥ 256 bits en production selon le NIST [40]).
- Un générateur g du sous-groupe d'ordre q de $\mathbb{Z}_p^*$.
- Second générateur h produit par méthode *nothing-up-my-sleeve* : $h \leftarrow \mathsf{HashToGroup}(\text{"VoteZK-h-2025"})$, garantissant $\log_g h$ inconnu [37].
- Un arbre de Merkle vide $\mathcal{M}$ avec SHA-256 et un identifiant d'élection ε.

❖ Les paramètres publics : $\mathsf{pp} = (p, q, g, h, \varepsilon)$.

Paramètre de sécurité du prototype ($\lambda = 128$ bits). Dans cette implémentation, p est un premier sûr de 128 bits, ce qui donne q de 127 bits. Comme expliqué au § 1.2.1, le NFS réduit la sécurité effective à quelques dizaines de bits seulement dans $\mathbb{Z}_p^*$ pour ce paramètre. Ce choix est *délibérément pédagogique* : il permet d'exécuter les benchmarks en millisecondes sur un ordinateur personnel. En production, on doit utiliser :

- ❖ $p \geq 2048$ bits dans $\mathbb{Z}_p^*$, offrant 112 bits de sécurité effective [25] ;
- ❖ ou migrer vers Curve25519 dont le corps à $2^{255} - 19$ éléments offre ≈ 128 bits de sécurité avec des clés 10 fois plus petites [8].

Les théorèmes de sécurité du § 5.5 s'appliquent pour tout λ et tout groupe vérifiant les hypothèses DLP et DDH.

5.2.4 Phase 2 — Registration : inscription des électeurs

Chaque électeur éligible génère $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q^*$, calcule $\mathsf{pk} = g^{\mathsf{sk}} \bmod p$, et soumet pk à l'autorité. Celle-ci insère pk dans $\mathcal{M}$ et publie $\mathsf{root} = \mathsf{root}(\mathcal{M})$. L'électeur reçoit son chemin path.

5.2.5 Phase 3 — Voting : génération de la preuve

L'électeur exécute l'Algorithme 1.

Algorithme 1 : Algorithme du prouveur $\mathcal{P}$ dans Π_{VoteZK} (ROM)

Entrées : $\mathsf{vote} \in \{0,1\}$, sk, path, pp

Sortie : $\pi = (C, \mathsf{nul}, \mathsf{root}, \mathsf{or_pi}, A, B, z_v, z_r, z_{sk}, \mathsf{path})$

`/* Étape 1 -- Engagement de Pedersen */`

$r \xleftarrow{\$} \mathbb{Z}_q$

$C \leftarrow g^{\mathsf{vote}} \cdot h^r \pmod p$

`/* Étape 2 -- Calcul du nullifieur */`

$\mathsf{nul} \leftarrow H(\mathsf{sk} \| \varepsilon)$

`/* Étape 2b -- OR-proof CDS (Propriété 3 : vote ∈ {0,1}) [30] */`

$\mathsf{or_pi} \leftarrow \mathsf{GenererOrProof}(\mathsf{vote}, r, C, \mathsf{pp})$

`/* Étape 3 -- Aléas éphémères (masques) */`

$k_v, k_r, k_{sk} \xleftarrow{\$} \mathbb{Z}_q$

$A \leftarrow g^{k_v} \cdot h^{k_r} \pmod p$

$B \leftarrow g^{k_{sk}} \pmod p$

`/* Étape 4 -- Défi Fiat-Shamir (ROM) */`

$c \leftarrow H(g \| h \| C \| \mathsf{nul} \| \mathsf{root} \| A \| B \| \varepsilon) \bmod q$

`/* Étape 5 -- Réponses du prouveur */`

$z_v \leftarrow (k_v + c \cdot \mathsf{vote}) \bmod q$

$z_r \leftarrow (k_r + c \cdot r) \bmod q$

$z_{sk} \leftarrow (k_{sk} + c \cdot \mathsf{sk}) \bmod q$

retourner π

5.2.6 Phase 4 — Tallying : vérification et décompte

Le vérificateur exécute l'Algorithme 2 pour chaque bulletin soumis.

Algorithme 2 : Algorithme du vérificateur $\mathcal{V}$ dans Π_{VoteZK}

Entrées : π, $\mathcal{N}$ (nullifieurs vus), pp
Sortie : $\top$ (valide) ou $\bot$ (rejeté)

`/* Étape 1 -- Détection du double-vote */`
si $\mathsf{nul} \in \mathcal{N}$ **alors retourner** $\bot$
$\mathcal{N} \leftarrow \mathcal{N} \cup \{\mathsf{nul}\}$

`/* Étape 2 -- Recalcul du défi Fiat-Shamir (ROM) */`
$c' \leftarrow H(g\|h\|C\|\mathsf{nul}\|\mathsf{root}\|A\|B\|\varepsilon) \bmod q$

`/* Étape 3 -- Schnorr : engagement du vote */`
si $g^{z_v} \cdot h^{z_r} \not\equiv A \cdot C^{c'} \pmod{p}$ **alors retourner** $\bot$

`/* Étape 4 -- Schnorr : connaissance de la clé secrète */`
si $g^{z_{sk}} \not\equiv B \cdot \mathsf{pk}^{c'} \pmod{p}$ **alors retourner** $\bot$

`/* Étape 5 -- OR-proof CDS : vote ∈ {0,1} (Propriété 3) [30] */`
si $\mathsf{VerifierOrProof}(C, \mathsf{or_pi}, \mathsf{pp}) \neq \top$ **alors retourner** $\bot$

`/* Étape 6 -- Merkle : appartenance au registre */`
si $\mathsf{MerkleVerify}(\mathsf{root}, H(\mathsf{pk}), \mathsf{path}) \neq \top$ **alors retourner** $\bot$

retourner $\top$

$\Pi_{\mathsf{ZK}} : \mathsf{Prove}(x \in L) \wedge \neg\mathsf{Reveal}(x)$

Décompte homomorphe. Grâce à l'homomorphisme additif de l'engagement de Pedersen :

$$C_{\mathsf{total}} = \prod_{i=1}^{N} C_i \pmod{p} = g^{\sum_i \mathsf{vote}_i} \cdot h^{\sum_i r_i} \pmod{p}. \tag{5.2.6}$$

Le résultat $\sum_i \mathsf{vote}_i$ est retrouvé par l'algorithme BSGS de Shanks sur $[0, N]$, en complexité $\mathcal{O}(\sqrt{N})$.

5.3 Implémentation en Python

La formalisation est validée par le code. Cette section présente les sept modules de l'implémentation de référence, les conventions de sérialisation adoptées, et la structure de la preuve π. Chaque module correspond directement à une primitive ou une phase du protocole.

5.3.1 Architecture du code

Le projet est découpé en **sept** modules :

- `params.py` : génération des paramètres publics pp.
- `merkle.py` : arbre de Merkle avec SHA-256.

- `or_proof.py` : OR-proof CDS pour la Propriété 3 ($\mathsf{vote} \in \{0, 1\}$).
- `prover.py` : Algorithme 1.
- `verifier.py` : Algorithme 2.
- `tally.py` : décompte homomorphe et BSGS.
- `test_votezk.py` : suite de 10 tests unitaires.

Remarque 5.2 (Convention de sérialisation). Tous les modules utilisent une sérialisation **uniforme de 32 octets** (`pk.to_bytes(32, "big")`). Pour $\lambda = 128$ bits, p occupe 17 octets ; le complément à 32 octets garantit la cohérence entre `merkle.py` et `verifier.py`.

Remarque 5.3 (Taille réelle des preuves). Avec la sérialisation sur 32 octets, la structure de π est :

- Partie Schnorr fixe : $7 \times 32 = 224$ oct. pour $(C, A, B, z_v, z_r, z_{sk}, \mathsf{pk})$.
- Condensés SHA-256 : $2 \times 32 = 64$ oct. pour $(\mathsf{nul}, \mathsf{root})$.
- OR-proof CDS : $8 \times 32 = 256$ oct. pour $(A_0, A_1, c_0, c_1, z_{0v}, z_{0r}, z_{1v}, z_{1r})$.
- **Total fixe :** $224 + 64 + 256 = \mathbf{544}$ **oct.**
- Chemin de Merkle : $33\lceil \log_2 N \rceil$ oct.

Soit :

$$|\pi| = 544 + 33\lceil \log_2 N \rceil \text{ octets.}$$

5.3.2 Module `params.py`

```python
import secrets
import hashlib
import time
def est_premier(n, k=25):
    if n < 2: return False
    if n == 2 or n == 3: return True
    if n % 2 == 0: return False
    d, r = n - 1, 0
    while d % 2 == 0:
        d //= 2; r += 1
    for _ in range(k):
        a = secrets.randbelow(n - 3) + 2
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
```

```
            if x == n - 1:
                break
        else:
            return False
    return True
def generer_premier_sur(bits=128):
    tentatives = 0
    while True:
        tentatives += 1
        q = secrets.randbits(bits - 1) | (1 << (bits - 2)) | 1
        if est_premier(q):
            p = 2 * q + 1
            if est_premier(p):
                return p, q, tentatives
def trouver_generateur(p, q):
    while True:
        g = secrets.randbelow(p - 2) + 2
        if pow(g, q, p) == 1 and pow(g, 2, p) != 1:
            return g
def generer_h_ntms(p, q, g, label=b"VoteZK-h-2025"):
    """h genere par HashToGroup : log_g(h) inconnu de tout le monde."""
    x_bytes = hashlib.sha256(label).digest()
    x_int = int.from_bytes(x_bytes, "big") % q
    if x_int == 0 or x_int == 1:
        return generer_h_ntms(p, q, g, label + b"\x01")
    h = pow(g, x_int, p)
    return h
def configuration(bits=128, epsilon=b"election-2025"):
    debut = time.time()
    p, q, tentatives = generer_premier_sur(bits)
    g = trouver_generateur(p, q)
    h = generer_h_ntms(p, q, g)
    duree = (time.time() - debut) * 1000
    print(f"[params] p={bits} bits | {tentatives} tentatives | {duree:.1f}
    ms")
    print(f"[params] h genere par HashToGroup (log_g(h) inconnu)")
    return {"p": p, "q": q, "g": g, "h": h, "epsilon": epsilon}
```

Code 5.1 – `params.py` — génération des paramètres publics `pp`.

