CHIFFREMENT ADAPTATIF AUX REQUETES UTILISATEURS OUTSOURCING

FEHIS SAAD

Ecole Nationale Supérieure d'Informatique (ESI ex INI) Alger
s_fehis@esi.dz

ABSTRACT

Le chiffrement est une technique conçue pour assurer la confidentialité des données sensibles. Il est capable de garantir une haute sécurité des données, mais il peut imposer des coûts substantiels sur la performance d'un système dans la gestion de ces données, traitement CPU (niveau site client) pour le chiffrement/déchiffrement et le filtrage des données, transmission des données (Quantité d'informations) sur le réseau entre le serveur et le client.

Dans cet article, nous avons traité cette problématique dans le cas d'une base de données relationnelle chiffrée et hébergé chez un fournisseur de services. En effet, nous proposons une approche de chiffrement adaptatif aux requêtes des utilisateurs, qui combine le chiffrement et la fragmentation verticale.

KEYWORDS : Fournisseur de services d'hébergements; Outsourcing; base de données; sécurité; confidentialité; chiffrement.

1 INTRODUCTION

La sous-traitance d'hébergement des bases de données est un nouveau paradigme de gestion de données qui commence à émerger. Le propriétaire des données n'est plus responsable sur sa gestion. En effet, une partie de ses données est confiée à des fournisseurs de services d'hébergements externes qui offrent des fonctionnalités de gestion des données. Il est donc nécessaire d'assurer la sécurité des données sensibles chez ces fournisseurs de services.

La sécurité des données sous-traitées est une discipline qui traite les problèmes de sécurité associée à la sous-traitance de la donnée. Parmi ces problèmes, la confidentialité des données (hébergées chez le fournisseur de services) qui est assurée par les techniques de chiffrement.

Le chiffrement des données relationnelles, peut être réalisé à plusieurs niveaux de granularité (valeur d'attribut, ligne, colonne ou page). Le chiffrement au niveau page est ignoré dans notre contexte car il ne permet pas au système de gestion de base de données (SGBD) de manipuler les données relationnelles sans déchiffrement.

L'introduction du chiffrement aux données relationnelles, implique des coûts supplémentaires ajoutés à la manipulation des données en termes de:

Traitement CPU (Machines avec des ressources limitées) au niveau site client (chiffrement / déchiffrement et filtrage des données).

Transmission des données (Quantité d'informations) sur le réseau entre le serveur et le client

Des résultats expérimentaux [1] ont montré que, chiffrer des grands morceaux de données est toujours considérablement plus efficace que chiffrer plusieurs petits morceaux. Le chiffrement doit être donc appliqué au niveau lignes. Dans ce cas, la quantité d'information transportée sur le réseau et le coût de son chiffrement/déchiffrement peuvent importantes. En effet la ligne est chiffrée dans une seule entité qui représente le format chiffrée de la ligne complet de la table.

Pour minimiser ces coûts, on doit éviter le transport des attributs non concernés par la requête. En effet les attributs provoqués par la sélection sont chiffrés avec d'autres attributs d'une même table qui ne sont pas concernés par la partie sélection dans une même requête, par conséquent le choix d'une granularité de chiffrement est très important pour l'optimisation du coût du processus d'exécution d'une requête.

Le type de chiffrement au niveau champs est le chiffrement le plus flexible pour les requêtes de sélection d'un sous ensemble de rubriques d'une même table, ainsi que pour le contrôle d'accès aux données.

Cependant le chiffrement au niveau champs, augmente la taille de l'espace disque d'une manière considérable et surtout dans le cas où le type de chiffrement est par bloc et la taille de bloc de chiffrement est très grande que la taille des champs concerné par cette opération de chiffrement. Sachant que dans ce cas, on doit impérativement remplir le

champ par d'autres bits pour atteindre la taille du bloc de chiffrement.

Dans cet article, nous proposons un chiffrement adaptatif aux requêtes utilisateur, minimisant la quantité d'information transportée sur le réseau, par la combinaison du chiffrement et les techniques de fragmentation verticale des tables relationnelles dans une base de données chiffrée et hébergée chez un fournisseur de service.

Cet article est organisé en plusieurs sections:

La section (II), décrit un état de l'art sur les modèles de stockage des tables relationnelles chez le fournisseur de service, le processus d'exécution des requêtes par le fournisseur de service sans déchiffrements des données, l'impact du modèle de stockage sur le processus de traitement des requêtes et la fragmentation verticale des tables relationnelles.

