EXERCICES CHAPITRE 1

MVONGO MEDJO ORDI FAREL

14 novembre 2025

Table des matières

Partie 1 : Fondements Philosophiques et Épistémologiques

1 Analyse Critique du Paradoxe de la Transparence

Le philosophe contemporain Byung-Chul Han identifie dans ses œuvres, notamment dans *La Société de la Transparence*, un paradoxe fondamental qui hante nos sociétés modernes. Ce paradoxe réside dans l'idée que la quête absolue de transparence, présentée comme un idéal d'ouverture, de vérité et de confiance, se mue en son contraire : un mécanisme de contrôle et d'appauvrissement de l'existence humaine. La transparence totale, en niant tout espace d'opacité, de secret et de négativité, devient aliénante.
Pour Han, l'individu n'est pas une simple donnée à exposer ; il se construit dans une dialectique entre le visible et l'invisible, le révélé et le caché. La transparence absolue, en exigeant que tout soit mis en lumière, éradique la distance nécessaire à la réflexion, à l'intimité et à la confiance authentique, laquelle repose justement sur un acte de foi envers ce qui n'est pas entièrement vérifiable. La société de la transparence est ainsi une société de la surveillance où l'individu, devenu objet de mesure et de quantification, s'auto-exploite dans une performance permanente sous le regard des autres. Le paradoxe est donc clair : la promesse de liberté par la transparence conduit à une nouvelle servitude.

Application au cas concret : la balance entre transparence gouvernementale et vie privée en Chine

Ce paradoxe éclaire de manière critique des politiques contemporaines, comme l'équilibre entre transparence de l'État et vie privée des citoyens. Prenons le cas de la Chine, qui a développé un système de « crédit social » et une surveillance numérique de masse. L'argument officiel invoque une transparence nécessaire : celle de l'État dans la gestion publique (par exemple, via des plateformes de données ouvertes) et celle des citoyens dans leurs comportements, présentée comme un gage de sécurité, d'ordre social et de confiance.

Cependant, l'analyse hanienne révèle le paradoxe à l'œuvre. La transparence exigée des citoyens n'est pas réciproque. L'opacité des mécanismes de décision de l'État, des critères exacts d'évaluation du crédit social ou de l'usage des données collectées persiste. Cette asymétrie crée un déséquilibre de pouvoir radical. La « confiance » que le système est censé générer n'est pas une confiance authentique, librement consentie entre des égaux, mais une relation de contrôle unilatéral. La vie privée, qui constitue l'opacité fondamentale de l'individu, est laminée au nom d'une sécurité transparente. En niant cette opacité essentielle, le système réduit le citoyen à un ensemble de données comportementales, annihilant sa dimension d'être libre et imprévisible. La transparence promise se transforme ainsi en un instrument de surveillance totale, produisant non pas de la confiance, mais de la conformité et de la peur.

Proposition de résolution pratique inspirée de l'éthique kantienne

Pour dépasser ce paradoxe, l'éthique kantienne offre une piste de résolution pratique. Kant fonde la moralité sur l'impératif catégorique, notamment sa deuxième formulation :

« Agis de telle sorte que tu traites l'humanité, aussi bien dans ta personne que dans la personne de tout autre, toujours en même temps comme une fin, et jamais simplement comme un moyen. »

Appliquée au principe de transparence, cette maxime impliquerait de refuser toute instrumentalisation de la personne. La transparence ne peut être une fin en soi ; elle ne devient légitime que si elle sert à respecter la dignité humaine. Une résolution pratique consisterait à instaurer un **principe de réciprocité et de finalité éthique**.

1. **Réciprocité asymétrique protectrice** : Le niveau de transparence exigé d'un individu ou d'un groupe doit être proportionnel au pouvoir qu'il détient. Un État ou une grande corporation, qui ont un immense pouvoir sur la vie des citoyens, doivent une transparence maximale sur leurs actions et leurs critères. À l'inverse, l'individu, dont le pouvoir est limité, a un droit fondamental à l'opacité (vie privée) comme condition de son autonomie. Cette asymétrie n'est pas injuste ; elle est protectrice de la dignité du plus faible.
2. **Finalité éthique** : Toute exigence de transparence doit être justifiée par une fin précise et légitime (ex. : lutte contre la corruption pour l'État, protection des données pour une entreprise). La collecte de données ne peut être généralisée et sans but précis. Elle doit être strictement encadrée par la loi, limitée dans le temps, et les citoyens doivent avoir un droit de regard et de contestation. La transparence est ainsi un moyen au service de la justice, non un outil de contrôle.