5.3.3 Module `merkle.py`

```python
import hashlib

def hacher_sha256(donnees: bytes) -> bytes:
    return hashlib.sha256(donnees).digest()

class ArbreMerkle:
    def __init__(self, feuilles: list[bytes]):
        if not feuilles:
            raise ValueError("Au moins une feuille requise.")
        self.feuilles_brutes = feuilles
        # AJOUT : Préfixe 0x00 pour différencier les feuilles
        self.feuilles = [hacher_sha256(b'\x00' + f) for f in feuilles]
        self.niveaux = self._construire(self.feuilles)

    def _construire(self, noeuds):
        niveaux = [noeuds[:]]
        while len(niveaux[-1]) > 1:
            couche = niveaux[-1]
            if len(couche) % 2 == 1:
                couche = couche + [couche[-1]]
            # AJOUT : Préfixe 0x01 pour différencier les nœuds internes
            niveaux.append([hacher_sha256(b'\x01' + couche[i] + couche[i + 1])
                            for i in range(0, len(couche), 2)])
        return niveaux

    @property
    def racine(self) -> bytes:
        return self.niveaux[-1][0]

    def obtenir_preuve(self, index: int):
        preuve = []
        for niveau in self.niveaux[:-1]:
            if len(niveau) % 2 == 1:
                niveau = niveau + [niveau[-1]]
            if index % 2 == 0:
                preuve.append((niveau[index + 1], "droite"))
            else:
                preuve.append((niveau[index - 1], "gauche"))
            index //= 2
```

```
        return preuve

def verifier_preuve_merkle(racine, donnees_feuille, preuve):
    # AJOUT : Préfixe 0x00 lors du hachage initial de la feuille à vérifier
    noeud = hacher_sha256(b'\x00' + donnees_feuille)
    for frere, direction in preuve:
        if direction == "droite":
            # AJOUT : Préfixe 0x01 lors de la reconstruction des nœuds
    parents
            noeud = hacher_sha256(b'\x01' + noeud + frere)
        else:
            # AJOUT : Préfixe 0x01 lors de la reconstruction des nœuds
    parents
            noeud = hacher_sha256(b'\x01' + frere + noeud)
    return noeud == racine
```

Code 5.2 – `merkle.py` — arbre de Merkle et vérification de chemin.

5.3.4 Module `or_proof.py`

```
import secrets
import hashlib
def _defi_or(pp, C, A0, A1):
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    oct_p = (p.bit_length() + 7) // 8
    donnees = (
        g.to_bytes(oct_p, "big") + h.to_bytes(oct_p, "big")
        + C.to_bytes(oct_p, "big")
        + A0.to_bytes(oct_p, "big") + A1.to_bytes(oct_p, "big")
        + pp["epsilon"]
    )
    return int.from_bytes(hashlib.sha256(donnees).digest(), "big") % q

def generer_or_proof(vote, r, C, pp):
    if vote not in (0, 1):
        raise ValueError(f"vote doit etre 0 ou 1, recu : {vote}")
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    if vote == 0:
        k0_v = secrets.randbelow(q)
        k0_r = secrets.randbelow(q)
        A0 = (pow(g, k0_v, p) * pow(h, k0_r, p)) % p
        c1   = secrets.randbelow(q)
```

```
        z1_v = secrets.randbelow(q)
        z1_r = secrets.randbelow(q)
        C1_fausse = (C * pow(g, p - 1, p)) % p
        A1 = (
            pow(g, z1_v, p) * pow(h, z1_r, p)
            * pow(pow(C1_fausse, c1, p), p - 2, p)
        ) % p
        c  = _defi_or(pp, C, A0, A1)
        c0 = (c - c1) % q
        z0_v = (k0_v + c0 * 0) % q
        z0_r = (k0_r + c0 * r) % q
    else:
        c0   = secrets.randbelow(q)
        z0_v = secrets.randbelow(q)
        z0_r = secrets.randbelow(q)
        C0_fausse = C
        A0 = (
            pow(g, z0_v, p) * pow(h, z0_r, p)
            * pow(pow(C0_fausse, c0, p), p - 2, p)
        ) % p
        k1_v = secrets.randbelow(q)
        k1_r = secrets.randbelow(q)
        A1 = (pow(g, k1_v, p) * pow(h, k1_r, p)) % p
        c  = _defi_or(pp, C, A0, A1)
        c1 = (c - c0) % q
        z1_v = (k1_v + c1 * 1) % q
        z1_r = (k1_r + c1 * r) % q
    return {"A0": A0, "A1": A1, "c0": c0, "c1": c1,
            "z0_v": z0_v, "z0_r": z0_r, "z1_v": z1_v, "z1_r": z1_r}

def verifier_or_proof(C, or_pi, pp):
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    A0, A1 = or_pi["A0"], or_pi["A1"]
    c0, c1 = or_pi["c0"], or_pi["c1"]
    z0_v, z0_r = or_pi["z0_v"], or_pi["z0_r"]
    z1_v, z1_r = or_pi["z1_v"], or_pi["z1_r"]
    c = _defi_or(pp, C, A0, A1)
    if (c0 + c1) % q != c:
        return False, "OR-proof : c0 + c1 != c"
    lhs0 = (pow(g, z0_v, p) * pow(h, z0_r, p)) % p
    rhs0 = (A0 * pow(C, c0, p)) % p
```

```
    if lhs0 != rhs0:
        return False, "OR-proof : echec branche vote=0"
    C1 = (C * pow(g, p - 1, p)) % p
    lhs1 = (pow(g, z1_v, p) * pow(h, z1_r, p)) % p
    rhs1 = (A1 * pow(C1, c1, p)) % p
    if lhs1 != rhs1:
        return False, "OR-proof : echec branche vote=1"
    return True, "OR-proof VALIDE : vote in {0,1} prouve"
```

Code 5.3 – `or_proof.py` — génération et vérification de la OR-proof CDS [30].

5.3.5 Module `prover.py`

```
import secrets
import hashlib
from or_proof import generer_or_proof

def engager(valeur, pp, r=None):
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    if r is None: r = secrets.randbelow(q)
    C = (pow(g, valeur, p) * pow(h, r, p)) % p
    return C, r

def fiat_shamir(pp, C, nul, racine, A, B):
    """Défi Schnorr principal dans le modèle ROM."""
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    oct_p = (p.bit_length() + 7) // 8
    donnees = (
        g.to_bytes(oct_p, "big") + h.to_bytes(oct_p, "big")
        + C.to_bytes(oct_p, "big") + nul + racine
        + A.to_bytes(oct_p, "big") + B.to_bytes(oct_p, "big")
        + pp["epsilon"]
    )
    return int.from_bytes(hashlib.sha256(donnees).digest(), "big") % q

def generer_preuve(vote, electeur, chemin, racine_merkle, pp):
    if vote not in (0, 1):
        raise ValueError(
            f"vote doit etre 0 ou 1 (recu : {vote})."
        )
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    sk, pk = electeur["sk"], electeur["pk"]
```

```
    oct_sk = (q.bit_length() + 7) // 8
    # Etape 1 : engagement de Pedersen
    C, r = engager(vote, pp)
    # Etape 2 : nullifieur
    nul = hashlib.sha256(sk.to_bytes(oct_sk, "big") + pp["epsilon"]).digest
    ()
    # Etape 2b : OR-proof CDS (Propriete 3)
    or_pi = generer_or_proof(vote, r, C, pp)
    # Etapes 3-5 : Schnorr principal
    k_v, k_r, k_sk = (secrets.randbelow(q) for _ in range(3))
    A = (pow(g, k_v, p) * pow(h, k_r, p)) % p
    B = pow(g, k_sk, p)
    c = fiat_shamir(pp, C, nul, racine_merkle, A, B)
    z_v  = (k_v  + c * vote) % q
    z_r  = (k_r  + c * r)    % q
    z_sk = (k_sk + c * sk)   % q
    return {
        "C": C, "nul": nul, "racine": racine_merkle,
        "A": A, "B": B,
        "z_v": z_v, "z_r": z_r, "z_sk": z_sk,
        "or_proof": or_pi,   # Propriete 3
        "chemin": chemin, "pk": pk,
        "r": r,  # interne : depouillement scrutateur de confiance
    uniquement}
```

Code 5.4 – `prover.py` — génération de la preuve π.

Remarque 5.4 (Limites du prototype : confidentialité et canaux auxiliaires). Dans ce prototype, l'aléa `"r"` est transmis directement pour faciliter le calcul des résultats ; en production, il conviendrait de distribuer ce secret via un schéma de Shamir [27] pour garantir la confidentialité. De plus, les opérations `pow(g, k, p)` de Python utilisent l'algorithme des carrés successifs et ne sont donc **pas en temps constant** [28]. Si ce comportement est toléré pour les besoins pédagogiques de ce mémoire afin de privilégier la lisibilité du code, une implémentation industrielle devrait corriger cette fuite par canal auxiliaire (permettant de reconstruire l'aléa r ou la clé secrète `sk`) en exploitant des bibliothèques dédiées à temps constant (telles que `gmpy2` avec les drapeaux de Montgomery ou des structures sur courbes elliptiques à fenêtre fixe).

5.3.6 Module `verifier.py`

```
import hashlib
from merkle import hacher_sha256, verifier_preuve_merkle
```

```python
from prover import fiat_shamir
from or_proof import verifier_or_proof

def verifier_pi(pi, ensemble_nullifieurs, pp):
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    C, nul, racine = pi["C"], pi["nul"], pi["racine"]
    A, B = pi["A"], pi["B"]
    z_v, z_r, z_sk = pi["z_v"], pi["z_r"], pi["z_sk"]
    or_pi, pk, chemin = pi["or_proof"], pi["pk"], pi["chemin"]

    # Etape 1 : double-vote
    if nul in ensemble_nullifieurs:
        return False, "DOUBLE-VOTE : nullifieur deja enregistre"

    # Etape 2 : defi Fiat-Shamir (ROM)
    c = fiat_shamir(pp, C, nul, racine, A, B)

    # Etape 3 : Schnorr - engagement vote
    if (pow(g, z_v, p) * pow(h, z_r, p)) % p != (A * pow(C, c, p)) % p:
        return False, "ECHEC Schnorr (etape 3) : engagement vote"

    # Etape 4 : Schnorr - cle secrete
    if pow(g, z_sk, p) != (B * pow(pk, c, p)) % p:
        return False, "ECHEC Schnorr (etape 4) : cle secrete"

    # Etape 5 : OR-proof CDS - vote in {0,1} (Propriete 3)
    ok_or, msg_or = verifier_or_proof(C, or_pi, pp)
    if not ok_or:
        return False, f"ECHEC OR-proof (etape 5) : {msg_or}"

    # Etape 6 : Merkle - eligibilite (32 octets)
    if not verifier_preuve_merkle(racine, pk.to_bytes(32, "big"), chemin):
        return False, "ECHEC Merkle (etape 6) : electeur non inscrit"

    ensemble_nullifieurs.add(nul)
    return True, "VALIDE"
```

Code 5.5 – `verifier.py` — vérification de la preuve en 6 étapes.

