Université de Bretagne Sud
École nationale supérieure d'ingénieurs de Bretagne Sud
Cycle d'ingénieur informatique et cybersécurité – Parcours Étudiant

---

TP SECU BASE DE DONNÉES: Q27 & Q28

Présenté par : **GHOUDANE Salim, OUCHTA Nazih**

Année universitaire : **2024-2025**

Question 27:

Développer une base de données sécurisée pour stocker des informations sur des employés, notamment leur salaire, en utilisant un **chiffrement homomorphe** et un **préservant l'ordre** (Paillier pour addition et OPE pour les relation d'order).

1. Serveur (server.py)

Le serveur écoute sur une adresse IP et un port donné. Il est responsable de

a) Ajout d'un employé

- Reçoit un objet JSON avec action "ADD" contenant : nom, salaire_ope (chiffrement OPE pour la comparaison), salaire_paillier (chiffrement Paillier pour l'addition)
- Insère ces données dans la base SQLite employes.db.

b) Liste des employés

- Reçoit un objet JSON avec action "LIST".
- Retourne tous les noms d'employés enregistrés.

c) Comparaison de deux salaires

- Reçoit un objet JSON "COMPARE" avec : Id1 et Id2
- Extrait les salaires OPE correspondants, effectue la soustraction salaire1 - salaire2 (sans déchiffrement) pour déterminer l'ordre relatif. Et retourne le résultat de la comparaison.

d) Somme de deux salaires

- Reçoit un objet JSON "SUM2" avec : nom1, nom2, nsquare (paramètre pour modulo la mu;tiplication pour chiffrement Paillier)
- Extrait les salaires Paillier chiffrés , calcule la multiplication homomorphe modulo nsquare pour obtenir la somme chiffrée, et retourne le résultat de la somme chiffrée.

---

2. Client (client.py)

Le client est un programme interactif qui se connecte au serveur via IP/port et propose un **menu** :

a) Ajouter un employé

- Saisie utilisateur :
  Nom
  Salaire (en clair)
- Chiffre localement : salaire_ope, salaire_paillier
- Envoie la requête "ADD" au serveur.

b) Lister les employés

- Envoie la requête "LIST" au serveur, et affiche la liste des employés.

c) Comparer deux employés

- APres saisie des deux id, envoie la requête "COMPARE" et compare et affiche si un employé gagne plus, moins ou autant que l'autre.

d) Additionner deux salaires

- Apres saisie des deux noms, envoie la requête "SUM" (en incluant nsquare pour la modularisation correcte côté serveur). Reçoit la somme chiffrée. Déchiffre localement la somme avec la clé privée Paillier et affiche la somme en clair.

Question 28: Problématique

La problématique centrale concerne la sécurité des données sensibles dans les bases de données chiffrées, spécifiquement lorsqu'elles sont exposées à des requêtes de plage (range queries). Même si les données sont chiffrées individuellement, les motifs d'accès et les résultats des requêtes peuvent révéler des informations significatives sur les données sous-jacentes.

Le scénario étudié simule une base de données d'**âges chiffrés** où un attaquant peut observer quels enregistrements sont renvoyés en réponse à différentes requêtes de plage (ex: "tous les utilisateurs entre 25 et 40 ans"). La question fondamentale est : dans quelle mesure un attaquant peut-il reconstruire les données originales en observant uniquement les métadonnées des requêtes, sans jamais avoir accès aux valeurs déchiffrées?