La section (III) décrit notre contribution, qui est le chiffrement adaptatif aux requêtes utilisateurs par la combinaison de la contrainte taille de bloc de chiffrement avec la taille des fragments générés. La section (IV), décrit la simulation pour évaluer notre approche et une conclusion.

2 ETAT DE L'ART

Le chiffrement est la technique utilisée pour assurer la confidentialité des données hébergées chez le fournisseur de services. Mais le problème qui se pose est comment manipuler ces données chiffrées par le serveur du fournisseur de service sans divulguer son contenu, et ceci pour répondu aux requêtes des utilisateurs clients.

Pour palier au problème de manipulation des données relationnelles chiffrées, la solution, est de stocker avec la ligne chiffrée de la table relationnelle des rubriques auxiliaires [2], [3], [4] et ceci pour permettre l'exécution des opérations de comparaison et arithmétiques sur des données chiffrées, dans les requêtes par le serveur sans divulguer la confidentialité des données.

Ce modèle de stockage permet de représenter toute relation $R = (r_1, r_2, \ldots, r_n)$ au niveau serveur sous son format chiffré:

$R^S = \left(etuple, P_1^{id}, \ldots, P_m^{id}, F_1^f, \ldots, F_k^f, W_1^a, \ldots, W_t^a, A_1^b, \ldots, A_j^b\right)$, avec :

$etuple = < E^t(r_1, r_2, \ldots, r_n) >$, où E^t est la fonction utilisée pour chiffrer une ligne de la relation R.

A_j: Attributs pour réaliser des opérations d'agrégation sur R.

W_t : Attributs de type texte pour les requêtes de type LIKE.

F_k : Attributs cryptés au niveau du champ pour les requêtes nécessitent des opérations d'égalité, d'équi-jointure et le groupement.

P_m : Attributs de partitionnement, pour les requêtes de sélection ou de jointure autres que l'égalité.

Ce modèle de stockage des données relationnelles chiffrées et hébergées chez un ou plusieurs fournisseurs de services est étroitement lié au processus d'exécution des requêtes.

Le processus d'exécution d'une requête client par le serveur sans déchiffrement des données par ce dernier, est donné par la figure 1.

Le client saisi une requête via une interface client. Cette requête est soumise à une transformation pour qu'elle puisse être exécutée sous le format chiffré de la base de données. Cette nouvelle forme de requête est envoyée vers le serveur, lequel l'exécuté et renvoi les résultats chiffrés vers le client. Ce dernier filtré les résultats avant les afficher. L'exécution complète de la requête nécessite parfois une interaction entre le client et le serveur.

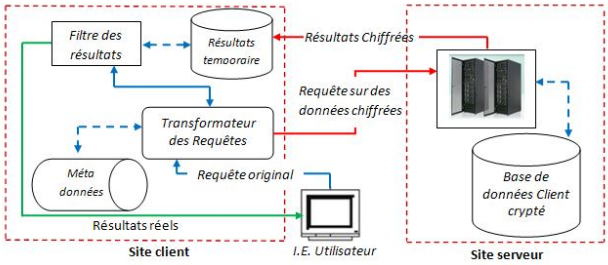

Figure 1: Processus d'exécution d'une requête [5]

Le modèle de stockage a un impact important sur le processus de traitement des requêtes, en termes de coûts de transmission des données sur le réseau (quantité d'informations) et en termes de déchiffrement et de filtrage des données (CPU) au niveau du site client. En effet, la quantité d'information transportée sur le réseau et le coût de son chiffrement/déchiffrement peut être importante; car ces informations contiennent au moins le champ **etuple** (sauf dans les opérations d'agrégation qui contiennent des cas particuliers) qui représente le format chiffrée de la ligne complète de la table.

La solution à ce problème est la fragmentation verticale des données. La fragmentation verticale est un processus qui permet de partitionner les attributs d'une table relationnelle en groupes selon certains critères. De nombreux algorithmes pour la fragmentation verticale sont basés sur l'affinité entre attributs. L'affinité entre attributs est représentée par une relation sur la base du fait que les attributs sont utilisés ensemble par les mêmes requêtes.