En somme, l'éthique kantienne nous invite à subordonner le principe de transparence au principe de dignité humaine. Il ne s'agit pas de rejeter toute transparence, mais de la canaliser pour qu'elle serve la liberté et l'autonomie des personnes, et non qu'elle les anéantisse. La confiance véritable ne naît pas de la surveillance totale, mais d'institutions justes et d'un respect mutuel qui accepte l'ombre nécessaire à l'épanouissement de l'être.

2 Transformation Ontologique du Numérique

Transformation Ontologique du Numérique : De l'Être heideggérien à l'Être-par-la-trace

1. La conception de l'être chez Heidegger et son adaptation à l'ère numérique

Pour Martin Heidegger, l'être humain (le *Dasein*) se caractérise fondamentalement par son *être-au-monde* (*In-der-Welt-sein*). Le *Dasein* n'est pas une substance isolée qui entre ensuite en relation avec le monde ; il est toujours déjà engagé dans un monde de significations, de préoccupations et de relations. L'outil (*Zeug*) révèle cet engagement : il se retire dans sa disponibilité (*Zuhandenheit*) pour laisser place à l'action. Le marteau n'est pas perçu comme un objet en soi, mais dans son "pour" (*Wozu*), son utilité pour construire.

L'ère numérique opère une transformation radicale de cet être-au-monde. Le monde n'est plus seulement l'horizon de nos actions immédiates, mais un espace de données, un *monde-comme-données*. L'outil numérique, comme un smartphone, ne se retire pas ; au contraire, il s'impose constamment comme un intermédiaire qui enregistre, quantifie et influence notre engagement. Notre être-au-monde devient un **être-sous-capture**. L'essence de la technique moderne, que Heidegger nommait *Gestell* (Dispositif, Arraisonnement),

trouve son accomplissement dans le numérique : le *Dasein* est lui-même désormais "arraisonné", mis en demeure de se rendre disponible et calculable comme une ressource, une *donnée* à optimiser. L'authenticité, qui pour Heidegger résidait dans la capacité à assumer son être-pour-la-mort, est remplacée par une **curiosité inauthentique** amplifiée par les flux numériques, où l'on fuit l'angoisse existentielle dans le zapping perpétuel et la quantification de soi.

2. Étude d'un profil social complet comme manifestation d'« être-par-la-trace »

Prenons l'exemple d'un profil Instagram complet d'un influenceur voyageur, "JulienExplorer".

- **Bio et nom d'utilisateur** : " Aventurier Digital | Photographe" – Une identité performative, définie par son activité en ligne et son désir d'ailleurs.
- **Publications (plus de 1000 posts)** : Chaque photo géolocalisée (Bali, Tokyo, Paris) n'est pas seulement le souvenir d'une expérience vécue, mais une **preuve d'existence** calibrée pour un algorithme et un public. Le like n'est pas une simple approbation, mais une participation à la construction de cette existence.
- **Stories et Reels** : Flux continu de micro-traces (24h) qui créent une illusion de transparence et de vie en direct, mais qui sont une performance soigneusement montée. C'est l'être dans son immédiateté médiatisée.
- **Abonnements/Abonnés** : Le réseau social constitue un "monde-ambiant" numérique, défini par des affiliations à des marques, d'autres influenceurs, des centres d'intérêt. L'être de Julien est un **être-avec** (*Mitsein*) algorithmique.
- **Données de navigation et métadonnées** : Le temps passé sur certaines publications, les interactions, les achats via des liens affiliés sont des traces passives, inconscientes, qui tracent un portrait comportemental bien plus précis que l'image projetée.

Analyse en tant qu'« être-par-la-trace » :

Le profil de Julien n'est *pas* Julien. Il est la **manifestation tangible de son être-numérique**. Son existence sociale, sa valeur économique, son identité même sont désormais constituées par l'agrégat de ces traces actives (publications) et passives (données). Il *est* par les traces qu'il laisse et qui, une fois agrégées, le définissent en retour. Son être n'est plus seulement temporel (être-pour-la-mort), il est **archivable et monétisable**. La quête d'authenticité (montrer le "vrai" soi) est paradoxalement le moteur qui alimente le plus efficacement ce régime de la trace, car elle génère un flux constant de données "personnelles".