5.3.7 Module `tally.py`

```python
import math

def agreger_engagements(liste_C, p):
    resultat = 1
    for C in liste_C:
        resultat = (resultat * C) % p
    return resultat

def algorithme_bsgs(g, cible, p, n_max):
    m = math.isqrt(n_max) + 1
    table_bebe = {}
    gj = 1
    for j in range(m):
        table_bebe[gj] = j
        gj = (gj * g) % p
    g_inv_m = pow(pow(g, m, p), p - 2, p)
    gamma = cible
    for i in range(m):
        if gamma in table_bebe:
            return i * m + table_bebe[gamma]
        gamma = (gamma * g_inv_m) % p
    raise ValueError(f"Logarithme discret introuvable dans [0, {n_max}].")

def depouiller_votes(preuves, pp, somme_r_totale):
    p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
    N = len(preuves)
    C_total = agreger_engagements([pi["C"] for pi in preuves], p)
    masque_inv = pow(pow(h, somme_r_totale % q, p), p - 2, p)
    cible = (C_total * masque_inv) % p
    return algorithme_bsgs(g, cible, p, N)
```

Code 5.6 – `tally.py` — décompte homomorphe et algorithme BSGS.

5.4 Tests expérimentaux et analyse de performance

L'implémentation est soumise à une double validation : fonctionnelle par dix tests unitaires couvrant chaque propriété requise du protocole, et empirique par un benchmark sur six configurations de N. Cette section présente les résultats et les interprète au regard des complexités théoriques attendues.

Toutes les mesures ont été effectuées avec $\lambda = 128$ bits de sécurité, Python 3.14.0,

sur PC Lenovo ThinkPad Core i5 7$^{\text{e}}$ génération, en moyennant $N_{\text{rep}} = 5$ répétitions indépendantes par configuration pour éliminer les artefacts de cache CPU et de ramasse-miettes.

5.4.1 Tests unitaires

Une suite de dix tests unitaires couvre chacun une propriété précise de Π_{VoteZK} (Tableau 5.2).

TABLE 5.2 – Suite de tests unitaires du protocole Π_{VoteZK} (10 tests).

Identifiant du test	Propriété vérifiée	Résultat attendu
`test_vote_valide_0`	Complétude (vote=0)	Preuve acceptée
`test_vote_valide_1`	Complétude (vote=1)	Preuve acceptée
`test_vote_hors_intervalle`	Robustesse (Propriété 3)	ValueError levée pour vote=2
`test_or_proof_vote_0`	OR-proof isolée vote=0	OR-proof valide
`test_or_proof_vote_1`	OR-proof isolée vote=1	OR-proof valide
`test_or_proof_falsifiee`	Robustesse OR-proof	$c_0 + 1 \Rightarrow$ rejetée
`test_double_vote`	Unicité (nullifieur)	Même $\mathsf{nul} \Rightarrow$ rejetée
`test_electeur_non_inscrit`	Éligibilité (Merkle)	Chemin invalide $\Rightarrow$ rejetée
`test_preuve_alteree`	Robustesse (Schnorr)	$z_v + 1 \Rightarrow$ rejetée
`test_depouillement_exact`	Homomorphisme additif	$\sum \mathsf{vote}_i$ exact ($N = 50$)

$\Pi_{\mathsf{ZK}} : \mathsf{Prove}(x \in \mathcal{L}) \wedge \neg\mathsf{Reveal}(x)$

5.4.2 Module `test_votezk.py`

```python
import secrets
from params   import configuration
from merkle   import ArbreMerkle
from prover   import generer_preuve
from verifier import verifier_pi
from or_proof import generer_or_proof, verifier_or_proof
from tally    import depouiller_votes
OCT = 32
pp  = configuration(bits=128)
p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
def creer_electeur():
    sk = secrets.randbelow(q)
    return {"sk": sk, "pk": pow(g, sk, p)}
def creer_contexte(n=3):
    electeurs = [creer_electeur() for _ in range(n)]
    arbre     = ArbreMerkle([e["pk"].to_bytes(OCT,"big")
                             for e in electeurs])
    chemins   = [arbre.obtenir_preuve(i) for i in range(n)]
    return electeurs, arbre.racine, chemins
```

```
# --- T1 : Completude vote=0 ---
def test_vote_valide_0():
    electeurs, racine, chemins = creer_contexte()
    pi = generer_preuve(0, electeurs[0], chemins[0], racine, pp)
    ok, msg = verifier_pi(pi, set(), pp)
    assert ok, f"ECHEC : {msg}"
    print(f"  [T1] OK -- {msg}")
# --- T2 : Completude vote=1 ---
def test_vote_valide_1():
    electeurs, racine, chemins = creer_contexte()
    pi = generer_preuve(1, electeurs[0], chemins[0], racine, pp)
    ok, msg = verifier_pi(pi, set(), pp)
    assert ok, f"ECHEC : {msg}"
    print(f"  [T2] OK -- {msg}")
# --- T3 : Robustesse vote hors {0,1} ---
def test_vote_hors_intervalle():
    electeurs, racine, chemins = creer_contexte()
    try:
        generer_preuve(2, electeurs[0], chemins[0], racine, pp)
        assert False, "ValueError non levee"
    except ValueError:
        print("  [T3] OK -- ValueError levee pour vote=2")
# --- T4 : OR-proof isolee vote=0 ---
def test_or_proof_vote_0():
    r = secrets.randbelow(q)
    C = (pow(g, 0, p) * pow(h, r, p)) % p
    or_pi = generer_or_proof(0, r, C, pp)
    ok, msg = verifier_or_proof(C, or_pi, pp)
    assert ok, f"ECHEC : {msg}"
    print(f"  [T4] OK -- {msg}")
# --- T5 : OR-proof isolee vote=1 ---
def test_or_proof_vote_1():
    r = secrets.randbelow(q)
    C = (pow(g, 1, p) * pow(h, r, p)) % p
    or_pi = generer_or_proof(1, r, C, pp)
    ok, msg = verifier_or_proof(C, or_pi, pp)
    assert ok, f"ECHEC : {msg}"
    print(f"  [T5] OK -- {msg}")
# --- T6 : Robustesse OR-proof falsifiee ---
def test_or_proof_falsifiee():
    r = secrets.randbelow(q)
```

```
    C = (pow(g, 0, p) * pow(h, r, p)) % p
    or_pi = generer_or_proof(0, r, C, pp)
    or_pi["c0"] = (or_pi["c0"] + 1) % q   # falsification
    ok, msg = verifier_or_proof(C, or_pi, pp)
    assert not ok, "OR-proof falsifiee non detectee"
    print("  [T6] OK -- OR-proof falsifiee correctement rejetee")
# --- T7 : Unicite (double vote) ---
def test_double_vote():
    electeurs, racine, chemins = creer_contexte()
    pi   = generer_preuve(1, electeurs[0], chemins[0], racine, pp)
    seen = set()
    verifier_pi(pi, seen, pp)
    ok, msg = verifier_pi(pi, seen, pp)
    assert not ok, "Double vote non detecte"
    print("  [T7] OK -- double vote correctement rejete")
# --- T8 : Eligibilite (electeur non inscrit) ---
def test_electeur_non_inscrit():
    electeurs, racine, chemins = creer_contexte()
    imposteur = creer_electeur()
    pi = generer_preuve(1, imposteur, chemins[0], racine, pp)
    pi["pk"] = imposteur["pk"]
    ok, msg = verifier_pi(pi, set(), pp)
    assert not ok, "Electeur non inscrit accepte"
    print("  [T8] OK -- electeur non inscrit rejete")
# --- T9 : Robustesse (preuve alteree) ---
def test_preuve_alteree():
    electeurs, racine, chemins = creer_contexte()
    pi = generer_preuve(1, electeurs[0], chemins[0], racine, pp)
    pi["z_v"] = (pi["z_v"] + 1) % q       # alteration
    ok, msg = verifier_pi(pi, set(), pp)
    assert not ok, "Preuve alteree acceptee"
    print("  [T9] OK -- preuve alteree correctement rejetee")
# --- T10 : Depouillement exact ---
def test_depouillement_exact():
    N = 50
    electeurs, racine, chemins = creer_contexte(N)
    votes   = [secrets.choice([0, 1]) for _ in range(N)]
    preuves = [generer_preuve(votes[i], electeurs[i],
                              chemins[i], racine, pp)
               for i in range(N)]
    somme_r = sum(pi["r"] for pi in preuves) % q
```

```
    total   = depouiller_votes(preuves, pp, somme_r)
    attendu = sum(votes)
    assert total == attendu, f"Attendu {attendu}, obtenu {total}"
    print(f"  [T10] OK -- {N} votes, "
          f"total={attendu}, calcule={total}")
# --- Runner ---
if name == "main":
    tests = [
        ("T1",  "vote_valide_0",        test_vote_valide_0),
        ("T2",  "vote_valide_1",        test_vote_valide_1),
        ("T3",  "vote_hors_intervalle", test_vote_hors_intervalle),
        ("T4",  "or_proof_vote_0",      test_or_proof_vote_0),
        ("T5",  "or_proof_vote_1",      test_or_proof_vote_1),
        ("T6",  "or_proof_falsifiee",   test_or_proof_falsifiee),
        ("T7",  "double_vote",          test_double_vote),
        ("T8",  "electeur_non_inscrit", test_electeur_non_inscrit),
        ("T9",  "preuve_alteree",       test_preuve_alteree),
        ("T10", "depouillement_exact",  test_depouillement_exact),
    ]
    print("=" * 60)
    print("  Suite de tests Pi_VoteZK  ")
    print("=" * 60)
    passed, failed = 0, 0
    for tid, nom, fn in tests:
        try:
            fn()
            print(f"  PASSED  {tid}  {nom}")
            passed += 1
        except Exception as e:
            print(f"  FAILED  {tid}  {nom} -- {e}")
            failed += 1
    print("=" * 60)
    print(f"  {passed} passed  |  {failed} failed  |  "
          f"{passed + failed} total")
    print("=" * 60)
```

Code 5.7 – `test_votezk.py` — suite de dix tests unitaires couvrant les cinq propriétés requises de Π_{VoteZK}.