[6] utilise l'algorithme BEA (Bond Energy Algorithm) pour réaliser la fragmentation verticale. Il consiste à permuter les lignes et les colonnes d'une matrice carrée afin d'obtenir une forme diagonale semi-bloc. L'objectif de l'algorithme est de regrouper les grandes valeurs de la matrice ensemble et les petits valeurs ensemble [7].

Navathe et al. [8] ont amélioré l'approche de BEA en proposant une approche à deux phases pour le partitionnement vertical. Dans un premier temps, ils utilisent une matrice d'utilisation d'attributs (AUM) pour

construire la matrice d'affinité entre attributs (AAM) sur laquelle le regroupement est effectué. Dans la deuxième étape, ils introduisent le facteur coût, qui reflète l'environnement physique de stockage des fragments, pour affiner le schéma de partitionnement.

Cette approche a été appliquée dans [9] en utilisant des facteurs physiques spécifiques, tels que le nombre d'attributs, leurs longueurs, la sélectivité et la cardinalité de la relation.

La plupart des méthodes de fragmentation sont conçues pour les systèmes de bases de données distribuées; où il ya plusieurs serveurs connectés via un réseau et dotés de plusieurs disques et mémoires, ce qui ajoute au problème de partitionnement vertical le problème de déploiement et d'allocation.

Dans ce travail, on s'intéresse plus précisément à la fragmentation verticale pour regrouper les rubriques en plusieurs sous ensembles de rubriques afin de nous permettre d'appliquer un chiffrement au niveau ligne par fragment, où chaque fragment est représenté par une seule colonne dans la table. Tous les problèmes de bases de données distribuées sont donc ignorés dans ce travail.

[8] a proposé une approche indépendante au problème d'allocation qui accepte en entrée des informations sur les requêtes utilisateur et au même temps paramétrable. Ceci nous permet d'introduire la contrainte taille du fragment en fonction de la taille du bloc de chiffrement. Dans ce travail, on propose pour la fragmentation verticale d'introduire donc la contrainte taille de fragment en fonction du bloc de chiffrement.

3 SOLUTION PROPOSÉE

Nous proposons un chiffrement adaptatif aux types de requêtes utilisateur qui consiste à ne pas chiffrer toute la ligne complète (etuple) en une seule unité, mais plutôt chiffrée au niveau ligne par fragment (sous ensemble de rubriques). La décomposition de la table en fragments (groupes de colonnes), est réalisée à partir de l'analyse de l'activité des requêtes.

Nous proposons une fragmentation dont la taille d'une ligne de fragment (la somme des tailles des rubriques qu'il le compose) est un multiple exacte de la taille du bloc de chiffrement (minimum égal à la taille de bloc de chiffrement). Ce qui nous permet d'atteindre la taille du bloc de chiffrement et au même temps minimiser la quantité de remplissage des blocs. Ce qui nous donne la nouvelle forme de $\mathbb{R}^S$ suivante:

$$R^S\left(etuple_1^{Fr=g}, \ldots, etuple_k^{Fr=g}, P_1^{id}, \ldots, P_m^{id}, F_1^f, \ldots, F_k^f, W_1^g, \ldots, W_k^g, A_1^h, \ldots, A_j^h\right)$$

Avec ce nouveau schéma, on peut transporter sur le réseau que les fragments concernés par la requête utilisateur, ce qui minimise le coût (Quantité d'information) du transport réseaux et le temps de réponse.

La réalisation de notre solution nécessite la collecte des informations sur les requêtes utilisateurs pour une période donnée dans le temps et puis l'application d'une méthode pour trouver la meilleure fragmentation de chaque relation concernée par le chiffrement et provoquée par les requêtes.

Notre processus de fragmentation verticale nécessite trois tables en entrée (figure 1):

Une table qu'on appelle une matrice d'utilisation qui contient en ligne des identificateurs de requêtes, et en colonne les identificateurs de rubriques.

Une deuxième table qui contient les fréquences d'exécution des requêtes

Une troisième contenant les tailles des rubriques.

A partir de ces informations en entrée, on construit une autre matrice qu'on l'appelé la matrice d'affinité. Ces informations d'affinités sont utilisées pour déterminer la composition des fragments. En effet, nous lançons l'algorithme de groupement BEA (bond energy algorithm) sur cette matrice d'affinité pour obtenir une forme diagonale semi-bloc (regroupement des grandes valeurs ensemble et les petits valeurs ensemble).