3. Impact sur la notion de preuve légale

Cette transformation ontologique bouleverse profondément la notion de preuve légale, traditionnellement fondée sur la matérialité et le témoignage d'un sujet.

1. **Déplacement de l'agentivité et de la crédibilité** : La preuve n'est plus seulement produite par un sujet conscient (un témoin, un aveu) ou un objet matériel (une empreinte). Elle émane de façon diffuse du **système technique lui-même**. La "preuve algorithmique" (une analyse de données de géolocalisation, un profil de comportement) acquiert une autorité supérieure au témoignage humain, jugé

faillible. La parole du *Dasein* est dévalorisée au profit de l'objectivité supposée de la trace numérique.

2. **La préemption de la preuve** : Le régime de la trace crée une **preuve potentielle permanente**. Nos actions laissent des traces *avant* même qu'un délit ou un litige ne survienne. La justice passe d'une logique d'enquête (recherche de preuves *a posteriori*) à une logique de **surveillance prédictive**, où les données permettent d'anticiper les comportements. La présomption d'innocence est érodée par ce soupçon algorithmique permanent.
3. **La crise de l'interprétation** : Une trace numérique (un like, un emplacement) est essentiellement pauvre en sens. Elle nécessite une interprétation algorithmique pour devenir une preuve ("l'accusé était sur les lieux du crime", "l'accusé a manifesté un intérêt pour des contenus radicaux"). Le débat judiciaire ne porte plus sur les faits bruts, mais sur la **fiabilité de l'interprétation technique** et l'opacité des boîtes noires algorithmiques. Qui est le véritable témoin ? L'individu ou l'algorithme qui interprète ses traces ?

Conclusion

Le numérique opère une transformation ontologique majeure : l'*être-au-monde* du *Dasein* devient un **être-par-la-trace**. Cette mutation, où notre existence se définit par nos données, a des conséquences juridiques profondes. La preuve légale, pierre angulaire de la justice, se déplace du sujet conscient vers le système technique, entraînant un bouleversement des équilibres entre preuve et présomption d'innocence, entre parole humaine et objectivité algorithmique. La question philosophique et juridique urgente n'est plus seulement de protéger la "vie privée", mais de préserver la capacité du *Dasein* à exister comme un être de parole et de sens, et non comme une simple collection de traces interprétables par des machines.

3 Calcul d'Entropie de Shannon Appliquée

Voici un script Python complet qui calcule l'entropie de Shannon pour différents types de fichiers :

```
import math
import collections
import os
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def calculate_entropy(data):
    """
    Calcule l'entropie de Shannon d'une s quence de bytes
    """
    if len(data) == 0:
        return 0

    # Compte la fr quence de chaque byte
    byte_counts = collections.Counter(data)
    entropy = 0.0
    total_length = len(data)

```

```
    # Calcul de l'entropie: H = -   p(x) * log2(p(x))
    for count in byte_counts.values():
        probability = count / total_length
        entropy -= probability * math.log2(probability)

    return entropy

# ... (le reste du code Python)
```

Ce script comprend également un fichier `requirements.txt` :

```
numpy
pycryptodome
```

Analyse des Résultats d'Entropie

1. Interprétation des Valeurs d'Entropie

Fichier Texte : H 1.5 bits/caractère

- **Très faible entropie** (sur 8 bits possibles)
- **Interprétation** : Données hautement structurées et redondantes
- **Caractéristiques** :
  - Langage naturel avec fortes redondances linguistiques
  - Fréquence inégale des caractères (e, espace, s très fréquents en français)
  - Fort potentiel de compression (théoriquement ∼80%)
- **Exemple** : "bonjour" → patterns prévisibles, lettres corrélées

Fichier JPEG : H 7.2 bits/octet

- **Entropie élevée** (proche du maximum)
- **Interprétation** : Données déjà compressées et partiellement randomisées
- **Caractéristiques** :
  - Compression DCT + Huffman élimine les redondances
  - Données transformées en fréquences, moins structurées
  - Header JPEG a une entropie plus faible que les données image

Fichier AES : H 7.9 bits/octet

- **Entropie quasi-maximale**
- **Interprétation** : Données parfaitement randomisées
- **Caractéristiques** :
  - Distribution uniforme des bytes (chaque valeur 0-255 équiprobable)
  - Aucun pattern discernable, indépendance statistique
  - Impossible à compresser davantage