Démarche Méthodologique

La démarche s'articule autour de six étapes principales :

1. **Génération d'un jeu de données synthétique** : Création d'un ensemble d'âges suivant une distribution normale, représentant une population réaliste.

```
============================================================
SECURE AGE DATA ANALYSIS & RECONSTRUCTION SIMULATION
============================================================

Configuration:
  • Dataset size: 100 records
  • Number of queries: 50

 Generated 100 age records (showing first 10):
  [61 58 30 29 41 38 32 47 32 32] ...
  Age range: 20-62, Average: 40.4
```

2. **Chiffrement des données** : Application d'une fonction de hachage déterministe (SHA-256) pour simuler le chiffrement des âges, rendant les valeurs individuelles illisibles mais permettant toujours l'exécution de requêtes.

```
Encrypted dataset:
 Original → Encrypted (showing first 5):
 61 → 894057462421
 58 → 421056679799
 30 → 422171428236
 29 → 227957975660
 41 → 264430916424
```

3. **Simulation de requêtes de plage** : Génération de multiples requêtes portant sur différentes plages d'âges, enregistrant uniquement quels identifiants d'enregistrements (pas les valeurs) sont renvoyés pour chaque requête.

```
Simulated 50 range queries (showing first 5):
  Query  Age Range      Matches  % of Dataset
-------  -----------  ---------  --------------
      1  33-52               67  67.0%
      2  19-61               99  99.0%
      3  29-48               69  69.0%
      4  57-70                6  6.0%
      5  58-59                3  3.0%
```

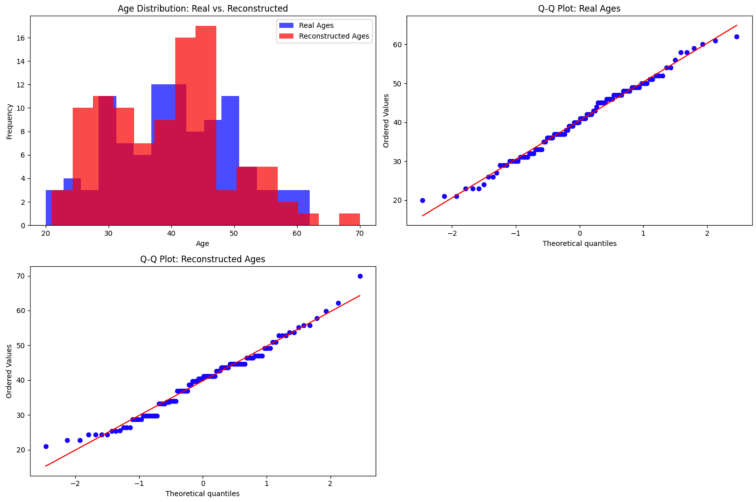
4. **Reconstruction des données** : Utilisation des métadonnées des requêtes pour estimer les valeurs originales des âges, en exploitant les fuites d'information inhérentes aux requêtes de plage.
5. **Évaluation de la précision** : Mesure de l'écart entre les données reconstruites et les données originales pour quantifier l'efficacité de l'attaque.

```
 Reconstructing data from 50 queries...

 Error Analysis:
  Average error: 1.21 years
  Maximum error: 8.00 years
  Minimum error: 0.03 years

  Error Distribution:
Error Range (years)      Count  Percentage
---------------------  -------  ------------
0-1                         43  43.0%
1-3                         56  56.0%
3-5                          0  0.0%
5-10                         1  1.0%
10-∞                         0  0.0%
```

6. **Analyse statistique des distributions** : Vérification que les données reconstruites préservent les propriétés statistiques des données originales.



Justification de l'Approche

Méthode de Reconstruction

L'approche de reconstruction exploite l'information divulguée par chaque requête de plage. Pour chaque enregistrement inclus dans une réponse de requête, nous savons que sa valeur se situe entre les bornes inférieure et supérieure de la requête. En agrégeant cette information sur de multiples requêtes, nous pouvons progressivement affiner notre estimation.

