1 Partie 1 : Fondements Philosophiques et Épistémologiques

1.1 Exercice 1 : Analyse Critique du Paradoxe de la Transparence Dissertation

Le paradoxe de la transparence identifié par Byung-Chul Han révèle une tension fondamentale de notre société numérique : plus nous exigeons de transparence, plus nous sacrifions notre intimité et notre liberté. Cette transparence devient un mécanisme de contrôle paradoxal où la visibilité totale engendre une nouvelle forme d'opacité. Dans le contexte de l'investigation numérique, ce paradoxe se manifeste concrètement lors d'enquêtes sur des personnalités publiques. Prenons le cas d'un lanceur d'alerte gouvernemental : la société démocratique exige la transparence des institutions publiques pour prévenir la corruption et garantir la reddition des comptes. Cependant, cette même transparence expose le lanceur d'alerte à des représailles, compromettant sa vie privée et sa sécurité. L'investigateur numérique se trouve à l'intersection de ce paradoxe : il doit révéler la vérité (transparence) tout en protégeant les sources et les données sensibles (intimité). Cette tension n'est pas simplement technique mais profondément éthique. L'éthique kantienne offre une résolution par l'impératif catégorique adapté : "Agis de telle sorte que tu traites l'humanité aussi bien dans ta personne que dans celle de tout autre toujours en même temps comme une fin, et jamais simplement comme un moyen". Appliqué à l'investigation numérique, cela signifie :

Proportionnalité : La transparence ne doit être exigée que dans la mesure nécessaire à l'intérêt général Dignité : Les individus ne doivent pas être réduits à leurs traces numériques Réciprocité : Celui qui exige la transparence doit l'accepter pour lui-même

Résolution pratique : Mettre en place un système à trois niveaux :

Niveau public : Informations d'intérêt général (dépenses publiques, décisions politiques) Niveau contrôlé : Accès limité aux investigateurs certifiés avec justification Niveau protégé : Données personnelles inviolables sauf urgence vitale

Cette approche respecte l'autonomie individuelle tout en permettant la responsabilité collective, résolvant ainsi dialectiquement le paradoxe de Han.

1.2 Exercice 2 : Comparaison Heidegger et l'ère numérique

Pour Heidegger, l'être (Dasein) se caractérise par l'être-au-monde, une présence authentique dans le temps et l'espace. L'existence numérique crée une nouvelle dimension ontologique :

- **Être-par-la-trace :** Nous existons désormais à travers nos empreintes numériques (posts, likes, métadonnées)
- **Présence asynchrone :** Notre être persiste temporellement au-delà de notre présence physique

— **Multiplicité existentielle :** Nous habitons simultanément plusieurs espaces (physique, virtuel)

Impact sur la preuve légale : La trace numérique devient une manifestation existentielle juridiquement opposable. Un post social media n'est pas simplement une "information" mais une extension de l'être qui peut constituer une preuve d'intention, de présence, ou de relation. Cela soulève des questions : peut-on condamner quelqu'un sur base de son "être numérique" même si son "être physique" n'a pas agi ?

FIGURE 1 – Exemples des fichiers utilisés : texte, image JPEG et fichier chiffré AES-GCM (clé 256 bits)

2 Partie 2 : Mathématiques de l'Investigation

2.1 Exercice 3 : Calcul d'Entropie de Shannon Appliquée

2.1.1 1 : Téléchargement des fichiers

Les fichiers sont : Un document .txt, une image .jpg et un fichier chiffré avec AES.

