@lgoritma
Jurnal Ilmiah STMIK GI MDP
Volume 2 Nomor 3, Oktober 2006

Implementasi Enkripsi Database Menggunakan *Transparent Data Encryption* Pada *Database Engine* Oracle

Antonius Wahyu Sudrajat

STMIK MDP Palembang
wayoe@stmik-mdp.net

Abstrak: Perkembangan jaringan komunikasi data secara global memberikan tantangan-tantangan di dalam keamanan akan data dan informasi. Berbagi cara dilakukan untuk menjaga kerahasiaan data, baik dari tingkat aplikasi maupun *database*. Terdapat tiga tipe dasar dari enkripsi, yaitu *manual*, *semi-transparent*, dan *transparent*. *Transparent Data Encryption* adalah salah satu jenis enkripsi yang mempunyai keuntungan dalam kemudahan dari sisi *user* karena *user* tidak perlu melakukan proses manajemen enkripsi dan dekripsi data secara langsung karena *database* telah melakukannya. Penelitian ini akan membahas cara kerja *Transparent Data Encryption*, keunggulannya, serta implementasinya dalam enkripsi *database*. Dari pembahasan ini dapat diketahui keunggulan *TDE*, seperti tidak adanya penurunan kinerja akibat penerapan enkripsi data dan kemudahan yang ditawarkan kepada pengguna.

Kata Kunci : Enkripsi, *Wallet*, Dekripsi, *view*, *trigger*.

1 PENDAHULUAN

Keamanan dan kerahasiaan data merupakan salah satu aspek penting dari suatu sistem informasi. Sebuah informasi hanya ditujukan bagi segolongan tertentu, hal tersebut terkait dengan bagaimana informasi tidak dapat diakses oleh orang yang tidak berhak. Oleh karena itu sangat penting untuk mencegah jatuhnya informasi kepada pihak-pihak lain yang tidak berkepentingan. Untuk melaksanakan tujuan tersebut maka dirancang suatu sistem keamanan yang berfungsi melindungi informasi dari berbagai ancaman. Untuk menjaga kerahasiaan data banyak organisasi atau perusahaan melakukan dengan mekanisme keamanan tradisional, yaitu:

1. Autentikasi *user* untuk mengenali *user*.
2. Kontrol akses yang *granular* untuk membatasi apa yang *user* bisa lihat dan lakukan.
3. *Auditing* untuk akuntabilitas.
4. Enkripsi jaringan untuk melindungi kerahasiaan data yang sensitif pada saat transmisi data

Dengan demikian dapat dikatakan bahwa enkripsi adalah komponen penting dari solusi-solusi di atas. Sebagai contoh, *Secure Sockets Layer* (SSL), sebuah protokol enkripsi jaringan berstandar Internet dan autentikasi, menggunakan enkripsi untuk meng-auntentifikasi user, melalui X.509 *digital certificate*.

Meskipun enkripsi bukanlah satu-satunya solusi untuk keamanan informasi, tetapi enkripsi merupakan *tool* yang digunakan untuk menangani ancaman-ancaman keamanan yang spesifik dan akan menjadi semakin penting dengan meningkatnya kebutuhan akan data dan informasi yang terhubung dengan sebuah jaringan. Sebagai contoh, nomer kartu kredit biasanya dilindungi ketika dikirim ke suatu situs web dengan menggunakan SSL, nomer kartu kredit biasanya disimpan pada keadaan *clear* (tidak terenkripsi), sehingga rentan kepada siapa saja yang bisa menerobos ke *host* dan memperoleh akses di *database*. Dengan demikian enkripsi pada *database* merupakan hal yang penting dan harus dijaga dari berbagai ancaman-ancaman keamanan.

2 TINJAUAN PUSTAKA

Kriptografi (*cryptography*) merupakan seni dan ilmu untuk menjaga pesan agar tetap aman. Kriptografi secara bahasa berasal dari Yunani yaitu kata *crypto* yang artinya rahasia dan *graphy* artinya

menulis. Jadi, kriptografi adalah suatu ilmu yang mempelajari penulisan secara rahasia. Kriptografi merupakan bagian dari suatu cabang ilmu matematika yang disebut *Cryptologi*. Kritografi bertujuan menjaga data dan informasi tersebut agar tidak dapat diketahui oleh pihak yang tak sah. Sebuah algoritma kriptografik (*cryptographic algorithm*) disebut *chiper* yang merupakan persamaan matematika yang digunakan untuk proses enkripsi dan dekripsi. Proses yang digunakan untuk mengamankan sebuah pesan (*plaintext*) menjadi pesan yang tersembunyi (*ciphertext*) adalah enkripsi (*encryption*). *Ciphertext* adalah pesan yang sudah tidak dapat dibaca dengan mudah. Sedangkan proses mengubah *ciphertext* menjadi *plaintext* disebut dekripsi (*decryption*). Tiga tipe dasar dari enkripsi bisa dibagi menjadi : *manual, semi-transparent,* dan *transparent*.