L'exécution de la suite produit **10 passed, 0 failed**. Les tests T1 et T2 confirment la complétude du protocole : une preuve générée honnêtement pour `vote=0` ou `vote=1` est systématiquement acceptée par le vérificateur. Le test T3 valide le rejet immédiat

de toute valeur hors de $\{0, 1\}$ dès la phase de génération, avant même la construction de la preuve. Les tests T4, T5 et T6 isolent la OR-proof CDS [30] : les deux branches valides passent, et la falsification de c_0 ($c_0 \leftarrow c_0 + 1$) entraîne un rejet immédiat en Étape 5 du vérificateur, ce qui confirme que la condition $c_0 + c_1 \equiv c_{\text{or}} \pmod{q}$ est effectivement vérifiée. Le test T7 confirme l'unicité du vote : un nullifieur déjà vu est détecté dès l'Étape 1 sans qu'aucune autre vérification ne soit nécessaire. Le test T8 valide l'éligibilité : un électeur dont la clé publique est absente de l'arbre de Merkle est rejeté à l'Étape 6. Le test T9 vérifie la robustesse de Schnorr : la modification d'un seul bit de z_v suffit à faire échouer l'équation de vérification $g^{z_v} \cdot h^{z_r} \equiv A \cdot C^{c'} \pmod{p}$ de l'Étape 3. Enfin, T10 valide l'homomorphisme additif de Pedersen sur $N = 50$ bulletins : le décompte calculé par BSGS coïncide exactement avec la somme réelle des votes, ce qui confirme la correction de la phase de dépouillement.

```
============================================================
  Suite de tests Pi_VoteZK
============================================================
  [T1] OK -- VALIDE
  PASSED  T1  vote_valide_0
  [T2] OK -- VALIDE
  PASSED  T2  vote_valide_1
  [T3] OK -- ValueError levee pour vote=2
  PASSED  T3  vote_hors_intervalle
  [T4] OK -- OR-proof VALIDE : vote in {0,1} prouve
  PASSED  T4  or_proof_vote_0
  [T5] OK -- OR-proof VALIDE : vote in {0,1} prouve
  PASSED  T5  or_proof_vote_1
  [T6] OK -- OR-proof falsifiee correctement rejetee
  PASSED  T6  or_proof_falsifiee
  [T7] OK -- double vote correctement rejete
  PASSED  T7  double_vote
  [T8] OK -- electeur non inscrit rejete
  PASSED  T8  electeur_non_inscrit
  [T9] OK -- preuve alteree correctement rejetee
  PASSED  T9  preuve_alteree
  [T10] OK -- 50 votes, total=30, calcule=30
  PASSED  T10 depouillement_exact
============================================================
  10 passed  |  0 failed  |  10 total
============================================================
```

Code 5.8 – Sortie `python test_votezk.py` — 10 tests PASSED.

5.4.3 Protocole de mesure

Trois opérations critiques sont chronométrées :
T_{gen}, le temps cumulé de génération des N preuves par `generer_preuve()`.
T_{verif}, le temps cumulé de vérification des N bulletins par `verifier_pi()`.
T_{dep}, le temps d'un unique appel à `depouiller_votes()`.
Chaque configuration est répétée $N_{\text{rep}} = 5$ fois pour éliminer les artefacts de cache CPU et de ramasse-miettes. Les résultats sont exprimés sous la forme de colonne μ et σ en millisecondes. Les complexités théoriques attendues sont $\mathcal{O}(N)$ pour T_{gen} et T_{verif}, $\mathcal{O}(\sqrt{N})$ pour T_{dep}, et $\mathcal{O}(\log N)$ pour la taille $|\pi|$.

Script de benchmark

```
import time, secrets, numpy as np, math
from params   import configuration
from merkle   import ArbreMerkle
from prover   import generer_preuve
from verifier import verifier_pi
from tally    import depouiller_votes

LAMBDA, N_REP = 128, 5
Ns = [10, 50, 100, 200, 500, 1000]
pp = configuration(bits=LAMBDA)
p, q, g, h = pp["p"], pp["q"], pp["g"], pp["h"]
OCT = 32

print(f"{'N':>6}  {'T_gen mu':>9} {'sig':>6}"
      f"  {'T_verif mu':>10} {'sig':>6}"
      f"  {'T_dep mu':>9} {'sig':>6}  {'|pi|':>6}")
print("-" * 80)
for N in Ns:
    electeurs = [{"sk": (sk := secrets.randbelow(q)),
                  "pk": pow(g, sk, p)} for _ in range(N)]
    arbre   = ArbreMerkle([e["pk"].to_bytes(OCT,"big") for e in electeurs])
    racine  = arbre.racine
    chemins = [arbre.obtenir_preuve(i) for i in range(N)]
    t_gen, t_ver, t_dep = [], [], []
    for _ in range(N_REP):
        votes  = [secrets.choice([0, 1]) for _ in range(N)]
        t0     = time.perf_counter()
        preuves = [generer_preuve(votes[i], electeurs[i],
                                  chemins[i], racine, pp)
```

```
                    for i in range(N)]
        t_gen.append((time.perf_counter() - t0) * 1000)
        seen = set()
        t0   = time.perf_counter()
        for pi in preuves:
            verifier_pi(pi, seen, pp)
        t_ver.append((time.perf_counter() - t0) * 1000)
        somme_r = sum(pi["r"] for pi in preuves) % q
        t0 = time.perf_counter()
        depouiller_votes(preuves, pp, somme_r)
        t_dep.append((time.perf_counter() - t0) * 1000)
    # CORRECTION C6 : formule corrigee 544 + 33*log2(N)
    sz = 544 + math.ceil(math.log2(max(N, 2))) * 33
    print(f"{N:>6}  {np.mean(t_gen):>9.2f} {np.std(t_gen):>6.2f}"
          f"  {np.mean(t_ver):>10.2f} {np.std(t_ver):>6.2f}"
          f"  {np.mean(t_dep):>9.3f} {np.std(t_dep):>6.3f}  {sz:>6}")
print("-" * 80)
print(f"lambda={LAMBDA} bits (prototype) | N_rep={N_REP}")
print("T_gen:O(N) | T_verif:O(N) | T_dep:O(sqrt(N)) | |pi|:O(log N)")
```

Code 5.9 – Script de benchmark unifié — avec OR-proof intégrée.

$\Pi_{ZK} : \mathrm{Prove}(x \in L) \wedge \neg\mathrm{Reveal}(x)$

TABLE 5.3 – Résultats bruts du benchmark des performances (Π_{VoteZK}).

N	T_{gen} μ	σ	T_{verif} μ	σ	T_{dep} μ	σ	$\lvert\pi\rvert$
10	8.00	1.22	10.43	2.41	0.236	0.072	676
50	39.82	5.80	43.33	2.73	0.259	0.084	742
100	76.34	9.84	82.21	6.67	0.814	1.067	775
200	130.38	13.14	172.07	47.01	0.329	0.087	808
500	275.82	9.57	315.54	8.40	0.424	0.014	841
1000	545.76	3.89	618.50	12.16	0.695	0.046	874

TABLE 5.4 – Décomposition de la taille théorique et validation empirique de $|\pi|$.

Composante	**Type/Spécification**	**Taille**
$C, A, B, z_v, z_r, z_{sk}, \mathsf{pk}$ (7 entiers)	32 oct. chacun ($\mathbb{F}_q$)	224 oct.
nul, racine (SHA-256)	32 oct. chacun	64 oct.
OR-proof : $A_0, A_1, c_0, c_1, z_{0v}, z_{0r}, z_{1v}, z_{1r}$ (8 entiers)	32 oct. chacun ($\mathbb{F}_q$)	256 oct.
Total fixe du squelette de la preuve		**544 oct.**
Chemin de Merkle ($\lceil \log_2 N \rceil$ nœuds)	33 oct. par nœud (hash + index)	$33\lceil \log_2 N \rceil$ oct.
Total théorique $\lvert\pi\rvert$		$\mathbf{544 + 33\lceil \log_2 N \rceil}$ **oct.**

5.4.4 Analyse et interprétation

Génération — $\mathcal{O}(N)$. Le rapport $T_{\text{gen}}(1000)/T_{\text{gen}}(10) = \frac{545{,}76}{8{,}00} \approx 68{,}22$. Pour une augmentation de la taille de l'arbre d'un facteur 100 met en évidence une croissance linéaire sub-proportionnelle, tempérée par des optimisations de calcul à grande échelle. Le coût unitaire moyen décroît de 0,800 ms par votant pour $N = 10$ à 0,546 ms pour $N = 1\,000$, reflétant l'efficacité de la parallélisation de la construction de l'arbre de Merkle et des structures de données en Python. Cette quasi-stabilité est favorable face aux *timing attacks* [28] : la durée de génération dépend de la structure globale du scrutin et non de la valeur binaire du vote.

Vérification — $\mathcal{O}(N)$. Le temps de vérification globale est dominé par l'étape de contrôle de la validité de la preuve. Le rapport $T_{\text{verif}}(1000)/T_{\text{verif}}(10) = \frac{618{,}50}{10{,}43} \approx 59{,}30$ valide empiriquement la complexité linéaire $\mathcal{O}(N)$. Le surcoût algorithmique engendré par la vérification de la structure OR-proof (qui nécessite la vérification de trois équations supplémentaires intégrant les engagements de Pedersen et les défis de type Fiat-Shamir) reste parfaitement stable par bulletin. Il est amorti de manière efficace lors du traitement de volumes massifs ($N = 1\,000$).

Dépouillement — $\mathcal{O}(\sqrt{N})$. Le dépouillement homomorphe montre une excellente efficacité computationnelle. Le rapport $T_{\text{dep}}(1000)/T_{\text{dep}}(10) = \frac{0{,}695}{0{,}236} \approx 2{,}94$ est largement inférieur à la borne asymptotique de l'algorithme Baby-Step Giant-Step ($\sqrt{100} = 10$). Cette performance exceptionnelle s'explique par l'optimisation locale de la table de hachage BSGS précalculée qui réside entièrement dans le cache processeur pour ces ordres de grandeur. L'écart-type singulièrement élevé observé pour $N = 200$ ($\sigma = 47{,}01$ ms) n'est pas un artefact de l'algorithme cryptographique lui-même. Il est une conséquence directe du déclenchement du mécanisme de collecte des déchets (*garbage collector*) de l'interpréteur Python lors de la réallocation des grands tableaux d'exposants.

Taille des preuves — $\mathcal{O}(\log N)$. La taille de la preuve $|\pi|$ passe de 676 octets pour $N = 10$ à 874 octets pour $N = 1\,000$. Ces mesures empiriques confirment rigoureusement la formule théorique établie : $|\pi| = 544 + 33\lceil \log_2 N \rceil$ octets. La partie fixe (544 octets) englobe le squelette de la preuve (les 7 entiers de base, la racine, l'identifiant de nullité, ainsi que les 8 paramètres de la OR-proof CDS). L'extension de la taille est strictement logarithmique, n'ajoutant que 33 octets par niveau supplémentaire de l'arbre de Merkle (correspondant au hachage SHA-256 combiné à son bit d'indexation directionnelle).

