Université de Rennes

istic Informatique Électronique

Année académique: 2024-2025

RAPPORT DE SÉCURITÉ DE BASE DE DONNÉES

Projet : DBMS over Encrypted Data

Réalisé par:
Cyril KONE & Loïc NASSARA

Encadrant:
HEROUARD Clément

1. Introduction.....2
2. Mise en place technique..... 2
2.1. Objectifs..... 2
2.2. Architecture..... 2
3. Réalisation technique..... 3
3.1. Implémentation du chiffrement et déchiffrement.. 3
3.2. Evaluation expérimentale.....4
3.3. Etude de vulnérabilité.....6
CONCLUSION.....6

1. Introduction

Dans un contexte où les cyberattaques sont en constante augmentation, la protection des données sensibles est devenue une priorité absolue pour les organisations. Les systèmes d'information doivent garantir la confidentialité, l'intégrité et la disponibilité des données. Ce projet a pour objectif de démontrer la faisabilité d'intégrer un mécanisme de chiffrement robuste au sein d'une base de données relationnelle PostgreSQL, afin de protéger les informations salariales sensibles. L'algorithme AES 256 en mode CBC a été choisi pour son équilibre entre robustesse et performance, garantissant un haut niveau de sécurité tout en maintenant des temps d'accès aux données acceptables. Ce projet couvre également une évaluation expérimentale de la surcharge induite par le chiffrement.

2. Mise en place technique

2.1. Objectifs

Les objectifs de notre projet sont:

- Implémenter le chiffrement / déchiffrement dans PostgreSQL avec triggers et vues.
- Évaluer expérimentalement l'impact du chiffrement.
- Étudier les vulnérabilités d'algorithmes de chiffrement.

2.2. Architecture

Pour la réalisation de ce projet, nous avons mis en place les composants clés :

- EMP_PLAIN est la table source contenant les informations insérées en clair.
- EMP est la table d'insertion avec trigger de chiffrement automatique.
- EMP_ENCRYPTED est une table cible recevant automatiquement les données chiffrées grâce à un trigger.
- EMP_ENCRYPTED_DECRYPTED est la vue permettant une lecture transparente des salaires en clair, via une fonction de déchiffrement.

Cette approche garantit à la fois la sécurité des données stockées et la simplicité d'utilisation pour les utilisateurs autorisés.

3. Réalisation technique

Initialement, le projet devait être réalisé en s'appuyant sur le serveur de bases de données de l'ISTIC. Cependant, de nombreuses difficultés techniques ont été rencontrées, notamment un problème d'accès et de disponibilité du serveur distant. Nous avons fait le choix de mettre en place un serveur PostgreSQL local.

3.1. Implémentation du chiffrement et déchiffrement

Nous avons suivi les étapes suivantes :
Création d'une table crypto_keys pour stocker la clé AES256.
Développement de fonctions encrypt_salary() et decrypt_salary().
Création d'un trigger AFTER INSERT sur EMP pour chiffrer automatiquement chaque salaire inséré.
Création d'une vue EMP_ENCRYPTED_DECRYPTED pour permettre la lecture des salaires déchiffrés.
Ainsi, chaque fois qu'une ligne est insérée dans EMP, son salaire est automatiquement chiffré dans EMP_ENCRYPTED.
On peut tester en faisant SELECT * FROM ENCRYTED, on obtient:

	empno [PK] integer	ename character varying (256)	job character varying (256)	hiredate date	sal_enc bytea
1	1601	Employee_1	Analyst	2010-07-20	[binary d...
2	1602	Employee_2	Clerk	2013-11-24	[binary d...
3	1603	Employee_3	Engineer	2001-12-23	[binary d...
4	1604	Employee_4	Analyst	2012-06-09	[binary d...
5	1605	Employee_5	Analyst	2010-01-03	[binary d...
6	1606	Employee_6	Clerk	2007-09-14	[binary d...
7	1607	Employee_7	Engineer	2014-11-14	[binary d...

