

Strategie per la cifratura dei dati sensibili nei database relazionali

Stefano Bendandi (info@stefanobendandi.com)

ICT & Security Consultant

Security+ Certified Professional

Versione 1.0 (novembre 2005)

Nell'era in cui viviamo l'accesso alle informazioni non viene più percepito come una semplice opportunità ma costituisce, semmai, un vero e proprio requisito necessario a supportare i vari processi decisionali e l'espletamento di attività di vario genere.

In questo contesto i sistemi di gestione dei database relazionali (RDBMS) svolgono il ruolo di custodi delle informazioni dovendone garantire la confidenzialità, l'integrità e la disponibilità.

La natura dei dati memorizzati all'interno di tali sistemi è, molto spesso, di una tale importanza o delicatezza da rendere pressante il problema della loro riservatezza.

Nell'ambito della sicurezza dei sistemi informativi è noto che la riservatezza richiede, in primo luogo, l'adozione di tecniche e strumenti idonei a consentire l'accesso alle informazioni soltanto ai soggetti effettivamente autorizzati.

Dunque uno dei presupposti per poter garantire la confidenzialità è rappresentato proprio dalla definizione ed implementazione di opportune politiche per il controllo degli accessi.

Tra le possibili implicazioni derivanti dall'adozione di tali politiche non può essere taciuta l'importanza del ricorso all'utilizzo di adeguati protocolli crittografici in grado di costituire un'ulteriore linea difensiva per le informazioni memorizzate nei database.

Questa necessità trova una giustificazione nel principio della cd. difesa stratificata (*defense in depth*) in virtù del quale la protezione di un sistema informativo non può essere correttamente demandata all'adozione di un'unica linea di difesa, ma presuppone il coinvolgimento di più misure tra loro complementari.

D'altra parte di questo importante principio sembra ormai essersi fatto portavoce anche il legislatore italiano che, nel codice della privacy, stabilisce che il trattamento dei dati sensibili deve avvenire mediante il ricorso a soluzioni tecniche idonee a rendere i dati inintelligibili, anche nei confronti dei soggetti autorizzati ad accedervi, indicando nell'uso della crittografia una di queste possibili soluzioni (articolo 22 comma 6 D.Lgs. 196/03).

Data l'importanza e la delicatezza del tema in questo documento vengono, dunque, proposte alcune considerazioni in ordine alle problematiche relative alla corretta identificazione ed utilizzo degli strumenti di crittografia per la protezione delle informazioni all'interno dei sistemi RDBMS.

Sebbene alcune di queste considerazioni abbiano una valenza generale il target assunto come riferimento è rappresentato dai database Oracle in quanto la presenza di queste soluzioni tecnologiche in ambienti "high-end", spesso caratterizzati da un elevato volume ed importanza dei dati trattati, rende la tematica di particolare interesse.

Il ruolo della crittografia nella strategia di protezione delle informazioni

Come già anticipato l'obiettivo principale della crittografia deve essere quello di fornire un'ulteriore livello di protezione delle informazioni, nella eventualità che si verifichi una circonvenzione del sistema di controllo degli accessi.

In tale contesto l'utilizzo di protocolli crittografici, unitamente alla presenza di congrue procedure per la conservazione e la gestione del ciclo di vita delle chiavi, può effettivamente rendere infruttuosi i tentativi di accesso illeciti ma non impedisce, tuttavia, il compimento di ulteriori azioni dannose aventi per oggetto l'alterazione e/o la cancellazione degli stessi dati, eventi questi che possono essere vanificati dall'esistenza di altre misure protettive come, ad esempio, la regolare esecuzione di procedure di backup.

Generalmente le soluzioni crittografiche adottabili possono essere distinte in due categorie a seconda che abbiano per oggetto i dati in transito sulla rete oppure quelli memorizzati direttamente all'interno dei database.

Ferma restando la validità del ricorso alle soluzioni del primo tipo (vedi SSL/TLS ed IPSec), volte a rendere difficoltosi una serie di attacchi (replay, sniffing e session hijacking), va tuttavia ricordato che gli attacchi di questo tipo sono statisticamente assai meno diffusi di quelli aventi per oggetto i dati che risiedono all'interno dei sistemi.