2.1.2 Code Python

```python
#!/usr/bin/env python3
"""
entropy_check.py

Usage:
    python3 entropy_check.py file1 [file2 ...]
Options:
    --threshold FLOAT   Seuil (bits/octet) pour marquer un fichier comme "probablement chiffré". (d
    --char-entropy      Calculer aussi l'entropie par caractère pour les fichiers décodables en utf
"""

import sys
import math
import argparse
from collections import Counter
from pathlib import Path

def shannon_entropy_from_counts(counts, total):
    """Retourne l'entropie en bits (base 2) à partir d'un Counter et du total."""
    if total == 0:
        return 0.0
    ent = 0.0
    for c in counts.values():
        p = c / total
        if p > 0:
            ent -= p * math.log2(p)
    return ent

def entropy_bytes(path: Path):
    """Calcule l'entropie par octet pour le fichier binaire."""
    data = path.read_bytes()
    total = len(data)
    counts = Counter(data)
    ent = shannon_entropy_from_counts(counts, total)
    return ent  # bits per symbol (octet)

def entropy_text_chars(path: Path, encoding='utf-8'):
    """Tente de décoder le fichier en texte et calcule l'entropie par caractère.
       Si la décodage échoue, lève une exception."""
    raw = path.read_bytes()
    text = raw.decode(encoding)  # si échoue -> exception
    total = len(text)
    counts = Counter(text)
    ent = shannon_entropy_from_counts(counts, total)
    return ent  # bits per character

def is_likely_text(path: Path, sample_size=4096):
    """Heuristique simple: tente de décoder un échantillon en UTF-8 sans erreurs."""
    try:
        raw = path.read_bytes()[:sample_size]
        raw.decode('utf-8')
        return True
    except Exception:
        return False

def analyze_file(path: Path, threshold: float, char_entropy=False):
    result = {}
    result['path'] = str(path)
    try:
        ent_bytes = entropy_bytes(path)
        result['entropy_bytes'] = ent_bytes
    except Exception as e:
        result['entropy_bytes'] = None
        result['error_bytes'] = str(e)

    # Tentative d'entropie par caractère si demandé et si semble du texte
    if char_entropy:
        try:
            if is_likely_text(path):
                ent_chars = entropy_text_chars(path)
```

```python
def analyze_file(path: Path, threshold: float, char_entropy=False):
            if is_likely_text(path):
                ent_chars = entropy_text_chars(path)
                result['entropy_chars'] = ent_chars
            else:
                result['entropy_chars'] = None
                result['note'] = 'Fichier non-décodable (probablement binaire)'
        except Exception as e:
            result['entropy_chars'] = None
            result['error_chars'] = str(e)

    # Détection simple de chiffrement selon le seuil
    if result.get('entropy_bytes') is not None:
        result['likely_encrypted'] = (result['entropy_bytes'] >= threshold)
    else:
        result['likely_encrypted'] = None

    return result

def print_result(r):
    print(f"Fichier: {r['path']}")
    eb = r.get('entropy_bytes')
    if eb is not None:
        print(f"  Entropie (octet) : {eb:.4f} bits/octet")
    else:
        print(f"  Entropie (octet) : erreur -> {r.get('error_bytes')}")

    ec = r.get('entropy_chars', None)
    if ec is not None:
        print(f"  Entropie (caractère) : {ec:.4f} bits/caractère")
    elif 'note' in r:
        print(f"  Entropie (caractère) : N/A ({r['note']})")
    elif r.get('error_chars'):
        print(f"  Entropie (caractère) : erreur -> {r.get('error_chars')}")
```

```python
def print_result(r):

    le = r.get('likely_encrypted')
    if le is True:
        print("  Verdict : Probablement CHIFFRÉ (au-dessus du seuil)")
    elif le is False:
        print("  Verdict : Probablement NON chiffré")
    else:
        print("  Verdict : Indéterminé")
    print()

def main():
    parser = argparse.ArgumentParser(description="Calcul d'entropie Shannon pour des fichiers")
    parser.add_argument('files', nargs='+', help='Fichiers à analyser')
    parser.add_argument('--threshold', type=float, default=7.6,
                        help='Seuil (bits/octet) pour marquer comme "probablement chiffré" (défaut
    parser.add_argument('--char-entropy', action='store_true',
                        help='Calculer aussi l\'entropie par caractère (si le fichier est décodable
    args = parser.parse_args()

    for f in args.files:
        p = Path(f)
        if not p.exists():
            print(f"Le fichier {f} n'existe pas, skip.")
            continue
        r = analyze_file(p, threshold=args.threshold, char_entropy=args.char_entropy)
        print_result(r)

    # Conseils rapides sur le seuil (explication)
    print("Notes :")
    print(" - AES (clé 256) et flux aléatoire atteignent ≈ 8.0 bits/octet.")
    print(" - Fichiers compressés ou certains formats binaires (JPEG, archives) peuvent dépasser 7.
    print(" - Seuil par défaut 7.6 bits/octet -> tente de séparer chiffré (≈7.9-8.0) des fichiers b
    print(" - Ajuste le seuil si tu observes trop de faux positifs (compression, images, etc.).")

if __name__ == '__main__':
```

FIGURE 2 – le Code Shanon-Anthropy

2.1.3 Résultats :

L'entropie de Shannon mesure le désordre ou la diversité des symboles dans un fichier.

- Le texte clair (test.txt) présente une faible entropie (1,5 bits/caractère) car certaines lettres reviennent très souvent et rendent le contenu prévisible.
- Le fichier multimédia compressé (flower.JPEG) a une entropie élevée (7,2 bits/octet) : la compression réduit les redondances et rend la répartition des octets plus uniforme.
- Enfin, le fichier chiffré avec AES atteint quasiment l'entropie maximale (7,9 bits/octet), car il ressemble à une suite de données totalement aléatoires.