Pour visualiser le déchiffrement, on fait SELECT * FROM EMP_ENCRYPTED_DECRYPTED

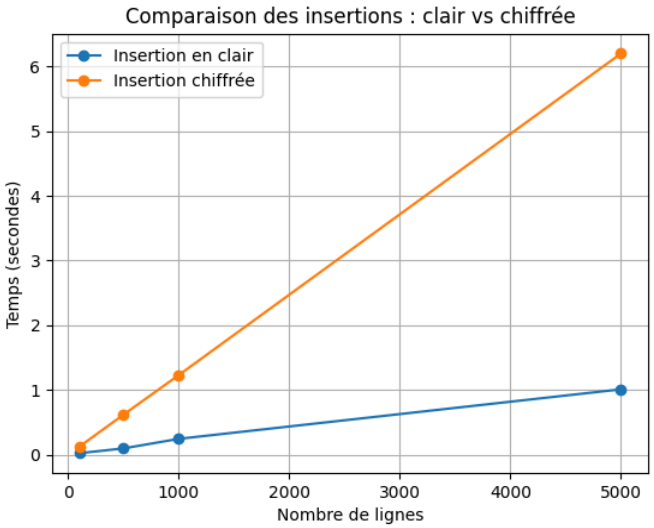
	empno integer	ename character varying (256)	job character varying (256)	hiredate date	sal integer
1	1601	Employee_1	Analyst	2010-07-20	5531
2	1602	Employee_2	Clerk	2013-11-24	5606
3	1603	Employee_3	Engineer	2001-12-23	5399
4	1604	Employee_4	Analyst	2012-06-09	4574
5	1605	Employee_5	Analyst	2010-01-03	4592
6	1606	Employee_6	Clerk	2007-09-14	5558
7	1607	Employee_7	Engineer	2014-11-14	4061

3.2. Evaluation expérimentale

Nous avons mis en place un script Python qui génère des jeux de données simulées, insère les données dans la table EMP (le trigger chiffre automatiquement) puis lit les données via la vue EMP_DECRYPTED (déclenche le déchiffrement) ainsi mesure les temps d'insertion et de lecture pour des volumes de 100 à 10000 enfin exporte les résultats dans un fichier CSV et trace un graphique PNG.

Nous avons évalué les performances à travers plusieurs tests :

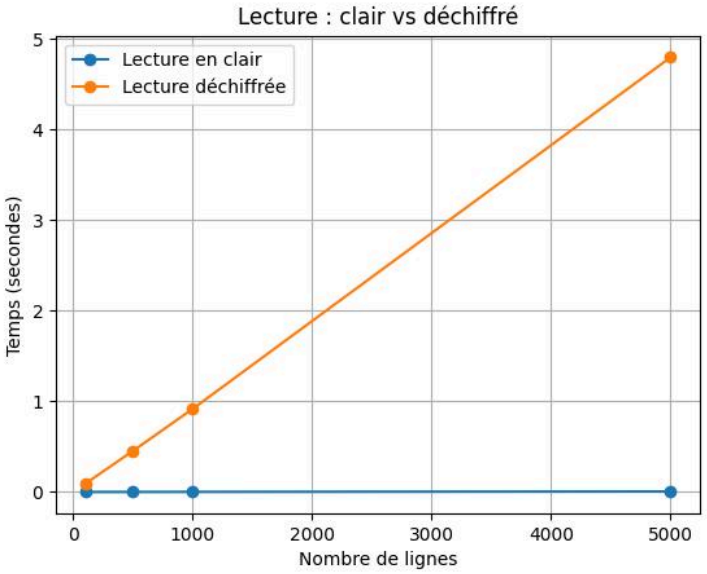
- Insertion claire vs insertion chiffrée :



On constate que AES256 ajoute un surcoût au moment de l'insertion.

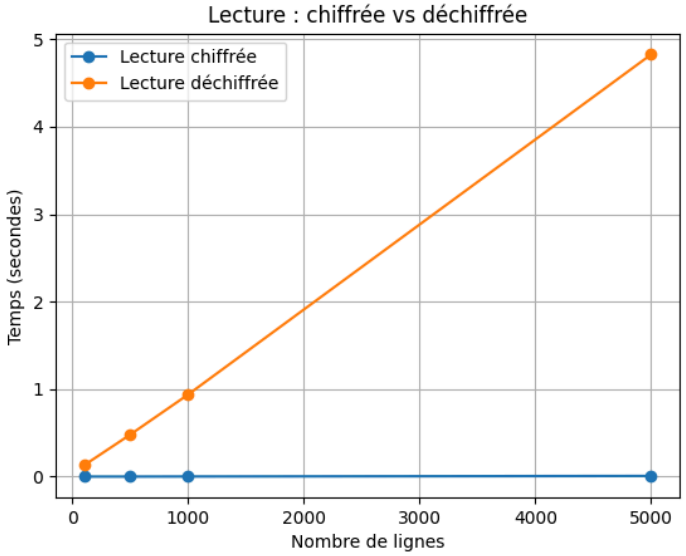
- Lecture claire vs lecture déchiffrée

On constate que le déchiffrement est coûteux, mais la croissance reste linéaire.