Ensuite, nous appliquons l'algorithme de partitionnement vertical binaire, SPLIT_NON_OVERLAP [8], sur la matrice obtenue par l'algorithme BEA. Ce qui nous donne deux fragments (matrices) non chevauchants. Pour avoir un partitionnement n-aire, nous appliquons cet algorithme d'une façon itérative. A chaque itération, les deux fragments génères sont considérés comme deux objets distincts, et l'algorithme SPLIT_NON_OVERLAP est réappliqué pour chacun des fragments.

Pour l'arrêt de ce processus de partitionnement, nous appliquons la contrainte de taille minimale sur des fragments en fonction de la taille du bloc de chiffrement. Cette contrainte peut empêchée les lignes partitionnées de devenir trop petite, est au même temps minimiser la taille de remplissage du bloc de chiffrement. A la fin de ce processus de partitionnement itératif, nous appliquons un chiffrement au niveau ligne par fragment.

4 EXPÉRIMENTATION

Dans l'objectif de valider notre approche et la démarche de notre solution, nous avons réalisé une expérimentation sur un jeu de données utilisé par plusieurs scientifiques [8], [9] et [10] pour valider leurs approches de résolution de problèmes de fragmentation verticale des tables de base de données relationnelles. Ce jeu de données est constitué d'informations sur les requêtes, leurs utilisations des attributs et les tailles de rubriques.

Nous avons choisi, une taille du bloc de chiffrement de 128 bits (16 octets) et nous avons varié la taille du fragment en fonction de la taille du bloc de chiffrement (1, 2, 3 et 4 fois la taille du bloc de chiffrement). Et ceci pour montrer l'avantage de la combinaison du chiffrement et la fragmentation verticale.

En effet, notre processus de fragmentation verticale nécessite trois tables en entrée (Table I), une table qu'on appelle une matrice d'utilisation qui contient en ligne des

identificateurs de requêtes ($Q_i, avec\ i = 1..15$), et en colonne les identificateurs de rubriques ($A_j, avec\ j = 1..20$) d'une seule table. Une deuxième table qui contient les fréquences d'exécution des requêtes ($Acc_k, avec\ k = 1..15$) et une troisième contenant les tailles des rubriques en octet (Tail Rub).

Table 1: Information en entrée pour la fragmentation [8]

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	
Q1	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	Acc 1= 50
Q2	0	1	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	Acc 2= 50
Q3	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	1	1	0	0	Acc 3= 50
Q4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	Acc 4= 50
Q5	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	Acc 5= 15
Q6	1	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	Acc 6= 15
Q7	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	Acc 7= 15
Q8	0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	Acc 8= 15
Q9	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	Acc 9= 10
Q10	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	Acc 10= 10
Q11	1	1	1	0	1	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	Acc 11= 10
Q12	0	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	1	Acc 12= 10
Q13	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	0	Acc 13= 10
Q14	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	Acc 14= 5
Q15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	Acc 15= 5
Tail Rub :	6	3	4	6	5	8	3	5	9	7	6	8	4	6	7	8	3	5	9	7	

Les résultats de la première étape consistant à construire une matrice d'affinité des attributs sont donnés par la table II.

Table 2: Matrice d'affinité initiale

	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
A1	105	15	15	55	80	65	5	85	25	15	15	10	10	0	0	0	0	10	0	0	
A2	15	90	15	5	25	15	5	5	65	0	10	75	75	75	15	15	0	15	10	15	
A3	15	15	80	5	15	15	70	5	15	65	65	10	10	0	0	0	65	50	0	0	
A4	55	5	5	65	55	55	15	55	5	10	0	0	0	10	0	0	0	0	10	10	
A5	80	25	15	55	90	65	5	70	15	0	10	10	10	10	0	0	0	0	10	0	
A6	65	15	15	55	65	65	5	55	15	0	0	10	10	0	0	0	0	0	0	0	
A7	5	5	70	15	5	5	80	5	5	75	65	0	0	10	0	0	65	50	10	10	
A8	85	5	5	55	70	55	5	95	5	15	25	0	0	0	10	10	10	10	0	0	
A9	25	65	15	5	15	15	5	5	75	0	0	60	60	50	0	0	0	10	0	0	
A10	15	0	65	10	0	0	75	15	0	90	80	0	0	10	0	0	65	50	10	10	
A11	15	10	65	0	10	0	65	25	0	80	100	0	0	10	10	10	75	60	10	0	
A12	10	75	10	0	10	10	0	0	60	0	0	75	75	65	15	15	0	15	0	15	
A13	10	75	10	0	10	10	0	0	60	0	0	75	75	65	15	15	0	15	0	15	
A14	0	75	0	10	10	0	10	0	50	10	10	65	65	85	15	15	0	15	20	25	
A15	0	15	0	0	0	0	0	10	0	0	10	15	15	15	80	80	15	30	55	70	
A16	0	15	0	0	0	0	0	10	0	0	10	15	15	15	80	80	15	30	55	70	
A17	0	0	65	0	0	0	65	10	0	65	75	0	0	0	15	15	80	65	5	5	
A18	10	15	50	0	0	0	50	10	10	50	60	15	15	15	30	30	65	90	5	20	
A19	0	10	0	10	10	0	10	0	0	10	10	0	0	20	55	55	5	5	75	65	
A20	0	15	0	10	0	0	10	0	0	10	0	15	15	25	70	70	5	20	65	80	