1) Enkripsi Manual, merupakan enkripsi yang dilakukan sepenuhnya oleh *user* (melalui software tertentu), dimana user harus secara manual memilih objek yang akan dienkripsi dan kemudian menjalankan *command* khusus untuk melakukan enkripsi atau dekripsi sebuah objek.
2) Enkripsi *Semi-Transparent*, merupakan enkripsi *On the fly*, dimana operasi *read/write* tidak secara permanen, sehingga dilakukan sebelum dan sesudah akses dilakukan.
3) Enkripsi *Transparent*, merupakan kebalikan dari enkripsi manual, dimana enkripsi dan dekripsi dilakukan pada *level* yang rendah (*low level*), secara permanen, saat melakukan operasi *read/write* sehingga data yang dienkripsi selalu disimpan dalam bentuk enkripsi.

Pada awal tahun 1970an, algoritma *Data Encryption Standard* diperkenalkan, yang menggunakan *key* 56 bit untuk mengenkripsi dan mendekripsikan informasi. DES membagi setiap pesan dalam blok-blok dan meng-*encode* setiap blok satu pada setiap waktu. DES diadopsi sebagai algoritma yang diakui untuk penggunaan US Federal, tetapi tidak lagi dianggap cukup aman karena sebuah *key* 56-bit bisa dibuka secara paksa dalam waktu yang relatif cepat. DES kemudian diganti oleh *Advanced Encryption Standard* (AES), menggunakan algoritma Rijndael. AES beroperasi dengan *key* 128,192, atau 256 bit.

Pada skema kriptografi *public key*, setiap pemakai mempunyai satu pasang *key* : satu *private* dan satu lagi *public*. *Public key* tidak bersifat rahasia dan biasanya disediakan kepada semua orang yang ingin mengirim sebuah pesan yang terenkripsi kepada pemilik *key*. Pengirim menggunakan *public key* untuk mengenkripsi sebuah pesan dan penerima (pemilik kedua *key*) kemudian menggunakan *private key* untuk mendekripsi pesan yang masuk. Hanya *private key* yang cocok yang bisa membuka pesan yang diamankan dengan *public key*.

2.1 *Transparent Data Encryption*

Transparent Data Encryption menyediakan kriptografi yang transparan pada *user* yang sah tetapi tidak pada penyusup dari luar maupun dari dalam (untuk lebih mudahnya cukup disebut dengan *attacker* saja). Enkripsi ini digunakan untuk kasus apabila terjadi pencurian *hardware* atau media *backup* atau *unathorized access* pada data yang sensitif di level sistem operasi. Salah satu untuk mengatasi pencurian media adalah mengenkripsi data yang sensitif di dalam *database* dan menyimpan *encryption key* nya di lokasi yang terpisah. Tetapi harus dipertimbangkan keseimbangan antara dua konsep yang bertentangan : kemudahan dimana aplikasi bisa mengakses *encryption keys* dan keamanan yang diperlukan bila terjadi pencurian *key*.

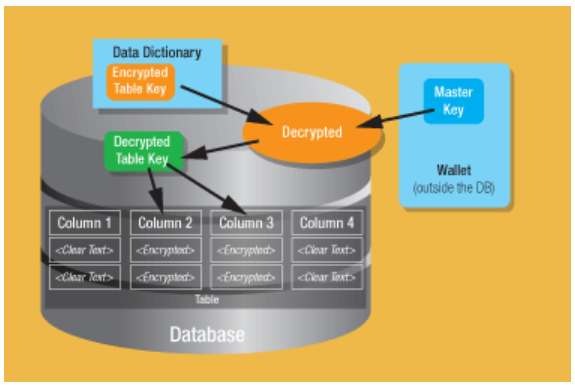
Transparent Data Encryption membuat proses enkripsi sederhana dengan meletakkan enkripsi di dalam *database* nya sendiri. Yang lebih penting lagi, aplikasi bisa melanjutkan pekerjaannya tanpa menggunakan *database trigger*, *view*, dan aplikasi lain yang digunakan solusi enkripsi *database* tradisional. Data secara otomatis dienkripsi ketika ditulis pada *file database* di disk. Data secara otomatis di dekripsi untuk semua *database user* setelah mereka di autentikasi pada *database* dan melewati semua pemeriksaan autentikasi tambahan. Pemeriksaan ini termasuk memastikan *user* mempunyai hak untuk melakukan perintah *select* dan *update* pada table aplikasi.