Dunque per questi ultimi si pone, in modo ancora più urgente, l'esigenza di individuare le strategie tali da garantire la loro riservatezza pur senza compromettere altre esigenze come quelle legate alle performance e/o alla fruibilità dei sistemi RDBMS.

Giunti a questo livello possiamo perciò fare una ulteriore distinzione tra i sistemi di cifratura che operano a livello di filesystem e quelli che, invece, operano a livello di database o, meglio ancora, a livello di singole tabelle, analizzando le caratteristiche, i vantaggi e gli svantaggi di ognuno di questi.

Crittografia dei dati a livello di filesystem

Le soluzioni di questo tipo agiscono direttamente a livello di filesystem sfruttando appositi layer crittografici propri del sistema operativo in uso (si pensi, ad esempio, all'EFS – Encrypted File System dei sistemi Windows) o di terze parti.

Il risultato che si ottiene è rappresentato pertanto dalla cifratura dell'intero contenuto dei datafiles che, normalmente, rappresentano le unità di memorizzazione delle strutture di funzionamento degli RDBMS (tablespace, tabelle, indici, ecc...)

Sebbene questa strategia sia teoricamente in grado di garantire la confidenzialità dei dati, essa presenta una serie di svantaggi:

- in primo luogo non è possibile circoscrivere l'utilizzo di questa tecnica a singole porzioni di dati. Questo può indurre dei seri problemi di performance poiché qualunque attività compiuta sul database, anche se necessaria per la semplice allocazione di ulteriori risorse (si pensi, ad esempio alle richieste di allocazione di ulteriore spazio sulle tablespace per accomodare le operazioni di scrittura), comporta inevitabilmente delle operazioni di lettura e, quindi, di decifratura, e ciò a prescindere dalla reale necessità che i dati vengano protetti;
- la cifratura è idonea a proteggere i dati contro gli attacchi portati a livello di sistema operativo ma nulla può in caso di aggiramento del sistema di controllo degli accessi alla base dati. Se un aggressore infatti riesce ad accedere in qualche modo al database verrà probabilmente visto come un utente legittimo e potrà leggere senza alcuna difficoltà i dati;
- infine l'adozione di questa strategia rende impossibile cifrare porzioni differenti di dati con chiavi diverse, come potrebbe essere richiesto nel caso in cui il database contenga informazioni sensibili di diversa natura destinate ad essere elaborate e trattate da soggetti appartenenti ad aree organizzative differenti;

E' evidente quindi che la presenza di queste controindicazioni rende il ricorso a tale soluzione non idoneo almeno nel caso specifico.

Crittografia dei dati a livello di database

Gli svantaggi insiti nella soluzione precedente possono essere superati adottando una strategia di cifratura a livello di database o, per essere più precisi, a livello di singole colonne di tabella.

Questo stratagemma non solo permette di circoscrivere l'uso della crittografia alle sole informazioni che necessitano di una effettiva protezione, ma non preclude la possibilità di predisporre un sistema di crittografia multilivello cifrando, se del caso, le informazioni con chiavi differenti.

Inoltre, anche nell'ipotesi in cui il sistema di controllo degli accessi alla base dati venga aggirato, la protezione continua a rimanere efficace poiché essa è insita nello stesso database.

Dal punto di vista delle performance l'intero RDBMS risente positivamente di questo diverso approccio a condizione che il ricorso alla crittografia sia limitato ai soli campi contenenti informazioni sensibili e non venga esteso, arbitrariamente, ad intere strutture tabellari.

D'altra parte non va dimenticato che, al di là delle performance, si pone anche il problema legato alla maggiore richiesta di spazio su disco in quanto il salvataggio di informazioni in forma cifrata ha dei requisiti di occupazione maggiori della memorizzazione di testo in chiaro.

Un'ultima considerazione va infine riservata alla necessità di inibire la visualizzazione dei dati sensibili nei confronti di quegli utenti che rivestono il ruolo di amministratori del sistema.