2. Seuil de Détection de Chiffrement Automatique

Voici un algorithme de détection basé sur l'entropie :

```python
import math
import numpy as np
from scipy import stats

class DetecteurChiffrement:
    def __init__(self):
        # Seuils d termin s empiriquement
        self.seuils = {
            'texte_non_chiffre': 6.0,  # < 6.0 : donn es structur es
            'zone_incertaine_basse': 6.0,  # 6.0-6.8 : potentiellement compress
            'zone_incertaine_haute': 6.8,  # 6.8-7.4 : compress / chiffrement faible
            'chiffrement_fort': 7.4,  # > 7.4 : chiffrement fort
            'aleatoire_parfait': 7.9  #     7.9 : al atoire parfait
        }

    def analyser_distribution(self, data):
        """Analyse statistique approfondie de la distribution"""
        if len(data) == 0:
            return None

        # Entropie de base
        entropy = self.calculer_entropie(data)

        # Tests statistiques suppl mentaires
        byte_array = np.frombuffer(data, dtype=np.uint8)

        # Test du chi-carr  pour l'uniformit
        observed = np.bincount(byte_array, minlength=256)
        chi2, p_value = stats.chisquare(observed)

        # Test de l' cart     la moyenne
        mean_deviation = np.mean(np.abs(byte_array - 128)) / 128

        # Compressibilit  (test de compression rapide)
        compression_ratio = len(data) / len(self.compress_quick(data))

        return {
            'entropie': entropy,
            'chi2_p_value': p_value,
            'uniformite': p_value > 0.05,  # Si p > 0.05, distribution uniforme
            'deviation_moyenne': mean_deviation,
            'taux_compression': compression_ratio,
            'entropie_normalisee': entropy / 8.0  # Normalis e entre 0 et 1
        }

    # ... (le reste du code)
```

Partie 3 : Révolution Quantique et Ses Implications

4 Expérience de Pensée Schrödinger Adaptée

Version Numérique du Chat de Schrödinger

```python
import threading
import time
import random

class SchrodingerFile:
    def __init__(self, filename):
        self.filename = filename
        self.state = None  # None repr sente la superposition
        self.observed = False
        self.superposition_lock = threading.Lock()

    def create_superposition(self):
        """Cr e un  tat  superpos  pr sent/effac """
        with self.superposition_lock:
            self.state = None  #  tat  superpos
            self.observed = False

    def observe(self):
        """Observation qui effondre la fonction d'onde"""
        with self.superposition_lock:
            if not self.observed:
                # L'observation d termine l' tat  final
                probability_present = 0.5  # Probabilit   gale
                self.state = random.random() < probability_present
                self.observed = True
            return self.state

    def exists(self):
        """V rifie l'existence sans observation directe"""
        if not self.observed:
            return " tat  superpos : pr sent ET effac "
        return "Pr sent" if self.state else "Effac "

# D monstration
file = SchrodingerFile("document_confidentiel.txt")
file.create_superposition()
print(f"Avant observation: {file.exists()}")
print(f"Apr s observation: {'Pr sent' if file.observe() else 'Effac
    '}")
```

Impact sur la Notion de Preuve "Certaine"

1. **Effondrement de la preuve** : L'observation modifie l'état du système
2. **Impossibilité de preuve objective** : La preuve n'existe pas indépendamment de l'observation
3. **Doute systémique** : Incertitude quantique intégrée au processus judiciaire

Protocole d'Observation Minimal

```
class MinimalObservationProtocol:
    def __init__(self):
        self.observations = []
        self.interaction_strength = 0

    def weak_measurement(self, system, precision=0.1):
        """Mesure faible pr servant la superposition"""
        # Impl mentation d'une mesure quantique non destructive
        self.interaction_strength += precision
        if random.random() < precision:
            return system.observe()
        return "Superposition pr serv e"

    def quantum_tomography(self, system, measurements=1000):
        """Tomographie quantique pour caract riser l' tat  sans
           effondrement"""
        results = []
        for _ in range(measurements):
            result = self.weak_measurement(system, 0.01)
            results.append(result)
        return self.analyze_superposition(results)
```