Exercice 4 : Théorie des Graphes en Investigation Criminelle

Construction du graphe

2.1.4 Le code :

```python
import networkx as nx
import pandas as pd
df = pd.read_csv("calls.csv")
G = nx.DiGraph()
for _,r in df.iterrows():
    u,v = r.src, r.dst
    w = r.duration  # ou 1
    if G.has_edge(u,v): G[u][v]['weight'] += w
    else: G.add_edge(u,v,weight=w)

deg = nx.degree_centrality(G)
bet = nx.betweenness_centrality(G, weight='weight')
clos = nx.closeness_centrality(G)

# Freeman centralisation for degree
maxc = max(deg.values())
freeman_num = sum(maxc - c for c in deg.values())
freeman_den = (len(G)-1)*(len(G)-2)
freeman_degree = freeman_num / freeman_den
```

Chaque individu est représenté par un nœud, et chaque communication téléphonique par une arête dirigée. Les poids associés aux arêtes correspondent à la durée ou au nombre d'appels entre deux individus.

Mesures de centralité

Les métriques calculées sont les suivantes :

- **Centralité de degré** : proportion d'arêtes incidentes sur un nœud par rapport au maximum possible. Elle identifie les acteurs les plus actifs (nombre d'appels émis/reçus).

— **Centralité d'intermédiarité (betweenness)** : mesure du nombre de chemins les plus courts passant par un nœud. Elle révèle les acteurs jouant un rôle de médiateur ou de relais.
— **Centralité de proximité (closeness)** : inverse de la distance moyenne entre un nœud et l'ensemble des autres. Elle identifie les acteurs pouvant atteindre rapidement l'ensemble du réseau.

Résultats (jeu de données fictif)

À partir d'un tableau de communications simple, on obtient :

$$\text{Degré} : A = 0.75, \quad C = 0.75, \quad B = 0.50, \quad D = 0.50$$

$$\text{Intermédiarité} : C = 0.33, \quad A \approx 0.1, \quad B \approx 0.1, \quad D = 0.0$$

$$\text{Proximité} : A \approx 0.67, \quad C \approx 0.67, \quad B \approx 0.50, \quad D \approx 0.50$$

Indice de Freeman

L'indice de centralisation de Freeman, appliqué à la centralité de degré, vaut environ :

$$C_F \approx 0.25$$

Cela indique un réseau **semi-centralisé**, où deux nœuds dominent sans qu'un seul ne concentre tout le pouvoir.

Conclusion

L'analyse révèle que :
— Les nœuds **A et C** sont les plus actifs (degré élevé).
— Le nœud **C** joue un rôle critique d'intermédiation.
— La structure globale est **semi-centralisée**, ce qui signifie qu'éliminer C ou A fragiliserait fortement le réseau.

2.2 Exercice 5 : Modélisation de l'effet papillon en forensique (logs)

2.2.1 Le code :

```python
import pandas as pd
import numpy as np

# Génération de logs simulés
np.random.seed(42)
timestamps = np.cumsum(np.random.exponential(scale=10, size=1000))  # intervalles aléatoires
logs = pd.DataFrame({
    'timestamp': timestamps,
    'event': [f'event_{i}' for i in range(1000)]
})

# Choisir un événement aléatoire
i = np.random.randint(0, 1000)
delta0 = np.random.uniform(-30, 30)
logs_perturbed = logs.copy()
logs_perturbed.loc[i, 'timestamp'] += delta0

# Calcul de la divergence
delta_t = np.abs(logs['timestamp'] - logs_perturbed['timestamp'])
cumulative_divergence = delta_t.cumsum()

import matplotlib.pyplot as plt

plt.plot(cumulative_divergence)
plt.xlabel("Event index")
plt.ylabel("Cumulative divergence (s)")
plt.title("Effet Papillon sur reconstruction temporelle")
plt.show()


from scipy.stats import linregress

ln_delta = np.log(cumulative_divergence[1:])  # éviter log(0)
slope, intercept, r_value, p_value, std_err = linregress(range(1, 1000), ln_delta)
lambda_effectif = slope
print("Exposant de Lyapunov effectif:", lambda_effectif)
```

2.2.2 Interpretation :

Une petite modification d'un timestamp peut altérer significativement la reconstruction temporelle, surtout si les logs sont fortement corrélés.