Matrice d'Affinité

La deuxième étape, est l'application de l'algorithme de regroupement BEA sur la matrice d'affinité initiale (Table II) qui produit une nouvelle matrice diagonale, donnée par la table III.

A ce niveau, nous lançons l'opération de fragmentation verticale plusieurs fois suivant la taille de fragment choisi en fonction de la taille du bloc de chiffrement (1, 2, 3 ou 4 fois la taille du bloc de chiffrement).

TABLE 3:Matrice d'affinité sous forme diagonale

	A0	A7	A3	A4	A8	A1	A5	A6	A9	A2	A12	A13	A14	A18	A11	A10	A17	A19	A20	A16	A15
A7	80	70	15	5	5	5	5	5	5	0	0	10	50	65	75	65	10	10	0	0	
A3	70	80	5	5	15	15	15	15	15	10	10	0	50	65	65	65	0	0	0	0	
A4	15	5	65	55	55	55	55	5	5	0	0	10	0	0	10	0	10	10	0	0	
A8	5	5	55	95	85	70	55	5	5	0	0	0	10	25	15	10	0	0	10	10	
A1	5	15	55	85	105	80	65	25	15	10	10	0	10	15	15	0	0	0	0	0	
A5	5	15	55	70	80	90	65	15	25	10	10	10	0	10	0	0	10	0	0	0	
A6	5	15	55	55	65	65	65	15	15	10	10	0	0	0	0	0	0	0	0	0	
A9	5	15	5	5	25	15	15	75	65	60	60	50	10	0	0	0	0	0	0	0	
A2	5	15	5	5	15	25	15	65	90	75	75	75	15	10	0	0	10	15	15	15	
A12	0	10	0	0	10	10	10	60	75	75	75	65	15	0	0	0	0	15	15	15	
A13	0	10	0	0	10	10	10	60	75	75	75	65	15	0	0	0	0	15	15	15	
A14	10	0	10	0	0	10	0	50	75	65	65	85	15	10	10	0	20	25	15	15	
A18	50	50	0	10	10	0	0	10	15	15	15	15	90	60	50	65	5	20	30	30	
A11	65	65	0	25	15	10	0	0	10	0	0	10	60	100	80	75	10	0	10	10	
A10	75	65	10	15	15	0	0	0	0	0	0	10	50	80	90	65	10	10	0	0	
A17	65	65	0	10	0	0	0	0	0	0	0	0	65	75	65	80	5	5	15	15	
A19	10	0	10	0	0	10	0	0	10	0	0	20	5	10	10	5	75	65	55	55	
A20	10	0	10	0	0	0	0	0	15	15	15	25	20	0	10	5	65	80	70	70	
A16	0	0	0	10	0	0	0	0	15	15	15	15	30	10	0	15	55	70	80	80	
A15	0	0	0	10	0	0	0	0	15	15	15	15	30	10	0	15	55	70	80	80	

En effet, dans cette article nous avons choisi comme taille de fragment égale à trois fois la taille du bloc de chiffrement, et cela pour montrer l'avantage de la combinaison de la fragmentation verticale et la technique de chiffrement dans la minimisation des différents coûts de traitement des requêtes. En effet, l'application répétitive de l'algorithme SPLIT_NON_OVERLAP (plusieurs itérations) nous donne le résultat dans la figure 2 suivante :