5 Calculs sur la Sphère de Bloch

Qubit avec $= /3$, $= /4$

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

class BlochSphere:
    def __init__(self):
        self.fig = plt.figure()
        self.ax = self.fig.add_subplot(111, projection='3d')

    def qubit_state(self, theta, phi):
        """Calcule l' tat  du qubit |      = cos(  /2)|0    + e^( i  )sin
           (  /2)|1   """
        alpha = np.cos(theta/2)
        beta = np.exp(1j * phi) * np.sin(theta/2)
        return np.array([alpha, beta])

    def probabilities(self, state):
        """Calcule P(0) et P(1)"""
        p0 = np.abs(state[0])**2
        p1 = np.abs(state[1])**2
        return p0, p1

    def plot_state(self, theta, phi):
        """Repr sente le qubit sur la sph re de Bloch"""
        # Coordonn es sur la sph re
        x = np.sin(theta) * np.cos(phi)
        y = np.sin(theta) * np.sin(phi)
```

```
        z = np.cos(theta)

        # Sph re de Bloch
        u = np.linspace(0, 2 * np.pi, 100)
        v = np.linspace(0, np.pi, 100)
        x_sphere = np.outer(np.cos(u), np.sin(v))
        y_sphere = np.outer(np.sin(u), np.sin(v))
        z_sphere = np.outer(np.ones(np.size(u)), np.cos(v))

        self.ax.plot_surface(x_sphere, y_sphere, z_sphere, alpha=0.1)
        self.ax.quiver(0, 0, 0, x, y, z, color='r', linewidth=2)
        self.ax.scatter([x], [y], [z], color='r', s=100)
        self.ax.set_xlabel('X')
        self.ax.set_ylabel('Y')
        self.ax.set_zlabel('Z')
        plt.title(f'Sph re de Bloch:   ={theta:.2f},   ={phi:.2f}')

# Calcul pour    =   /3,    =   /4
bloch = BlochSphere()
theta = np.pi/3  # 60
phi = np.pi/4    # 45
state = bloch.qubit_state(theta, phi)
p0, p1 = bloch.probabilities(state)
print(f" tat  quantique: {state}")
print(f"P(0) = {p0:.4f} ({p0*100:.1f}%)")
print(f"P(1) = {p1:.4f} ({p1*100:.1f}%)")
bloch.plot_state(theta, phi)
plt.show()
```

Résultats :
- $\mathbf{P(0)} = \cos^2(/6) = (3/2)^2 = \mathbf{0.75}$ - $\mathbf{P(1)} = \sin^2(/6) = (1/2)^2 = \mathbf{0.25}$

Impact sur les Systèmes de Preuve Quantique

1. **Preuves probabilistes** : Certitude remplacée par des probabilités
2. **Vulnérabilité à l'observation** : La mesure altère la preuve
3. **Nécessité de protocoles quantiques** : Cryptographie quantique requise

6 Analyse du Théorème de Non-Clonage

Explication du Théorème

```
class NoCloningTheorem:
    def __init__(self):
        self.fundamental_limit = True

    def demonstrate_impossibility(self):
        """D montre l'impossibilit  du clonage parfait"""
        # Pour cloner |      vers |        |      , il faudrait U tel que
            :
        # U(|     |0   ) = |     |      pour tout |
        # Mais l'op rateur U doit  tre  lin aire et unitaire
        # Ce qui est impossible pour des  tats  arbitraires
        states = ['|0   ', '|1   ', '|+   ', '|-   ']
```

```
        for state in states:
            print(f"Tentative de clonage de {state}:  CHEC ")
        return "Clonage parfait impossible - th or me d montr "

# Implications pour la conservation des preuves
class QuantumEvidencePreservation:
    def __init__(self):
        self.original_required = True

    def preservation_challenges(self):
        challenges = [
            "Impossibilit  de copie de sauvegarde",
            "D gradation quantique in vitable",
            "N cessit  de chaine de custody quantique"
        ]
        return challenges
```

Alternative ZK-NR (Zero-Knowledge No-Read)