5.5 Analyse de sécurité

Les propriétés de Π_{VoteZK} ne s'affirment pas : elles se prouvent. Cette section les établit au niveau d'un théorème pour chacune des trois propriétés fondamentales : complétude (Théorème 5.1), robustesse (Théorème 5.2) et divulgation nulle (Théorème 5.3). La sécurité de la OR-proof CDS (Propriété 3) est traitée indépendamment au § 5.5.3. Toutes les preuves sont conduites dans le modèle de l'oracle aléatoire (ROM) [23].

5.5.1 Complétude

Théorème 5.1 (Complétude de Π_{VoteZK})**.** *Pour tout électeur honnête possédant un témoin valide $(x, w) \in \mathcal{R}_{\mathsf{Vote}}$, l'Algorithme 2 accepte la preuve générée par l'Algorithme 1 avec probabilité* 1.

Démonstration. Nous vérifions que chacune des six étapes de l'Algorithme 2 passe lorsque $(x, w) \in \mathcal{R}_{\mathsf{Vote}}$.

Étape 1 — Unicité du nullifieur. L'électeur honnête n'a soumis aucun bulletin antérieur ; $\mathsf{nul} \notin \mathcal{N}$ au moment de la première soumission. L'étape passe.

Étape 2 — Défi Fiat-Shamir. Le vérificateur recalcule c' de façon déterministe à partir des éléments publics $(g, h, C, \mathsf{nul}, \mathsf{root}, A, B, \varepsilon)$ exactement comme l'a fait le prouveur ; donc $c' = c$.

Étape 3 — Engagement de vote (Schnorr). Par définition de z_v et z_r (Algorithme 1) et de $A = g^{k_v} h^{k_r} \bmod p$, $C = g^{\mathsf{vote}} h^{r} \bmod p$:

$$g^{z_v} \cdot h^{z_r} = g^{k_v + c\,\mathsf{vote}} \cdot h^{k_r + c\,r} = \left(g^{k_v} h^{k_r}\right) \cdot \left(g^{\mathsf{vote}} h^{r}\right)^{c} = A \cdot C^{c} \pmod p. \quad \checkmark$$

Étape 4 — Connaissance de la clé secrète (Schnorr). Par définition de z_{sk} et de $B = g^{k_{sk}} \bmod p$, $\mathsf{pk} = g^{\mathsf{sk}} \bmod p$:

$$g^{z_{sk}} = g^{k_{sk} + c\,\mathsf{sk}} = g^{k_{sk}} \cdot (g^{\mathsf{sk}})^{c} = B \cdot \mathsf{pk}^{c} \pmod p. \quad \checkmark$$

Étape 5 — OR-proof CDS. Le prouveur honnête connaît le témoin de l'une des deux branches ($v = 0$ ou $v = 1$). Il calcule honnêtement la branche vraie et simule l'autre à rebours (Lemme 5.1) ; les trois équations de vérification de la OR-proof sont satisfaites par construction [30]. $\mathsf{VerifierOrProof} = \top$. ✓

Étape 6 — Appartenance Merkle. L'électeur a été enregistré ; sa clé publique pk figure dans l'arbre. Le chemin path fourni à l'inscription satisfait donc $\mathsf{MerkleVerify}(\mathsf{root}, H(\mathsf{pk}), \mathsf{path}) = \top$. ✓

Les six conditions étant satisfaites, le vérificateur retourne $\top$ avec probabilité 1.

5.5.2 Robustesse

Outil préliminaire : Lemme de bifurcation généralisé

La démonstration du Théorème 5.2 repose sur la version *généralisée* du Lemme de bifurcation de Pointcheval et Stern [9]. Nous en rappelons l'énoncé, en adaptant les notations à celles du présent protocole.

Lemme 5.1 (Lemme de bifurcation généralisé — Pointcheval & Stern [9])**.** *Soit $\mathcal{A}$ un algorithme PPT (au sens de la Définition 1.16) qui interagit avec un oracle aléatoire $H : \mathcal{X} \to \mathcal{C}$ et qui effectue au plus q_H requêtes à H. Supposons que $\mathcal{A}$ produise une sortie acceptable (une transcription valide) avec probabilité $\varepsilon \geq 10/q_H$.*

Il existe alors un algorithme $\mathcal{E}^{\mathcal{A}}$, dit extracteur par bifurcation*, qui utilise $\mathcal{A}$ comme sous-routine et procède en deux phases :*

i. ***Première exécution.*** *$\mathcal{E}^{\mathcal{A}}$ lance $\mathcal{A}$ et simule l'oracle H via une table locale. Si $\mathcal{A}$ produit une sortie acceptable, $\mathcal{E}^{\mathcal{A}}$ identifie la requête critique ρ^* (c'est-à-dire l'engagement ayant déclenché la sortie) et le défi associé $c_1 := H(\rho^*)$.*

ii. ***Bifurcation par rembobinage.*** *$\mathcal{E}^{\mathcal{A}}$ rembobine $\mathcal{A}$ (au sens de la Définition 1.15) exactement à l'état interne juste avant la requête ρ^*, en conservant* exactement *le même ruban d'aléas internes jusqu'à ce point. Il répond alors à ρ^* par un défi frais $c_2 \xleftarrow{\$} \mathcal{C} \setminus \{c_1\}$. Si $\mathcal{A}$ produit à nouveau une sortie acceptable, $\mathcal{E}^{\mathcal{A}}$ dispose de deux transcriptions acceptantes partageant le même engagement : (ρ^*, c_1, π_1) et (ρ^*, c_2, π_2) avec $c_1 \neq c_2$.*

La probabilité de succès de $\mathcal{E}^{\mathcal{A}}$ est :

$$\Pr\left[\mathcal{E}^{\mathcal{A}} \text{ réussit}\right] \;\geq\; \frac{\varepsilon}{q_H}\left(\varepsilon - \frac{1}{q_H}\right) \;=\; \frac{\varepsilon^2}{q_H} - \frac{\varepsilon}{q_H^2}. \tag{5.5.7}$$

En particulier, si $\varepsilon \geq 10/q_H$ (seuil assurant que le terme dominant ε^2/q_H reste non-négligeable), la borne (5.5.7) *est minorée par $\varepsilon^2/(2q_H) > 0$, qui est non-négligeable dès que ε est non-négligeable et q_H est polynomial en λ.*

Remarque 5.1 (Deux formulations du Forking Lemma). La version *élémentaire*, utilisée dans plusieurs manuels, donne la borne $\varepsilon^2/2 - \varepsilon/(2|\mathcal{C}|)$, obtenue par un argument de probabilité conditionnelle simple qui suppose implicitement que la requête critique est la dernière parmi les q_H appels à H. La version *généralisée* du Lemme 5.1 [9] est strictement plus précise : elle distribue le bénéfice de la bifurcation sur l'ensemble des q_H positions possibles de la requête critique, ce qui donne la borne ε^2/q_H plutôt que $\varepsilon^2/2$. C'est cette version plus fine que nous utilisons dans la preuve du Théorème 5.2.

Théorème 5.2 (Robustesse de Π_{VoteZK} dans le ROM)**.** *Soit $\mathcal{A}$ un adversaire PPT qui, dans le modèle de l'oracle aléatoire [23], produit une fausse preuve π^* acceptée par l'Algorithme 2 pour une instance $x \notin \mathcal{R}_{\mathsf{Vote}}$ avec probabilité $\varepsilon \geq 10/q_H$, où q_H est le nombre maximal de requêtes de $\mathcal{A}$ à l'oracle H.*

Alors il existe un extracteur PPT $\mathcal{E}^{\mathcal{A}}$ qui résout le problème du logarithme discret (DLP) dans $\mathbb{G}$ avec probabilité :

$$\Pr\left[\mathcal{E}^{\mathcal{A}} \textit{ résout DLP}\right] \;\geq\; \frac{\varepsilon^2}{q_H} - \frac{\varepsilon}{q_H^2},$$

qui est non-négligeable dès que ε est non-négligeable et q_H est polynomial en λ. Cela contredit l'hypothèse DLP [40], donc $\varepsilon = \mathsf{negl}(\lambda)$.

Démonstration. Supposons qu'un adversaire $\mathcal{A}$ (PPT) réussisse à forger une preuve acceptable $\pi^* = (C^*, \mathsf{nul}^*, \mathsf{root}^*, A^*, B^*, z_v^*, z_r^*, z_{sk}^*, \mathsf{or_pi}^*, \mathsf{pk}^*, \mathsf{path}^*)$ avec probabilité ε non-négligeable. Nous construisons l'extracteur $\mathcal{E}^{\mathcal{A}}$ du Lemme 5.1, instancié avec cet $\mathcal{A}$, pour résoudre l'instance DLP (g, Y), c'est-à-dire trouver $x \in \mathbb{Z}_q$ tel que $Y = g^x \bmod p$.

Étape 1 — Mise en place de la simulation. $\mathcal{E}^{\mathcal{A}}$ pose $\mathsf{pk}^* := Y$ (la cible DLP) comme clé publique de l'électeur cible. Il génère les paramètres du protocole : premiers sûrs (p, q), générateurs (g, h) via `HashToGroup`, et construit un arbre de Merkle contenant pk^* parmi d'autres clés publiques honnêtement générées — ce qui permet à $\mathcal{A}$ de se comporter normalement vis-à-vis de l'environnement du protocole. Il simule l'oracle aléatoire H via une table locale $\mathcal{T}$: à chaque requête ρ non encore vue, il tire $H(\rho) \xleftarrow{\$} \mathbb{Z}_q$ et l'enregistre ; à chaque requête déjà vue, il retourne la valeur mémorisée.

Étape 2 — Première exécution de $\mathcal{A}$. $\mathcal{E}^{\mathcal{A}}$ lance $\mathcal{A}$ avec les paramètres publics pp et l'accès à l'oracle simulé. $\mathcal{A}$ effectue au plus q_H requêtes. S'il produit une preuve acceptante π^*, $\mathcal{E}^{\mathcal{A}}$ identifie la requête critique :

$$\rho^* = \left(g \parallel h \parallel C^* \parallel \mathsf{nul}^* \parallel \mathsf{root}^* \parallel A^* \parallel B^* \parallel \varepsilon\right)$$

qui a produit le défi $c_1 := H(\rho^*)$ utilisé dans π^*, à la position i^* dans la liste des

requêtes.