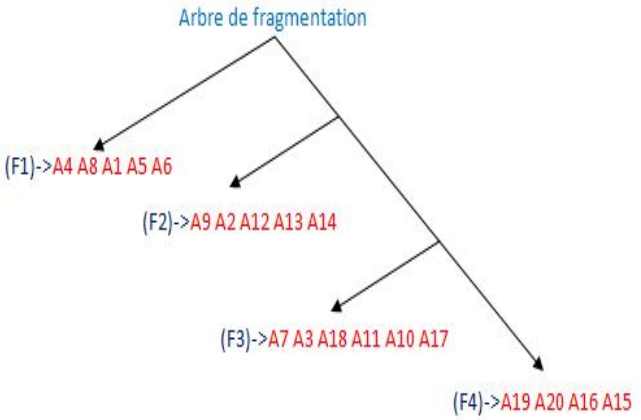

Figure 2: Application répétitif de SPLIT_NON_OVERLAP

En suite, nous avons réalisé un ensemble de comparaisons des coûts des requêtes avec fragmentation (TF_Enc) et sans fragmentation (SF_Enc). En effet, les résultats de comparaison dans la table IV (SF%Frag) et présenter sous forme d'histogramme (figure 3) qui montre que notre approche optimise les coûts des requêtes utilisateurs.

Figure 3:Histogramme de Comparaison des coûts des requêtes avec et sans fragmentation (3 X 128 bits)

Table 4 : Fragmentation avec une taille de fragment égale à trois fois la taille du bloc de chiffrement (3 x 128 bits)

Requêtes	F1	F2	F3	F4	TF_Enc	SF_Enc	SF%Frag
Q1	1	0	0	0	32	128	25%
Q2	0	1	0	0	32	128	25%
Q3	0	0	1	0	32	128	25%
Q4	0	0	0	1	32	128	25%
Q5	1	0	0	0	32	128	25%
Q6	1	0	1	0	64	128	50%
Q7	0	0	1	0	32	128	25%
Q8	0	1	1	1	96	128	75%
Q9	1	1	1	1	128	128	100%
Q10	1	1	1	0	96	128	75%
Q11	1	1	1	0	96	128	75%
Q12	1	1	1	1	128	128	100%
Q13	1	0	1	1	96	128	75%
Q14	1	1	1	0	96	128	75%
Q15	0	0	1	1	64	128	50%
T_Frag:	30	30	28	31			
T_Encr:	32	32	32	32			
Rempli:	2	2	4	1			

Table 4:Comparaison des coûts des requêtes avec des différentes tailles de fragmentation

Requêtes	Taille			
	1 x 128 bits	2 x 128 bits	3 x 128 bits	4 x 128 bits
Q1	48	32	32	32
Q2	64	32	32	32
Q3	64	32	32	64
Q4	48	32	32	64
Q5	16	32	32	32
Q6	48	64	64	96
Q7	48	32	32	64
Q8	112	96	96	96
Q9	80	128	128	128
Q10	48	96	96	128
Q11	96	96	96	128
Q12	80	128	128	128
Q13	80	96	96	96
Q14	80	96	96	128
Q15	80	64	64	64

Par exemple, la requête Q1 à besoin des attributs A1, A4, A5, A6 et A8 (Voir table I), ces attributs appartiennent à un seul fragment F1 (Figure 2). Si la fragmentation est appliquée, alors les fragments sont chiffrés séparément, et puisque cette requête a besoin seulement d'un seul fragment F1 (Table IV), donc la quantité d'information transportée sur le réseau pour une seule ligne est de 32 octets.

Dans le cas contraire, tous les attributs de A1 à A20 sont chiffrés ensemble, dans ce cas, toutes les rubriques sont transportées ensemble sous format chiffré ce qui implique que la quantité d'information transportée sur le réseau pour une seul ligne est de 128 octets. A partir de ces données on remarque que la quantité d'information transportée sur le réseau avec fragmentation pour une seule ligne est inferieure à celle sans fragmentation (Figure 3).

A partir de l'analyse et de la comparaison des coûts des requêtes (Quantité transportée sur le réseau tableau V et figure 4) et les coûts de stockage des données chiffrées (Taille en Octet tableau VI) avec l'application de la fragmentation verticale, nous constatons que le bon choix de la taille de fragment en fonction (Multiple) du bloc de chiffrement est très important dans la minimisation de ces coûts.