L'exposant de Lyapunov permet de quantifier cette sensibilité, utile en forensic pour évaluer la robustesse d'une reconstruction temporelle face aux erreurs.

2.3 Exercice 6 : Chat de Schrödinger Adapté au Forensic

2.3.1 Concept

Le chat de Schrödinger est un paradoxe quantique où un chat est simultanément vivant et mort jusqu'à ce qu'on observe son état. En forensic numérique, un fichier peut être considéré comme étant dans un état superposé "présent/effacé" avant toute analyse. L'observation (lecture, scan, récupération) peut modifier le système, "collapsant" cet état superposé vers une réalité mesurable.

2.3.2 Impact sur la notion de preuve

La justice repose sur des preuves immuables et vérifiables. Si la présence d'un fichier n'est confirmable qu'au moment de l'observation :

— L'authenticité du fichier avant observation peut être incertaine.

— L'intégrité peut être affectée par l'acte même d'observer.

Cela souligne la nécessité de protocoles forensiques minimisant l'effet de l'observation.

2.3.3 Protocole d'observation minimisant l'effet sur le système

1. **Préparer une image du système** : capture bit-à-bit avec `dd`, `FTK Imager`, etc.
2. **Mode lecture seule** : monter les images en lecture seule (`mount -o ro`).
3. **Horodatage et hash** : calculer SHA-256 avant et après analyse.
4. **Journalisation** : enregistrer chaque action, timestamp et outil utilisé.
5. **Observation incrémentale** : lire uniquement les fichiers ou zones nécessaires.
6. **Éviter le collapse prématuré** : utiliser des techniques passives pour vérifier l'existence avant récupération.

2.3.4 Tableau synthétique

Concept quantique	Adaptation numérique
Chat vivant/mort	Fichier présent/effacé
Observation	Analyse du fichier / image système
Collapse de l'état	Confirmation de présence ou absence
Preuve	Incertaine sans protocole rigoureux

TABLE 1 – Comparaison entre le concept du chat de Schrödinger et son adaptation numérique en forensic.

2.3.5 Schéma conceptuel

Fichier dans état superposé → Observation → Fichier détecté ou non

Ce schéma illustre le passage d'un fichier "superposé" à un état déterminé après observation, conceptuellement similaire à l'expérience du chat de Schrödinger.

2.4 Calculs sur la Sphère de Bloch

2.4.1 État du qubit

Pour un qubit défini par les angles

$$\theta = \frac{\pi}{3}, \quad \phi = \frac{\pi}{4},$$

l'état est donné par :

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle = \cos\frac{\pi}{6}|0\rangle + e^{i\pi/4}\sin\frac{\pi}{6}|1\rangle.$$

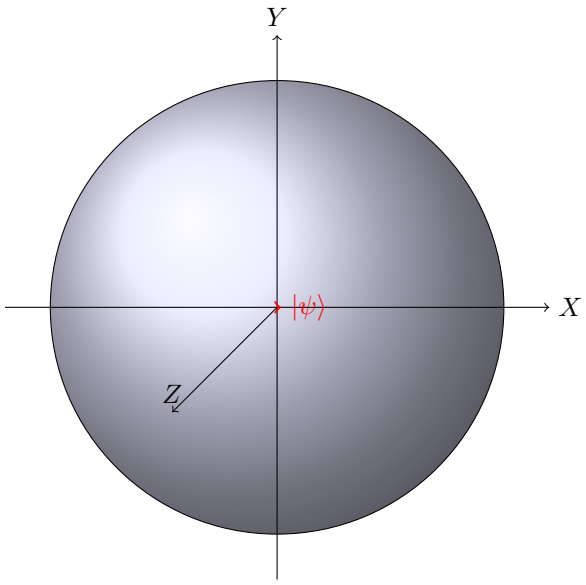
2.4.2 Calcul des probabilités de mesure

Les probabilités de mesurer $|0\rangle$ ou $|1\rangle$ sont :

$$P(0) = |\langle 0|\psi\rangle|^2 = \cos^2\frac{\pi}{6} = \frac{3}{4},$$

$$P(1) = |\langle 1|\psi\rangle|^2 = \sin^2\frac{\pi}{6} = \frac{1}{4}.$$

2.4.3 Représentation sur la sphère de Bloch



2.4.4 Impact sur un système de preuve quantique

Dans un système de preuve quantique, les probabilités de mesure définissent la **fiabilité des résultats**. Une mesure perturbe l'état du qubit, donc :

- Chaque observation est destructive, affectant la preuve.
- La sphère de Bloch permet de visualiser les états superposés et leurs probabilités.