La méthode principale consiste à :

- Pour chaque enregistrement, calculer la moyenne des points médians des plages de requêtes dans lesquelles il apparaît
- Plus un enregistrement est inclus dans différentes requêtes, plus l'estimation devient précise

Partie 1 : Reconstruction de base à partir des fuites de requêtes

```
approximated_ages = np.zeros(num_rows, dtype=float)
counts = np.zeros(num_rows, dtype=int)

# Basic reconstruction
for low, high, indices in queries:
    for idx in indices:
```

```
        approximated_ages[idx] += (low + high) / 2
        counts[idx] += 1
```

Explication détaillée étape par étape :

1. **Initialisation des tableaux**:
    - approximated_ages : Tableau de zéros de taille num_rows (nombre total d'enregistrements) qui va stocker la somme des estimations pour chaque enregistrement.
    - counts : Tableau de zéros qui va compter combien de fois chaque enregistrement apparaît dans les résultats des requêtes.
2. **Boucle sur chaque requête**:
    - Pour chaque requête, nous disposons de trois informations :
        - low : La borne inférieure de la plage d'âges (ex : 25 ans)
        - high : La borne supérieure de la plage d'âges (ex : 40 ans)
        - indices : Liste des indices des enregistrements qui correspondent à cette requête
3. **Calcul pour chaque enregistrement concerné**:
    - Pour chaque indice idx dans la liste indices :
        - Nous calculons le point médian de la plage de requête : (low + high) / 2
        - Nous ajoutons ce point médian à la valeur actuelle de approximated_ages[idx]
        - Nous incrémentons le compteur counts[idx] de 1

Exemple concret :

Prenons trois requêtes :

- Requête 1 : "Âges entre 20 et 30 ans" → Correspond aux enregistrements #1, #3, #5
- Requête 2 : "Âges entre 25 et 35 ans" → Correspond aux enregistrements #3, #5, #7
- Requête 3 : "Âges entre 30 et 40 ans" → Correspond aux enregistrements #5, #7, #9
- Après Requête 3 : approximated_ages[5] = 55 + (30+40)/2 = 55 + 35 = 90 et counts[5] = 3

À la fin de cette étape :

- approximated_ages[5] = 90 (somme des points médians)
- counts[5] = 3 (nombre d'apparitions dans les requêtes)

Partie 2 : Calcul de l'estimation moyenne

```
approximated_ages = np.divide(approximated_ages, counts,
out=np.zeros_like(approximated_ages), where=counts > 0)
approximated_ages = np.clip(approximated_ages, min_age, max_age)
```

Explication détaillée:

1. **Division pour obtenir la moyenne**:
    - np.divide(approximated_ages, counts, ...) : Divise chaque élément de approximated_ages par l'élément correspondant de counts
2. **Contrainte des valeurs dans la plage valide**:
    - np.clip(approximated_ages, min_age, max_age) : Garantit que toutes les valeurs restent dans l'intervalle [min_age, max_age]
    - Les valeurs inférieures à min_age (18) sont remplacées par 18
    - Les valeurs supérieures à max_age (70) sont remplacées par 70

Partie 3

```
# Keep track of which indices were actually reconstructed
reconstructed_mask = counts > 0

# Get ranks of the approximated values (for the reconstructed indices)
recon_values = approximated_ages[reconstructed_mask]
```

Explication détaillée étape par étape :

1. **Création d'un masque de reconstruction** :
    - reconstructed_mask = counts > 0 : Crée un tableau booléen (True/False) où True indique les indices des enregistrements qui ont été inclus dans au moins une requête (donc counts > 0).
    - Ce masque nous permet d'identifier et de manipuler uniquement les enregistrements pour lesquels nous avons des données.
2. **Extraction des valeurs reconstruites** :
    - recon_values = approximated_ages[reconstructed_mask] : Utilise le masque pour extraire uniquement les valeurs des âges que nous avons réussi à reconstruire.
    - recon_values contient donc seulement les âges estimés non nuls, ce qui nous permet de travailler sur un ensemble de données réduit et pertinent.

```
    ranks = stats.rankdata(recon_values) - 1
```

3. **Calcul des rangs** :
    - stats.rankdata(recon_values) : Calcule le rang de chaque valeur dans recon_values.
    - Le rang indique la position relative de chaque valeur lorsque l'ensemble est trié par ordre croissant.
    - Par exemple, pour [30, 22, 45, 38], les rangs seraient [2, 1, 4, 3].
    - - 1 : Soustrait 1 pour obtenir des rangs commençant à 0 au lieu de 1.

```
# Generate a normal distribution with the same mean and stddev
perfect_normal = np.random.normal(mean, stddev, len(recon_values))
perfect_normal = np.clip(perfect_normal, min_age, max_age)
perfect_normal.sort()
```