```
class ZKNRProtocol:
    def __init__(self):
        self.quantum_channel = None
        self.classical_channel = None

    def setup_evidence_proof(self, evidence):
        """Met en place une preuve ZK-NR pour une preuve quantique"""
        # G n re un  tat  de r f rence
        reference_state = self.create_reference(evidence)
        # Protocole de v rification sans r v lation
        verification_token = self.generate_verification_token(
            reference_state)
        return {
            'evidence_hash': self.quantum_hash(evidence),
            'verification_token': verification_token,
            'reference_state': reference_state
        }

    def verify_without_reading(self, proof, challenge):
        """V rifie la preuve sans lire son contenu"""
        # Impl mentation du protocole de v rification  connaissance
            nulle
        response = self.generate_response(proof, challenge)
        return self.validate_response(response)
```

Partie 4 : Paradoxe de l'Authenticité Invisible

7 Formalisation Mathématique du Paradoxe

```
class AuthenticityParadox:
    def __init__(self):
        self.systems = []
        self.h_bar_num = 0.1  # Constante num rique de Planck

```

```
    def evaluate_system(self, authenticity, confidentiality,
        observability):
        """ value  un syst me de preuve selon le paradoxe"""
        # V rifie l'in galit  fondamentale
        inequality_holds = authenticity * confidentiality <= 1 - self.
            h_bar_num

        # Calcul des incertitudes
        delta_A = self.calculate_uncertainty(authenticity)
        delta_C = self.calculate_uncertainty(confidentiality)

        # V rifie le principe d'incertitude
        uncertainty_holds = delta_A * delta_C >= self.h_bar_num / 2

        return {
            'system': (authenticity, confidentiality, observability),
            'inequality_holds': inequality_holds,
            'uncertainty_holds': uncertainty_holds,
            'delta_A': delta_A,
            'delta_C': delta_C
        }

    def experimental_h_bar(self, measurements):
        """D termine exp rimentalement  _num """
        products = []
        for a, c in measurements:
            delta_a = self.calculate_uncertainty(a)
            delta_c = self.calculate_uncertainty(c)
            products.append(delta_a * delta_c)
        return 2 * np.mean(products)

# Application  trois syst mes
paradox = AuthenticityParadox()
systems = [
    (0.8, 0.3, 0.9),  # Syst me 1: Haute authenticit , faible
        confidentialit
    (0.5, 0.7, 0.6),  # Syst me 2:  quilibre
    (0.3, 0.9, 0.2)   # Syst me 3: Haute confidentialit , faible
        authenticit
]

for i, system in enumerate(systems, 1):
    result = paradox.evaluate_system(*system)
    print(f"Syst me {i}: {result}")
```

8 Implémentation ZK-NR Simplifiée

```
import hashlib
import time

class SimpleZKNR:
    def __init__(self):
        self.proofs = {}

    def create_proof(self, data, metadata=None):
        """Cr e une preuve ZK-NR pour des donn es"""
```

```
        timestamp = time.time()
        data_hash = self.quantum_safe_hash(data)
        proof = {
            'hash': data_hash,
            'timestamp': timestamp,
            'metadata_hash': self.quantum_safe_hash(str(metadata)) if
                metadata else None,
            'zk_proof': self.generate_zk_proof(data_hash)
        }
        proof_id = self.quantum_safe_hash(str(proof))
        self.proofs[proof_id] = proof
        return proof_id, proof

    def verify_proof(self, proof_id, challenge):
        """V rifie une preuve sans r v ler son contenu"""
        if proof_id not in self.proofs:
            return False
        proof = self.proofs[proof_id]
        response = self.compute_zk_response(proof, challenge)
        return self.validate_zk_response(response, challenge)

    def measure_overhead(self, data_sizes):
        """Mesure l'overhead computationnel"""
        results = []
        for size in data_sizes:
            data = 'x' * size
            start_time = time.time()
            proof_id, proof = self.create_proof(data)
            end_time = time.time()
            overhead = (end_time - start_time) * 1000  # ms
            results.append((size, overhead))
        return results

# Test du compromis confidentialit /v rifiabilit
zk_nr = SimpleZKNR()
test_data = "Preuve quantique sensible"
proof_id, proof = zk_nr.create_proof(test_data)
print(f"Preuve cr   e : {proof_id}")

# V rification sans r v lation
challenge = "test_challenge"
is_valid = zk_nr.verify_proof(proof_id, challenge)
print(f"V rification r ussie : {is_valid}")

# Mesure d'overhead
sizes = [100, 1000, 10000]
overheads = zk_nr.measure_overhead(sizes)
for size, overhead in overheads:
    print(f"Taille {size}: {overhead:.2f} ms")
```