Perlu diingat, keamanan pada sistem ini tidak hanya tergantung pada sistem kriptonya dan pemyimpanan *key* saja, tetapi juga seberapa baik *user* yang sah bisa dibedakan dengan *attacker*.

2.2 Mekanisme *Transparent Data Encryption*

Enkripsi membutuhkan kita untuk melakukan suatu algoritma enkripsi dan sebuah *encryption key* pada data input berupa *clear-text*. Dan untuk mendekripsi suatu nilai yang telah dienkripsi, kita harus tahu nilai dari algoritma dan *key* yang sama. Selain itu, untuk melakukan enkripsi pada *database*, kita harus membuat suatu infrastruktur enkripsi. Tetapi dengan menggunakan transparent data encryption, kita hanya melakukan suatu proses pendefinisian sebuah kolom yang ingin dienkripsi, setelah itu sistem *database* membuat sebuah *encryption key* yang aman secara kriptografikal untuk table yang berisi kolum tersebut dan mengenkripsi data input *clear-text* yang berada di dalamnya, menggunakan suatu algoritma enkripsi yang ingin kita gunakan. Menjaga *table key* ini sangat penting: sistem *database* mengenkripsinya menggunakan sebuah *master key* dan menyimpannya disebuah lokasi yang aman, yang disebut dengan *wallet*, yang bisa berupa sebuah file pada *database server*. *Table key* yang telah dienkripsi tersebut diletakkan di sebuah *data dictionary*. Ketika seorang *user* memasukkan data ke sebuah kolom yang didefinisikan sebagai terenkripsi, sistem *database* mengambil *master key* dari *wallet*, mendekripsi *encryption key* untuk table tersebut dari *data dictionary*, menggunakan *encryption key* pada nilai input dan menyimpan data yang dienkripsi pada *database* seperti tampak pada gambar 1.

Kita bisa mengenripsi salah satu atau semua kolom pada tabel. Jika sebuah table mempunyai 4 kolom seperti gambar di atas, dan kolom 2 dan 3 dienkripsi, sistem *database* akan men-*generate* sebuah *table key* yang dienkripsi dan menggunakannya untuk mengenkripsi kolom-kolom tersebut. Pada disk, kolom 1 dan 4 disimpan sebagai *clear text* dan nilai pada 2 kolom lainnya disimpan dalam format terenkripsi. Karena data disimpan dalam keadaan terenkripsi, semua komponen *downstream*, seperti *backup log* dan *archieve log*



Gambar 1: Mekanisme *Transparan Data Encryption* (TDE)

juga dalam format terenkripsi. Ketika *user* memilih kolom yang terenkripsi, sistem *database* secara transparan mengambil *table key* yang terenkripsi dari *data dictionary*, lalu mengambil *master key* dari *wallet*, dan mendekripsi *table key*. Kemudian *database* mendekripsi data yang terenkripsi pada disk dan mengembalikan nilai *clear text* pada *user*.

Dengan data yang terenkripsi ini, jika data dalam disk dicuri, maka data yang terenkripsi ini tidak bisa diambil tanpa *master key*, yang berada di dalam *wallet* dan bukan bagian dari data yang dicuri. Bahkan jika *wallet* nya dicuri, *master key* nya tidak bisa diambil dari *wallet* tanpa *wallet password*. Oleh karena itu, pencurinya tidak bisa mendekripsikan data, bahkan jika dia mencuri disk nya atau melakukan *copy* pada *file* data.

Sampai sekarang ini satu-satunya *database* yang mendukung *Transparent Data Encryption* adalah Oracle 10g Release 2, maka pada penelitian ini menggunakan sistem *database* tersebut untuk melakukan implementasi dan analisis.

3 PEMBAHASAN

Tipe enkripsi yang digunakan dalam penelitian ini adalah Enkripsi transparent dimana proses enkripsi dan dekripsi dilakukan pada tingkat yang rendah ((*low level*) secara permanen. Proses enkripsi dan dekripsi dilakukan ketika melakukan proses *read/write database*.