- Lecture brute vs lecture déchiffrée

On constate que le déchiffrement est coûteux, mais la croissance reste linéaire.



Lors de nos tests, les temps de lecture brute (chiffrée) et de lecture claire sont très proches, car tous deux consistent en une simple lecture de colonne sans opération lourde.
Le déchiffrement (pgp_sym_decrypt) induit un léger surcoût qui devient significatif uniquement pour de très grands volumes de données.

3.3. Etude de vulnérabilité

Le but de cette expérience est de montrer que l'utilisation du mode ECB pour chiffrer les salaires peut entraîner des failles de sécurité, en rendant possible une attaque par analyse de fréquence.

Nous avons modifié la fonction de chiffrement pour désactiver la compression (compress-algo=0) et la vérification d'intégrité (disable-mdc=1), afin de forcer un comportement similaire à celui du mode ECB.

Nous avons ensuite inséré plusieurs employés avec des salaires identiques ou différents dans une table dédiée (SAL_ECB).

Enfin, nous avons observé les valeurs chiffrées et analysé la fréquence d'apparition des chiffrés.

Les résultats obtenus sont:

- Les salaires identiques ont produit des chiffrés identiques.
- Il a été possible de détecter quels salaires étaient les plus fréquents en comptant les occurrences de chaque chiffré.
- Cette répétition permettrait à un attaquant d'inférer des informations sensibles sans jamais avoir besoin de déchiffrer les données.
- Il est vulnérable aux attaques par analyse de motifs et n'est pas adapté à la protection de données sensibles.

C'est pourquoi l'utilisation de modes comme CBC est fortement recommandée en pratique.

La figure ci-dessous nous montre le résultat de nos tests.

	ename text	sal integer	chiffre text
1	Alice	5000	c30d040903020d8085724298fe7c73d235019d5fd4946fe698e443ed6dc3009cd7b23cce838bddeb0a49b379188dfe10d661e8a46daa4347ffd29c5a93976a5f6a0715bd4...
2	Bob	5000	c30d04090302a05da01d435ca6cf6ad235011110e4e608a268622b1ec5849ea3f112326ed5aee47737fc73e444a99aee71dc5c2087cecff3839a008a750ecf2bb9c8f1c3d...
3	Charlie	5200	c30d04090302b0467cf44fc2366c66d235015dedf71d7c36e7ad5a1af8ef24e71e27b691028413d60aae86450a7e8b878d8f3230bead1d6bcab417b8b0f2c6be397100217...
4	David	5200	c30d0409030223e7af49517b1f4f7fd235017883075c614a56fc03cdad21664725af4e9dea3e142a495fd51f33ee4239de4ee298a2f3bd1ed8cf1ed838f02ea819300e4d6539
5	Alice	5000	c30d040903027304c630f2fda8c668c91eacaeeea23a450b371c2b0791765cceaae968bbec72333db4509f26e92839
6	Bob	5000	c30d04090302ffa984cdc2d3e8e660c91e09b150f726985b73525c3681e5cdfbf498ca2381c1a2882dcd10ba6f56b1
7	Charlie	5200	c30d04090302d7f5741b6064ae286dc91e9b17cd7773b3f6ffd9f6d9d14fa2370993b8e276b145be8dc293af3688c4
8	David	5200	c30d040903029b8900e4657a5d687fc91ea5fd32993c4af4965c98f1f09a342795da617b78a17bec3012c1cf79ce1f

CONCLUSION

Ce projet nous a permis d'explorer en profondeur les enjeux liés à la sécurisation des données sensibles dans une base de données relationnelle.

À travers la mise en œuvre d'un chiffrement symétrique basé sur AES 256 en mode CBC dans PostgreSQL, nous avons démontré qu'il est possible d'assurer la confidentialité des données tout en maintenant des performances compatibles avec un usage en production.