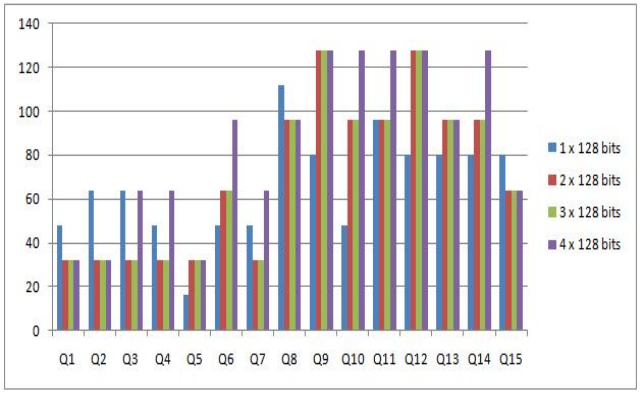

Figure 4: Histogramme de Comparaison des coûts des requêtes avec différentes tailles de fragmentation

Table 6 : Comparaison des coûts (Taille en Octet) de stockage avec différentes tailles de fragmentation

Taille de fragment	Taille de la ligne Sans chiffrement	Taille de la ligne chiffrée	Nombre de fragments	Total de Tailles des fragments avec chiffrement	Taille de remplissage
1 x 16	119	128	14	14 x 16 = 224	105
2 x 16	119	128	4	4 x 32 = 128	9
3 x 16	119	128	4	4 x 32 = 128	9
4 x 16	119	128	3	32 + 32 + 64 = 128	9

En effet, valider une approche est une phase importante dans tout travail de recherche. A travers ces expérimentations, nous avons montré par une étude de cas que la combinaison du chiffrement et les techniques de fragmentation verticale des tables relationnelles, minimise Le coût (Quantité d'information) de transport sur le réseau, le coût de chiffrement / déchiffrement et le filtrage des données.

5 CONCLUSION

Dans cet article nous avons proposé un chiffrement adaptatif aux requêtes utilisateur, qui minimise la quantité d'information transportée dans le réseau par la combinaison du chiffrement et les techniques de fragmentation verticale des tables relationnelles dans une base de données chiffrée et hébergée chez un fournisseur de service.

Notre démarche est l'analyse des requêtes utilisateurs et puis utilisation des techniques de fragmentation verticale des tables avec la contrainte de la taille des fragments en fonction de la taille de bloc de chiffrement.

REFERENCES

[1] B. Iyer1, S. Mehrotra2, E. Mykletun, G. Tsudik, and Y. Wu "A Framework for Efficient Storage" Security in RDBMS, Advances in database technology - EDBT 2004 pp. 147-164 (Heraklion, 14-18 March 2004)

[2] H. Hacigumus, B. Iyer, and S. Mehrotra, "Providing database as a service" in International Conference on Data Engineering, pp. 0029 © 2002 IEEE.

[3] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service provider model" in ACM SIGMOD Conference on Management of Data, pp. 216-227, ACM Press, June 2002.

[4] H. Hacigumus, B. Iyer, and S. Mehrotra, ."Efficient execution of aggregation queries over encrypted databases" in International Conference on Database Systems for Advanced Applications (DASFAA), 2004. pp.125-136

[5] H. Hacigumus, B. Iyer, and S. Mehrotra, "Managing and Querying Encrypted Data" Handbook of Database Security Applications and Trends, © 2008 Springer Science+Business Media, LLC (pp172-199).

[6] S. Chakravarthy, J. Muthuraj, R. Varadarajan, and S. Navathe, "An objective function for vertically partitioning relations in distributed databases and its analysis" Distributed and Parallel Databases 2 © Springer (2004) pp. 183-207.

[7] W.T. MCCORMICK, P.J. SCHWEITZER, AND T.W WHITE,"Problem decomposition and data reorganization by a clustering technique". Oper. Res. 20,5 (Sept. 1972), pp. 993-1009.

[8] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dour, "Vertical partitioning algorithms for database design" ACM TODS, vol. 9, no. 4, pp. 680-710, 1984.

[9] D. Cornell and P. Yu, "A vertical partitioning algorithm for relational databases" in International Conference on Data Engineering, Los Angeles, California, 1987, pp. 30-35.

[10] S. B. Navathe and M. Ra, "Vertical partitioning for database design: A graphical algorithm" SIGMOD Record, vol. 14, no. 4, pp. 440-450, 1989.