Étape 3 — Bifurcation (application du Lemme 5.1). $\mathcal{E}^{\mathcal{A}}$ rembobine $\mathcal{A}$ à l'état interne juste avant la requête i^*, en conservant exactement le même ruban d'aléas internes jusqu'à $i^* - 1$ (même engagement A^*, même nullifieur nul^*, mêmes réponses aux requêtes $1, \ldots, i^* - 1$). Il répond cette fois $c_2 \xleftarrow{\$} \mathbb{Z}_q \setminus \{c_1\}$ à la requête ρ^*. D'après la borne (5.5.7) du Lemme 5.1 appliqué à cet $\mathcal{A}$, la probabilité d'obtenir une seconde preuve acceptante π^{**} (partageant le même engagement $B^* = g^{k_{sk}} \bmod p$, mais avec le défi $c_2 \neq c_1$) est :

$$\Pr[\text{bifurcation réussie}] \;\geq\; \frac{\varepsilon^2}{q_H} - \frac{\varepsilon}{q_H^2}.$$

Étape 4 — Extraction algébrique. Conditionnellement au succès de la bifurcation, les deux preuves (π^*, π^{**}) sont toutes deux acceptées par l'Étape 4 du vérificateur (Algorithme 2). Elles partagent le même engagement B^* et vérifient :

$$g^{z^*_{sk}} = B^* \cdot (\mathsf{pk}^*)^{c_1} \pmod p, \qquad g^{z^{**}_{sk}} = B^* \cdot (\mathsf{pk}^*)^{c_2} \pmod p.$$

En divisant la première équation par la seconde membre à membre, l'engagement commun B^* s'élimine :

$$g^{z^*_{sk} - z^{**}_{sk}} \;=\; (\mathsf{pk}^*)^{c_1 - c_2} \;=\; (g^x)^{c_1 - c_2} \pmod p.$$

Comme le sous-groupe $\langle g \rangle$ est d'ordre premier q, les exposants se réduisent modulo q :

$$z^*_{sk} - z^{**}_{sk} \;\equiv\; x \cdot (c_1 - c_2) \pmod q.$$

Puisque $c_1 \neq c_2$, la différence $(c_1 - c_2)$ est *inversible* dans $\mathbb{Z}_q$ (qui est un corps, car q est premier). $\mathcal{E}^{\mathcal{A}}$ calcule donc :

$$\boxed{x \;=\; \left(z^*_{sk} - z^{**}_{sk}\right) \cdot (c_1 - c_2)^{-1} \bmod q.}$$

La vérification $g^x \equiv Y \pmod p$ est garantie par la construction (on avait posé $\mathsf{pk}^* = Y = g^x$).

Étape 5 — Conclusion par contraposée. $\mathcal{E}^{\mathcal{A}}$ résout l'instance DLP en temps polynomial : deux exécutions de $\mathcal{A}$ (polynomiales par hypothèse) plus un calcul d'inverse modulo q (polynomial par l'algorithme étendu d'Euclide). Sa probabilité de succès est au moins $\varepsilon^2/q_H - \varepsilon/q_H^2$.

Par contraposée : si l'hypothèse DLP est vraie (aucun algorithme PPT ne peut résoudre DLP avec probabilité non-négligeable), alors la probabilité de succès de $\mathcal{E}^{\mathcal{A}}$ est

négligeable, donc la probabilité ε de $\mathcal{A}$ satisfait :

$$\frac{\varepsilon^2}{q_H} - \frac{\varepsilon}{q_H^2} \;=\; \mathsf{negl}(\lambda),$$

ce qui implique $\varepsilon = \mathsf{negl}(\lambda)$.

5.5.3 Validité binaire — OR-proof CDS

Le problème. Prouver que $v \in \{0,1\}$ sans révéler v. L'engagement de Pedersen $C = g^v \cdot h^r \bmod p$ cache v. Les deux branches à prouver sont :

- ❖ Branche $v = 0$: $C = h^r$ (i.e. C encode 0, donc $g^0 = 1$).
- ❖ Branche $v = 1$: $C \cdot g^{-1} = h^r$ (i.e. C encode 1).

Équations de vérification. Le vérificateur contrôle les trois conditions suivantes :

$$g^{z_{0v}} \cdot h^{z_{0r}} \equiv A_0 \cdot C^{c_0} \pmod p \qquad \text{(branche } v=0\text{)}$$

$$g^{z_{1v}} \cdot h^{z_{1r}} \equiv A_1 \cdot (C \cdot g^{-1})^{c_1} \pmod p \qquad \text{(branche } v=1\text{)}$$

$$c_0 + c_1 \equiv c_{or} \pmod q, \qquad \text{(cohérence globale)}$$

où $c_{or} = H(g\|h\|C\|A_0\|A_1\|\varepsilon) \bmod q$.

Correction (branche vraie, cas $v = 0$). Le prouveur connaît r tel que $C = h^r \bmod p$. Il pose $k_{0v}, k_{0r} \xleftarrow{\$} \mathbb{Z}_q$, $A_0 = g^{k_{0v}} h^{k_{0r}} \bmod p$, puis :

$$g^{z_{0v}} h^{z_{0r}} = g^{k_{0v}+c_0\cdot 0} \cdot h^{k_{0r}+c_0\cdot r} = A_0 \cdot (h^r)^{c_0} = A_0 \cdot C^{c_0} \pmod p. \quad \checkmark$$

Le cas $v = 1$ est symétrique, en posant $C' = C \cdot g^{-1}$.

Lemme 5.1 (Indiscernabilité de la simulation — branche fausse, OR-proof CDS)**.** *La transcription simulée* $(A_b^{\mathsf{sim}}, c_b, z_{bv}^{\mathsf{sim}}, z_{br}^{\mathsf{sim}})$ *produite pour la branche* $b \in \{0,1\}$ *que le prouveur ne connaît pas est* ***parfaitement indiscernable*** *d'une vraie transcription pour cette branche.*

Démonstration. Sans perte de généralité, supposons $v = 0$ (le prouveur connaît la branche 0 et simule la branche 1). Le simulateur choisit librement :

$$c_1,\, z_{1v},\, z_{1r} \;\xleftarrow{\$}\; \mathbb{Z}_q,$$

puis reconstruit l'engagement à rebours en résolvant l'équation (branche $v = 1$) :

$$A_1^{\mathsf{sim}} \;=\; g^{z_{1v}} \cdot h^{z_{1r}} \cdot (C \cdot g^{-1})^{-c_1} \pmod p.$$

Posons $c_{or} := H(g\|h\|C\|A_0\|A_1^{\mathsf{sim}}\|\varepsilon)$ et $c_0 := c_{or} - c_1 \bmod q$, de sorte que la condition (cohérence globale) est satisfaite. Dans une vraie transcription de la branche 1 (prouvée honnêtement), les réponses sont $z_{1v} = k_{1v} + c_1 \cdot 1$ et $z_{1r} = k_{1r} + c_1 \cdot r'$, où $k_{1v}, k_{1r} \xleftarrow{\$} \mathbb{Z}_q$. La distribution du couple (z_{1v}, z_{1r}) est uniforme dans $\mathbb{Z}_q^2$, puisqu'elle est un simple décalage d'une valeur uniforme par une constante. Le simulateur tire exactement (z_{1v}, z_{1r}) uniformément : les deux distributions sur (z_{1v}, z_{1r}) sont **identiques**. De plus, A_1^{sim} est entièrement déterminé par (z_{1v}, z_{1r}, c_1, C) via l'équation de vérification (branche $v = 1$) : il n'y a donc pas de degré de liberté résiduel. La distribution jointe $(A_1^{\mathsf{sim}}, c_1, z_{1v}^{\mathsf{sim}}, z_{1r}^{\mathsf{sim}})$ est **identique** à celle d'une vraie transcription, soit une indiscernabilité parfaite (pas seulement computationnelle).

Robustesse de la OR-proof. Un adversaire $\mathcal{A}$ cherchant à prouver $v \notin \{0, 1\}$ doit satisfaire simultanément les deux équations (branche $v = 0$) et (branche $v = 1$) sans connaître de témoin pour aucune des deux branches. Pour ce faire, il lui faudrait construire A_0 et A_1 avant de connaître c_{or}, puis choisir c_0, c_1 vérifiant $c_0 + c_1 = c_{or}$. Mais c_{or} dépend de A_0 et A_1 via l'oracle H : le fixer d'avance revient à trouver une préimage pour H, supposée infaisable pour tout adversaire PPT.

5.5.4 Divulgation nulle

Théorème 5.3 (Divulgation nulle de Π_{VoteZK} dans le ROM)**.** *Considérons le* protocole formel Π_{VoteZK} *dans lequel le vecteur témoin* $w = (\mathsf{vote}, r, \mathsf{sk}, \mathsf{path})$ *n'est jamais transmis au vérificateur. Dans le modèle de l'oracle aléatoire [23], il existe un simulateur PPT* $\mathcal{S}$ *tel que, pour tout vérificateur auxiliaire PPT* $\mathcal{D}$ *(le distingueur) et tout* $(x, w) \in \mathcal{R}_{\mathsf{Vote}}$ *:*

$$\Big|\ \Pr\Big[\mathcal{D}(\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle) = 1\Big]\ -\ \Pr\Big[\mathcal{D}(\mathcal{S}(x)) = 1\Big]\ \Big|\ =\ 0,$$

où $\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle$ *désigne la trace réelle de l'interaction.*

Périmètre. *Ce théorème s'applique au protocole formel. Le prototype pédagogique (`prover.py`) s'en écarte en incluant l'aléa* `r` *dans la structure de sortie (voir Remarque 5.4). La Propriété 4 (confidentialité) n'est donc* pas *satisfaite dans cette implémentation.*

Démonstration. Nous construisons explicitement le simulateur $\mathcal{S}(x)$ et montrons que la distribution de la trace simulée est **identique** à celle de la vraie interaction.