3.1 *Transparent Data Encryption* Pada *Database* Oracle

Langkah pertama yang harus kita lakukan yaitu memilih lokasi dari *wallet*, membuat password *wallet*, dan membuka *wallet*.

3.1.1 Memilih Lokasi *Wallet*

Wallet digunakan untuk menyimpan *master key*. Secara *default*, *wallet* dibuat dalam *directory*$ORACLE_BASE/admin/$ORACLE_SID/ walletJadi, jika $ORACLE_BASE adalah /u01/app/oracle dan $ORACLE_SID adalah SWBT4, maka *wallet* akan disimpan di dalam *directory* /u01/app/oracle/admin/SWBT4/wallet. Kita juga bisa memilih directory yang berbeda dengan mengubahnya di file sqlnet.ora yang terletak di $ORACLE_HOME/network/admin directory. Misalkan, jika kita ingin *wallet* nya berada di direktori /orawall, masukkan baris-baris berikut di file sqlnet.ora:

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE=  (METHOD=file)
   (METHOD_DATA=DIRECTORY=/orawall)))
```

Pada penelitian ini, lokasi penyimpanan *master key* pada direktori *default.*

3.1.2 Membuat *Wallet*

Untuk melakukan hal ini, lakukan perintah di bawah ini sebagai *user* dengan *previlege alter system*:

```
alter system set encryption key
authenticated by "wahyu";
```

Perintah diatas akan melakukan:

a. Membuat *wallet* di lokasi yang telah ditentukan.
b. Membuat password *wallet* sebagai "wahyu"
c. Membuka *wallet* untuk Transparent Data Encryption untuk menyimpan dan mengambil kembali *master key.*

Passwordnya bersifat *case*-sensitive dan harus ditutup dengan tanda kutip ganda. Password "wahyu" tidak akan muncul dalam *clear* text di *dynamic performance view* atau log.

3.1.3 Membuka *Wallet*

Wallet hanya dibuat sekali saja, sehingga *wallet* harus dibuka secara eksplisit setalah *instance database* dimulai. Ketika membuat *wallet*, berarti juga membuka *wallet* untuk beroperasi. Sebelum mengakses *database*, aktifkan *wallet* dan *password* yang sama ketika membuat *wallet*, seperti di bawah ini :

```
alter system set encryption wallet open authenticated
by "wahyu";
```

Tutup *wallet* dengan cara :

```
alter system set encryption wallet close;
```

akses *database* tetap dapat dilakukan meski *Wallet* dalam kondisi tertutup, namun akses hanya dapat dilakukan pada kolom yang tidak dienkripsi, tetapi tidak kolom yang terenkripsi.

3.2 Implementasi *Transparent Data Encryption*

Pada penelitian ini akan di implementasiakan enkripsi menggunakan *Transparent Data Encryption* pada tabel pembayaran dengan struktur seperti pada tabel 1.

Tabel 1. Struktur Tabel Pembayaran

Nama Field	Tipe	Ukuran	Keterangan
no_pesan	varchar	6	Nomor pemesanan produk
nama	varchar	36	Nama pemegang kartu *c*redit *card*
jenis_kartu	varchar	12	Jenis kartu (Visa/*Master Card*)
Nomor_kartu	number	16	Nomor kartu kredit
expire_date	date	-	Tanggal kadaluarsa

Sintaks membuat tabel pada oracle:

```
SQL> create table pembayaran(
       No_pesan varchar (6) not null,
       Nama varchar (36) not null,
```

```
        Jenis_kartu varchar (12) not null,
        Nomor_kartu number (16) encrypt,
        Expire_date date (6) not null);
```

Menampilkan hasil pembuatan tabel

```
SQL> desc pembayaran

Name          Null          Type
----------  ----------   ----------
No_pesan                  varchar (6)
Nama                      varchar (36)
Jenis_kartu               varchar (12)
Nomor_kartu               number (16)
Expire_date               date
```

Mengisi data kedalam tabel pembayaran

```
SQL>  insert  into  pembayaran  values
      ('001','Schmid','visa','544695970881
      2985','23 march 2007');
SQL>  insert  into  pembayaran  values
      ('002','Mark','visa','45569887082369
      02','20 feb 2007');
```

Dari tabel pembayaran dilakukan enkripsi pada kolom nomor_kartu dengan cara:

```
SQL> alter pembayaran modify (nomor_kartu
encrypt);
```

Algoritma enkripsi yang digunakan adalah AES 128-bit, yaitu:

```
SQL> alter pembayaran modify (nomor_kartu
encrypt using 'AES128');
```

Untuk mengakhiri proses enkripsi sehingga dapat mengakses *database* perintah yang digunakan adalah

```
SQL> alter pembayaran modify (nomor_kartu
decrypt);
```

3.2.1 Pertimbangan Performa

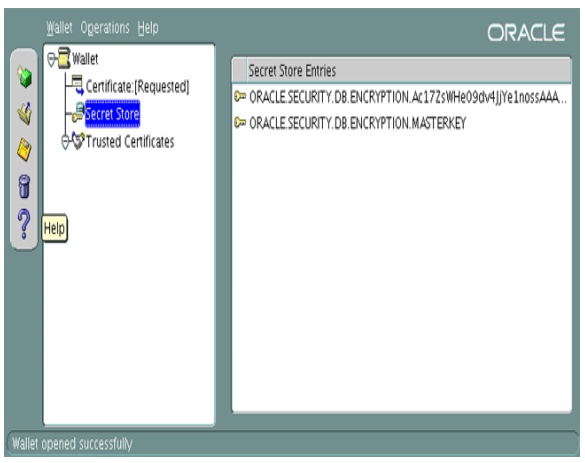
Berdasarkan hasil implementasi tabel pembayaran, proses enkripsi dan dekripsi menghabiskan siklus CPU, sehingga mengakibatkan menurunnya performa. Penurunan performa juga terjadi ketika mengakses kolom yang tidak dienkripsi pada table tanpa *Transparent Data Encryption*.

3.3 Manajemen *Key* dan *Password*

Perubahan pada *table key* dimungkinkan pada saat-saat tertentu, yaitu dengan perintah:

```
SQL> alter table pembayaran rekey using
     'AES255';
```

Sedangkan penggantian password pada wallet dapat dilakukan melalui Oracle Wallet Manager (gambar 2), tetapi tidak mengganti *key* nya.



Gambar 2: Oracle Wallet Manager

3 SIMPULAN

Berdasarkan hasil implementasi, maka dapat disimpulkan bahwa:

1. Kecilnya penalti *performance* ketika melakukan enkripsi dan dekripsi karena pada saat melakukan proses-proses ini, *Transparent Data Encryption* tidak mengubah tipe data dan panjang data.
2. *Transparent Data Encryption* menawarkan kemudahan dari sisi pengguna, sehingga pengguna bisa langsung melakukan proses enkripsi data tanpa melakukan *coding* dan kompleksitas manajemen *key*.
3. Dengan menggunakan *Transparent Data Encryption*, aplikasi bisa melanjutkan pekerjaan tanpa adanya *database trigger*, *view* dan perubahan aplikasi lainnya yang diasosiasikan dengan solusi enkripsi database tradisional.

4 SARAN

1. Kelemahan *Transparent Data Encryption* adalah TDEMasterkey disimpan dalam keadaan tidak dienkripsi pada SGA (memori dari database.
2. *Transparent Data Encryption* tidak dapat *mobile* atau sangat sulit untuk dapat diimplementasikan secara sempurna.

DAFTAR PUSTAKA

[1] Fraase, Michael, "Crypthography", http://www.eclipse.org. Diakses pada 25/06/2006,

[2] Kornbrust, Alexander, "Circumvent Oracle's Database Encryption and Reverse Engineering of Oracle Key Management Algorithms". Red-Database Security, www.oracle.com/technology/deploy/security/pdf/cpu-jan-2005_advisory.pdf. Diakses pada 03/02/2005.

[3] Kurniawan, Yusuf, 2004, *Kriptografi Keamanan Internet dan Jaringan Komunikasi*. Bandung: Informatika Bandung.

[4] Rahardjo, Budi, 2001, *Keamanan Sistem Informasi Berbasis Internet*. Bandung: PT Insan Komunikasi/Indonesia.

[5] ----------, "Encryption of Data At Rest – Database Encryption-", www.appsecinc.com /presentations/Encryption_of_Data_at_Rest. pdf. Diakses pada 05/09/2006.

[6] ----------, "Oracle Advanced Security", www.mid.main.vsu.ru/docs/oracle/network.816/a76932.pdf. Diakses pada 18/09/2006.

[7] ----------, "Transparent Data Encryption Technology, http://www.oracle.com/technology/products/database/oracle10g/pdf/twp_general_10gdb_product_family.pdf. Diak-ses pada 05/09/2006.