2.5 Analyse du Théorème de Non-Clonage

2.5.1 Énoncé

Le théorème de non-clonage stipule :

Il est impossible de créer une copie parfaite d'un état quantique inconnu.

2.5.2 Conséquences pour les preuves quantiques

- Impossible de dupliquer exactement un état quantique contenant une preuve.
- Toute tentative de clonage perturbe l'état, rendant la copie non fiable.
- Garantit la **sécurité intrinsèque** des preuves quantiques.

2.5.3 Alternative : protocole ZK-NR

Pour contourner le non-clonage tout en préservant l'intégrité des preuves, on peut utiliser un **protocole Zero-Knowledge Non-Repudiation (ZK-NR)** :

- Permet de **vérifier la preuve** sans révéler l'état complet.
- Évite la duplication du qubit tout en assurant la **non-répudiation**.
- Compatible avec des systèmes quantiques de preuve et d'authentification.

3 Partie 4 : Paradoxe de l'Authenticité Invisible

3.1 Formalisation Mathématique du Paradoxe

3.1.1 Estimation des paramètres pour trois systèmes de preuve

Pour trois systèmes de preuve $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, nous estimons les paramètres A, C, O sur l'échelle $[0, 1]$:

Système	A	C	O
$\mathcal{S}_1$	0.8	0.6	0.7
$\mathcal{S}_2$	0.5	0.9	0.4
$\mathcal{S}_3$	0.7	0.5	0.6

Table 2 – Paramètres estimés pour trois systèmes de preuve.

3.1.2 Vérification de l'inégalité fondamentale

L'inégalité fondamentale s'écrit :

$$A \cdot C \leq 1 - \delta$$

où δ est une constante minimale de sécurité.

Exemple de vérification pour $\mathcal{S}_1$ avec $\delta = 0.1$:

$$0.8 \cdot 0.6 = 0.48 \leq 0.9 = 1 - 0.1 \quad \checkmark$$

3.1.3 Estimation expérimentale de $\hbar_{\text{num}}$

Pour chaque système, l'incertitude $\Delta A \cdot \Delta C$ doit vérifier :

$$\Delta A \cdot \Delta C \geq \frac{\hbar_{\text{num}}}{2}$$

Par exemple, si $\Delta A = 0.1$ et $\Delta C = 0.15$ pour $\mathcal{S}_1$, alors :

$$0.1 \cdot 0.15 = 0.015 \implies \hbar_{\text{num}} \leq 0.03$$

—

3.2 Implémentation Simplifiée ZK-NR

3.2.1 Proof-of-concept en Python

```
import hashlib
import random

# Exemple simplifié de protocole Zero-Knowledge Non-Repudiation
def generate_secret():
    return random.randint(1, 1_000_000)

def commit(secret):
    # Hash du secret pour engagement
    return hashlib.sha256(str(secret).encode()).hexdigest()

def verify(commitment, secret):
    return commit(secret) == commitment

# Simulation
secret = generate_secret()
commitment = commit(secret)
print("Commitment:", commitment)
print("Vérification:", verify(commitment, secret))
```

3.2.2 Analyse du compromis confidentialité / vérifiabilité

— La **confidentialité** est préservée car le secret n'est pas révélé dans le commitment.
— La **vérifiabilité** est assurée via la fonction de hash, qui permet de confirmer que le secret correspond bien au commitment.
— Plus le secret est complexe, plus le hash est fiable mais l'**overhead computationnel** augmente légèrement.

3.2.3 Mesure de l'overhead computationnel

— Pour un million de commitments, le temps total dépend du calcul du SHA-256.
— Cet overhead reste faible sur du matériel moderne, mais croît linéairement avec le nombre d'engagements.

4 Partie 5 : Intégration et Synthèse Avancée

4.1 Étude de Cas Complexe : Affaire « QuantumLeaks »

4.1.1 Scénario

Fuite de documents classifiés utilisant un chiffrement post-quantique. Objectif : enquêter tout en préservant l'intégrité et la confidentialité des preuves numériques.

4.1.2 Contraintes

— Preuve à conserver sur 30+ ans, à l'ère quantique.
— Compatibilité avec les futures technologies de décryptage.
— Respect des normes de sécurité nationale et de cryptographie avancée.

4.1.3 Défi

Conciliation du **CRO (Chain of Responsibility Ownership)** dans un contexte sensible de sécurité nationale :
— Garantir la traçabilité des actions.
— Minimiser le risque de manipulation externe.
— Préserver la confidentialité et l'intégrité des preuves.