Construction de $\mathcal{S}(x)$.

i. **Tirage des réponses à rebours.** $\mathcal{S}$ choisit librement :

$$c,\ z_v,\ z_r,\ z_{sk} \xleftarrow{\$} \mathbb{Z}_q.$$

Ces valeurs jouent le rôle des réponses Schnorr dans la transcription simulée.

ii. **Reconstruction des engagements.** Les équations de vérification des Étapes 3 et 4 imposent :

$$A = g^{z_v} h^{z_r} \cdot C^{-c} \pmod{p},$$
$$B = g^{z_{sk}} \cdot \mathsf{pk}^{-c} \pmod{p}.$$

$\mathcal{S}$ calcule A et B ainsi ; par construction, les équations de vérification des Étapes 3 et 4 sont satisfaites.

iii. **Programmation de l'oracle (rom).** $\mathcal{S}$ programme :

$$H\Big(g \| h \| C \| \mathsf{nul} \| \mathsf{root} \| A \| B \| \varepsilon\Big) \;:=\; c,$$

de sorte que le recalcul du défi à l'Étape 2 du vérificateur produise exactement $c' = c$.

iv. **OR-proof simulée.** $\mathcal{S}$ choisit :

$$c_0,\ z_{0v},\ z_{0r},\ z_{1v},\ z_{1r} \xleftarrow{\$} \mathbb{Z}_q,$$

pose $c_{or} := c_0 + c'_1 \bmod q$ pour une valeur $c'_1 := c_{or}^{\mathsf{fresh}} - c_0$ programmée dans l'oracle, puis reconstruit A_0, A_1 par la méthode du Lemme 5.1.

Comparaison des distributions.

Dans la trace réelle, le prouveur génère :

$$z_v = k_v + c \cdot \mathsf{vote} \bmod q, \quad z_r = k_r + c \cdot r \bmod q, \quad z_{sk} = k_{sk} + c \cdot \mathsf{sk} \bmod q,$$

avec $k_v, k_r, k_{sk} \xleftarrow{\$} \mathbb{Z}_q$. Pour toute valeur fixée du témoin $(\mathsf{vote}, r, \mathsf{sk})$, chaque $z.$ est un décalage affine d'une valeur uniforme : sa distribution est **uniforme dans** $\mathbb{Z}_q$.

Le simulateur tire (z_v, z_r, z_{sk}) directement uniformément : les distributions sont identiques. De plus, A et B sont uniquement déterminés par $(z_v, z_r, z_{sk}, c, C, \mathsf{pk})$ via les équations de reconstruction : l'identité des distributions sur les réponses implique l'identité sur les engagements.

Élément de π	**Trace réelle**	**Trace simulée**	**Distribution**
z_v	$k_v + c\,\mathsf{vote} \bmod q$	tiré uniforme	$\mathcal{U}(\mathbb{Z}_q)$
z_r	$k_r + c\,r \bmod q$	tiré uniforme	$\mathcal{U}(\mathbb{Z}_q)$
z_{sk}	$k_{sk} + c\,\mathsf{sk} \bmod q$	tiré uniforme	$\mathcal{U}(\mathbb{Z}_q)$
A	$g^{k_v} h^{k_r}$	$g^{z_v} h^{z_r} C^{-c}$	identique
B	$g^{k_{sk}}$	$g^{z_{sk}}\,\mathsf{pk}^{-c}$	identique
Vérif. Étapes 3-4	satisfaites	satisfaites	—
OR-proof	honnête (Lemme 5.1)	simulée (Lemme 5.1)	identique

La distribution jointe de la trace simulée $\mathcal{S}(x)$ est identique à celle de la vraie interaction : $|\Pr[\mathcal{D}(\langle \mathcal{P}, \mathcal{V} \rangle) = 1] - \Pr[\mathcal{D}(\mathcal{S}(x)) = 1]| = 0$.

Tout ce qu'un vérificateur peut calculer après avoir vu π, il pourrait le calculer sans elle : la trace ne contient aucune information sur le témoin $(\mathsf{vote}, r, \mathsf{sk})$.

5.5.5 Résistance aux attaques spécifiques au vote

TABLE 5.5 – Résistance de Π_{VoteZK} aux attaques du vote électronique.

Attaque	**Mécanisme de défense**	**Référence**
Double-vote	Nullifieur $H(\mathsf{sk} \| \varepsilon)$ unique par élection.	§ 5.4.1
Vote invalide ($\notin \{0, 1\}$)	OR-proof CDS vérifiée à l'Étape 5.	§ 5.3.4
Électeur non inscrit	Chemin de Merkle dans π vérifié à l'Étape 6.	§ 5.3.6
Rejeu (*replay*)	Nonce ε intégré dans le défi Fiat-Shamir.	§ 5.3.5
Compromission du serveur	L'urne ne stocke que $(C, \mathsf{nul}, \mathsf{or_pi}, \pi)$, jamais $(\mathsf{vote}, \mathsf{sk})$ en clair.	Théorème 5.3
Attaque quantique (Shor)	Hors périmètre ; voir § 5.6.1.	[29, 12]

5.6 Limites du protocole et perspectives d'amélioration

Aucun prototype ne prétend à la complétude. Cette section identifie les cinq limites structurelles de Π_{VoteZK} (paramètre de sécurité sous-dimensionné, fragilité quantique, dépouillement centralisé, modèle de l'oracle aléatoire, scalabilité du décompte) et propose pour chacune une piste de résolution concrète ancrée dans la littérature.

5.6.1 Limites identifiées

- **Paramètre de sécurité sous-dimensionné.** $\lambda = 128$ bits dans $\mathbb{Z}_p^*$ n'offre pas 128 bits de sécurité effective (NFS, § 1.2). Production : $p \geq 2048$ bits [25] ou migration vers Curve25519.
- **Dépouillement distribué.** $\mathsf{somme_r}$ accessible au scrutateur uniquement en circuit fermé. Un dépouillement multi-parties (DHE) est nécessaire en production [27] : le secret est partagé entre k scrutateurs par un schéma de Shamir ; aucun d'eux ne peut reconstruire la somme seul.
- **Fragilité quantique.** L'algorithme de Shor [29] résoudrait le DLP en temps polynomial sur un ordinateur quantique à grande échelle. Π_{VoteZK} hérite de cette vulnérabilité via Schnorr [6].
- **Modèle de l'oracle aléatoire.** La transformation de Fiat-Shamir [1] n'est prouvée sûre que dans ce modèle idéalisé [23]. La sécurité dans le modèle standard reste ouverte.
- **Scalabilité du décompte.** BSGS est en $\mathcal{O}(\sqrt{N})$ en temps et en espace. Pour $N = 10^6$, la table contient 10^3 éléments (≈ 16 Ko). Praticable pour un bureau de vote, mais contraignant à l'échelle nationale.

Π_{ZK} : Prove($x \in L$) $\wedge \neg$Reveal(x)

5.6.2 Perspectives d'amélioration

- **zk-STARKs [12] :** éliminent la dépendance au DLP et offrent une résistance quantique naturelle, au prix d'une taille de preuve de l'ordre du kilooctet.
- **zk-SNARKs (Groth16 / PLONK [16]) :** taille de preuve constante ($\approx$ 288 octets) et vérification quasi-instantanée, au prix d'un *trusted setup*.
- **Curve25519 [8] :** remplacer $\mathbb{Z}_p^*$ par Curve25519 réduit de moitié la taille des clés sans modifier la structure logique du protocole.
- **Intégration blockchain :** publier $(\mathsf{nul}, \mathsf{root})$ sur une blockchain publique garantirait la transparence et l'immutabilité des bulletins sans autorité centrale, permettant un audit citoyen indépendant.

Conclusion

Π_{VoteZK} repose sur quatre primitives : Pedersen [37] pour la confidentialité, Schnorr–Fiat-Shamir [6, 1] pour la non-interactivité, la OR-proof CDS [30] pour la validité binaire des bulletins, et Merkle pour l'éligibilité. Il satisfait les cinq propriétés requises (éligibilité, unicité, validité, confidentialité, vérifiabilité) dont les preuves formelles de

complétude, robustesse (Théorème 5.2) et divulgation nulle (Théorème 5.3) sont toutes conduites dans le modèle de l'oracle aléatoire [23].

L'implémentation Python valide ces résultats : dix tests unitaires passent sans échec, et les benchmarks confirment les complexités théoriques $\mathcal{O}(N)$ pour la génération et la vérification, $\mathcal{O}(\sqrt{N})$ pour le dépouillement, avec un surcoût de $\times 1{,}7$ dû à la OR-proof.

Les limites identifiées (fragilité quantique, modèle de l'oracle aléatoire, scalabilité du BSGS, dépouillement centralisé) sont des contraintes héritées du DLP, communes à tous les protocoles de cette famille. Leur résolution naturelle passe par les zk-STARKs [12], Groth16 [16] et un dépouillement multi-parties [27].

$\Pi_{\mathsf{ZK}} : \mathsf{Prove}(x \in L) \wedge \neg\mathsf{Reveal}(x)$

CONCLUSION GÉNÉRALE

Ce mémoire a permis d'explorer de bout en bout le paradigme des preuves à divulgation nulle de connaissance, en traçant une trajectoire complète allant de leurs fondements mathématiques abstraits jusqu'à leur validation expérimentale. Dans un premier temps, nous avons établi les structures algébriques sur corps finis, les problèmes réputés difficiles comme le logarithme discret, ainsi que les primitives indispensables que sont les fonctions de hachage et les schémas d'engagement. Ce socle a permis de formaliser rigoureusement la triple exigence qui définit les ZKP, à savoir la complétude, la robustesse et la divulgation nulle. En cartographiant les protocoles de la littérature, nous avons mis en lumière les compromis entre l'interactivité classique et la non-interactivité moderne issue de la transformation de Fiat-Shamir. Enfin, l'analyse de leurs applications concrètes a mis en perspective le rôle crucial que ces technologies s'apprêtent à jouer dans les architectures décentralisées, notamment dans des contextes en pleine mutation comme celui du Sénégal, où la souveraineté numérique devient un impératif stratégique.

La concrétisation majeure de ce travail réside dans la conception et le développement de Π_{VoteZK}. Ce protocole original de vote électronique démontre de manière tangible qu'il est mathématiquement possible de concilier deux exigences démocratiques historiquement antagonistes : l'anonymat absolu de l'électeur et la transparence totale du décompte. Nous avons prouvé qu'un utilisateur peut attester de son éligibilité, de l'unicité de son vote et de la validité binaire de son bulletin sans jamais révéler la nature de son choix. Les preuves formelles de sécurité, menées avec rigueur dans le modèle de l'oracle aléatoire [23], s'adossent directement aux briques théoriques posées en début de manuscrit. Pour valider cette architecture, l'implémentation en langage Python a fourni des résultats probants. La validation par une suite de dix tests unitaires et les mesures de performance empiriques confirment la viabilité du protocole, traduisant fidèlement les complexités théoriques en temps d'exécution machine, tant pour la génération des preuves que pour le dépouillement homomorphe.