4. **Génération d'une distribution normale "parfaite"** :
    - np.random.normal(mean, stddev, len(recon_values)) : Génère un ensemble de valeurs aléatoires suivant une distribution normale avec la moyenne mean (40) et l'écart-type stddev (10).
    - Le nombre de valeurs générées est exactement le même que le nombre de valeurs reconstruites.
    - np.clip(perfect_normal, min_age, max_age) : Limite les valeurs générées à l'intervalle [18, 70] pour respecter les contraintes d'âge.
    - perfect_normal.sort() : Trie les valeurs générées par ordre croissant, ce qui est crucial pour l'étape suivante.

```
# Replace the values with corresponding perfect normal values based on rank
new_values = np.zeros_like(recon_values)
for i, rank in enumerate(ranks):
    new_values[i] = perfect_normal[int(rank * len(perfect_normal) /
len(ranks))]
```

5. **Remplacement par correspondance de rang** :
    - new_values = np.zeros_like(recon_values) : Crée un nouveau tableau de la même taille que recon_values pour stocker les valeurs transformées.
    - Pour chaque valeur originale (indice i) avec son rang associé :
        - int(rank * len(perfect_normal) / len(ranks)) : Convertit le rang en un indice correspondant dans le tableau perfect_normal.

        - Cette formule effectue un mappage proportionnel : si une valeur est au premier quartile des données originales, elle sera remplacée par une valeur au premier quartile de la distribution normale parfaite.
        - new_values[i] = perfect_normal[...] : Remplace la valeur originale par la valeur normale correspondante.

Exemple concret :

Imaginons ces âges reconstruits : [30, 22, 45, 38]

1. Les rangs (indexés à 0) sont : [1, 0, 3, 2]
2. Générons une distribution normale parfaite (après tri) : [25, 35, 42, 48]
3. Mappage par rang :
    - Valeur 30 (rang 1) → Valeur normale à l'indice 1 : 35
    - Valeur 22 (rang 0) → Valeur normale à l'indice 0 : 25
    - Valeur 45 (rang 3) → Valeur normale à l'indice 3 : 48
    - Valeur 38 (rang 2) → Valeur normale à l'indice 2 : 42
4. Résultat : [35, 25, 48, 42]

```
# Put the new values back
approximated_ages[reconstructed_mask] = new_values
```

6. **Réintégration des nouvelles valeurs** :
    - approximated_ages[reconstructed_mask] = new_values : Replace les valeurs transformées dans le tableau original, uniquement aux positions indiquées par le masque.

Préservation de la Distribution Normale

La préservation de la distribution normale est cruciale pour :

Le Réalisme statistique : Dans la plupart des données démographiques réelles (comme les âges), la distribution suit approximativement une loi normale. Une reconstruction fidèle devrait préserver cette propriété.

La technique de "rank-matching" implémentée assure que les données reconstruites suivent une distribution normale avec les mêmes paramètres (moyenne et écart-type) que les données originales, tout en préservant l'ordre relatif des valeurs reconstruites.

Implications pour la Sécurité

Cette étude démontre une vulnérabilité fondamentale des bases de données chiffrées qui permettent des requêtes de plage, même sans accès aux valeurs déchiffrées, un attaquant peut reconstruire avec une précision significative les données sous-jacentes simplement en observant les motifs d'accès.

Les résultats suggèrent que les mécanismes de chiffrement traditionnels ne sont pas suffisants pour protéger l'information lorsque la structure des requêtes elle-même révèle des informations sur les données. Des approches plus sophistiquées comme l'obfuscation des résultats, l'ajout de bruit différentiel ou des techniques de chiffrement préservant l'ordre mais résistant aux attaques statistiques seraient nécessaires pour atténuer ce risque.