Malgré ces résultats encourageants, les limites identifiées au cours de nos travaux

sont réelles et soulignent le fossé qui sépare un prototype académique d'un système à déploiement massif. Sur le plan de la sécurité, le choix d'un paramètre de sécurité $\lambda = 128$ bits, bien que suffisant pour nos simulations, devra être réévalué à la hausse pour un système en production. De plus, la dépendance de notre protocole vis-à-vis du problème du logarithme discret le rend structurellement vulnérable à long terme face à l'émergence du calcul quantique et à l'application de l'algorithme de Shor [29]. Enfin, le modèle de l'oracle aléatoire, bien qu'incontournable pour formaliser nos preuves, reste une idéalisation théorique dont les fonctions de hachage réelles ne partagent pas toutes les propriétés idéales. Ces contraintes, loin d'infirmer la valeur de notre travail, partagent les défis globaux de la cryptographie moderne et tracent précisément la feuille de route des développements futurs.

L'évolution naturelle de ce projet s'articule ainsi autour de trois axes scientifiques majeurs. À court terme, la compacité des données transmises pourrait être optimisée en adaptant le protocole au formalisme de Groth16 [16], réduisant la taille des preuves à une dimension constante. À moyen terme, l'intégration d'un schéma de calcul multi-parties sécurisé (MPC) [27] permettrait de fragmenter et de distribuer la clé de déchiffrement du système, éliminant ainsi le risque lié à l'existence d'une autorité de scrutin unique. Enfin, pour inscrire définitivement ce protocole dans l'ère de la sécurité à long terme, une migration vers des constructions fondées sur les réseaux euclidiens ou vers l'usage exclusif de zk-STARKs [12] garantirait une résistance éprouvée face aux menaces post-quantiques.

Au-delà de la rigueur des équations et des lignes de code, ce mémoire illustre le pouvoir transformateur de la cryptographie lorsqu'elle est mise au service du citoyen. Permettre à un individu de prouver son identité, ses droits ou son choix sans jamais exposer ses données privées est un vecteur de confiance indispensable pour nos sociétés futures. En définitive, pour des pays en plein essor technologique, la maîtrise endogène de ces outils de pointe ne relève pas de la simple curiosité scientifique. Elle constitue un levier indispensable pour bâtir des infrastructures numériques indépendantes, transparentes, et pleinement souveraines.

BIBLIOGRAPHIE

[1] A. Fiat and A. Shamir, “How to Prove Yourself : Practical Solutions to Identification and Signature Problems,” in *Proc. CRYPTO ’86*, 1986, pp. 186–194.

[2] A. Noël, “Étude de cas d’utilisation de Zero Knowledge Proof (ZKP),” *Blog OCTO*, 2021. Disponible sur : `https://blog.octo.com/etude-de-cas-dutilisation-de-zero-knowledge-proof-zkp`.Consulté le 28 Fèvrier 2026.

[3] B. Adida,“Helios : Web-based open-audit voting,”in *Proc. 17th USENIX Security Symposium (USENIX Security’08)*, San Jose, CA, USENIX Association, Jul. 2008, pp. 335–348.DOI : 10.5555/1496711.1496734. Disponible : `https://www.usenix.org/conference/17th-usenix-security-symposium/helios-web-based-open-audit-voting`. Consulté le 25 juin 2026

[4] B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater,“Electing a university president using open-audit voting : Analysis of real-world use of Helios,” in *Proc. EVT/WOTE’09 : Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*, USENIX Association, Montreal, Aug. 2009, p. 10.DOI : 10.5555/1855491.1855501. Disponible :`https://dl.acm.org/doi/10.5555/1855491.1855501`. Consulté le 25 juin 2026

[5] B. Bünz, J. Bootle, D. Boneh, et al., “Bulletproofs : Short Proofs for Confidential Transactions and More,” in *Proc. IEEE S&P*, 2018, pp. 315–334.

[6] C.-P. Schnorr, “Efficient Identification and Signatures for Smart Cards,” in *Proc. CRYPTO ’89*, 1989, pp. 239–251.

[7] Direction Générale des Élections / Sénégal Politique, “Organisation électorale : rôle et missions de la DGE,” 2024. Disponible : `https://senegalpolitique.org/organisation-electorale/`. Consulté le 10 juin 2026.

[8] D. J. Bernstein, “Curve25519 : New Diffie-Hellman Speed Records,” in *Proc. PKC ’06*, 2006, pp. 207–228.

[9] D. Pointcheval and J. Stern, “Security Arguments for Digital Signatures and Blind Signatures,” *Journal of Cryptology*, vol. 13, no. 3, pp. 361–396, 2000.

[10] D. Raffo, “Digital Certificates and the Feige-Fiat-Shamir zero-knowledge protocol,” Traineeship report, LIX, Palaiseau, France, 2002.

[11] D. Shanks, “Class Number, a Theory of Factorization, and Genera,” *Proc. Symp. Pure Math.*, vol. 20, pp. 415–440, 1971.

[12] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, “Scalable, transparent, and post-quantum secure computational integrity,” *IACR Cryptol. ePrint Arch.*, p. 46, 2018.

[13] Ethereum Stack Exchange,“ZK-SNARKs vs ZK-STARKs vs Bulletproofs (updated),” *Ethereum Stack Exchange*, 2023.
Disponible sur : `https://ethereum.stackexchange.com/questions/59145/zk-snarks-vs-zk-starks-vs-bulletproofs-updated`. Consulté le 9 mars 2026.

[14] I. Damgård,“On Σ-protocols,” *Lecture Notes, University of Aarhus*, 2010.

[15] Journal du Coin, “Comprendre les Zero-Knowledge Proofs,” 2021. Disponible : `https://journalducoin.com/analyses/preuves-a-divulgation-nulle-de-connaissance-zero-knowledge-proofs-zk-snarks-zk-starks/`. Consulté le 5 avril 2026.

[16] J. Groth, “On the Size of Pairing-Based Non-interactive Arguments,” in *Proc. EUROCRYPT 2016*, *LNCS* vol. 9666, Springer, 2016, pp. 305–326.

[17] J.-J. Quisquater, L. C. Guillou, and T. A. Berson, “How to Explain Zero-Knowledge Protocols to Your Children,” in *Proc. CRYPTO ’89*, 1990, pp. 628–631.

[18] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. CRC Press, 2014.

[19] J. Okz,“Comprendre le Zero-Knowledge Proof et ses applications concrètes,” *Medium*, 2023. Disponible : `https://jonathanokz.medium.com/9e69fa5de37b`. Consulté le 5 avril 2026.

[20] K. Momin,“A Non-Mathematical Introduction to Zero Knowledge Proof,” *Medium*, 2023. Disponible : `https://kayprasla.medium.com/c1a4a269e308`. Consulté le 28 février 2026.

[21] Le Monde Informatique, “Sécurité des mots de passe en 2023,” 2023. Disponible : `https://www.lemondeinformatique.fr/actualites/lire-mots-de-passe-plus-efficaces-et-robustes-90341.html`. Consulté le 4 avril 2026.

[22] LockSelf,“Comprendre le Zero Knowledge Proof (ZKP) (+cas d’usage),” *LockSelf Blog*, 2024. Disponible sur
URL : `https://www.lockself.com/blog/comprendre-zero-knowledge-proof`. Consulté le 4 avril 2026.

[23] M. Bellare and P. Rogaway, “Random oracles are practical : A paradigm for designing efficient protocols,” in *Proc. 1st ACM Conf. CCS ’93*, 1993, pp. 62–73.

[24] M. Blum, P. Feldman, and S. Micali, “Non-Interactive Zero-Knowledge and Its Applications,” in *Proc. STOC ’88*, 1988, pp. 103–112.

[25] National Institute of Standards and Technology, “Recommendation for Key Management, Part 1 : General (Revision 5),” NIST Special Publication 800-57, 2024.

[26] O. Goldreich, *Foundations of Cryptography, Volume 1*. Cambridge University Press, 2001.

[27] O. Goldreich, S. Micali, and A. Wigderson, “Proofs that Yield Nothing But Their Validity,” *Journal of the ACM*, vol. 38, no. 3, pp. 690–728, 1991.

[28] P. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” in *Proc. CRYPTO ’96*, 1996, pp. 104–113.

[29] P. W. Shor, “Algorithms for Quantum Computation : Discrete Logarithms and Factoring,” in *Proc. FOCS ’94*, 1994, pp. 124–134.

[30] R. Cramer, I. Damgård, and B. Schoenmakers, “Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols,” in *Advances in Cryptology — CRYPTO ’94*, Springer, 1994, pp. 174–187.

[31] République du Sénégal, “Loi n° 2005-07 du 11 mai 2005 portant création de la Commission Électorale Nationale Autonome (CENA),” Disponible : `https://www.cena.sn/pages/creation.html`. Consulté le 10 juin 2026.

[32] République du Sénégal,*Code Électoral du Sénégal*, loi n° 2021-35, Ministère de l’Intérieur, 2021. Disponible : `https://interieur.sec.gouv.sn/processus-electoraux/election-legislative`. Consulté le 10 juin 2026.

[33] R. Lidl and H. Niederreiter, *Finite Fields*. Cambridge University Press, 1997.

[34] S. Arora and B. Barak, *Computational Complexity : A Modern Approach*. Cambridge University Press, 2009.

[35] S. A. Cook, “The complexity of theorem-proving procedures,” in *Proc. STOC ’71*, 1971, pp. 151–158.

[36] S. Goldwasser, S. Micali, and C. Rackoff, “The knowledge complexity of interactive proof-systems,” in *Proc. STOC ’85*, 1985, pp. 291–304.

[37] T. Pedersen, “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing,” in *Proc. CRYPTO ’91*, 1991, pp. 129–140.

[38] U. Feige, A. Fiat, and A. Shamir, “Zero-Knowledge Proofs of Identity,” *Journal of Cryptology*, vol. 1, no. 2, pp. 77–94, 1988.

[39] V. Cortier and B. Smyth,“Attacking and fixing Helios : An analysis of ballot secrecy,” *Journal of Computer Security*, vol. 21, no. 1,pp. 89–148, 2013. DOI :

10.3233/JCS-2012-0458. IACR ePrint 2010/625. Disponible : https://eprint.iacr.org/2010/625. Consulté le 25 juin 2026

[40] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[41] Y. Sakemi et al., “Pairing-Friendly Curves,” IETF Internet-Draft, draft-irtf-cfrg-pairing-friendly-curves-11, 2020.

[42] 01net, “États-Unis : un plan pour protéger les objets connectés,” *01net actualités*, 2023. Consulté le 4 avril 2026.

$$\Pi_{\mathsf{ZK}} : \mathsf{Prove}(x \in L) \wedge \neg\mathsf{Reveal}(x)$$