Perché tale requisito possa dirsi soddisfatto è necessario che l'intero meccanismo di cifratura sia basato sulla limitazione della possibilità per gli utenti amministratori di venire a conoscenza della chiave ed utilizzarla per decifrare i dati.

Strategie per la creazione delle chiavi

La generazione delle chiavi rappresenta un aspetto cruciale dal quale può dipendere l'efficacia di un intero sistema crittografico.

In linea di massima possiamo distinguere tre ipotesi a seconda che i dati sensibili debbano essere protetti mediante:

1. una chiave per tabella condivisa fra tutti gli utenti;
2. una chiave per tabella differente per ciascun utente;
3. una chiave per tabella calcolata in automatico ed in modo differente per ogni record condivisa fra tutti gli utenti;

Nei primi due casi è importante avvalersi di un generatore di numeri casuali che sia sicuro dal punto di vista crittografico e, quindi, immune da qualsiasi forma di predizione.

Generatori di questo tipo sono stati introdotti nella prima release della versione 10g (vedi RandomBytes, RandomInteger e RandomNumber del package DBMS_CRYPTO¹).

Precedentemente, a partire dalla versione 9i release 2, erano state rese disponibili nel package DBMS_OBFUSCATION_TOOLKIT delle funzioni per la generazione automatica delle chiavi da usare con gli algoritmi DES/3DES (vedi DesGetKey e Des3GetKey²).

Analoghe funzionalità non sono invece contemplate per le versioni di database 8i, né si può pensare di utilizzare, a tal fine, il package DBMS_RANDOM in quanto non adeguato allo scopo.

Con riferimento all'ultima ipotesi, invece, la fase di generazione può essere demandata ad un apposito algoritmo che permetta di calcolare una chiave differente per ogni riga della tabella come, ad esempio, nel seguente schema in cui il valore utilizzabile per la cifratura è ricavato dall'hash del risultato di uno XOR tra il valore della chiave primaria della tabella ed una stringa arbitraria:

```
key := md5(xor(chiave_primaria_tabella,'stringa'));
```

Questo tipo di approccio può essere particolarmente vantaggioso per aggirare il problema della conservazione delle chiavi (vedi paragrafo successivo) ma richiede comunque una congrua protezione dell'algoritmo di calcolo.

¹ Queste funzioni erano inizialmente affette da un bug applicativo risolto con la versione 1.0.1.3..

² Anche queste erano inizialmente affette da un bug applicativo risolto con la versione 10.

A tal fine si potrebbe pensare di includere l'algoritmo all'interno di un package sottoposto all'utility Oracle wrap in modo da non risultare intelligibile ³ (particolare attenzione deve essere inoltre riservata alla concessione del privilegio di esecuzione sul package).

Strategie per la conservazione delle chiavi

Un altro aspetto da valutare attentamente è quello relativo alle modalità di conservazione delle chiavi.

A questo proposito possiamo prendere in considerazione i seguenti scenari:

1. custodia delle chiavi all'esterno del database;
2. custodia delle chiavi nel database;
3. custodia delle chiavi direttamente da parte degli utenti;

La prima ipotesi può comportare, ad esempio, l'utilizzo di appositi file di testo come unità di memorizzazione delle chiavi all'interno del filesystem.

Così facendo la sicurezza delle informazioni mantenute nel database viene a dipendere fortemente dalla protezione accordata al sistema operativo e richiede, pertanto, un hardening adeguato della piattaforma sottostante.

Si può eventualmente valutare l'opportunità di sfruttare, per la stessa finalità, un server applicativo esterno e/o di proteggere i file mediante una ulteriore master password.

Questo tipo di soluzione si caratterizza per un livello di complessità extra, almeno per quello che riguarda l'accesso ed il recupero delle chiavi da parte degli utenti del database, ma si dimostra tuttavia efficace, soprattutto nei casi in cui le esigenze di protezione dei dati nei confronti degli utenti DBA abbiano una importanza primaria.

Più flessibile si dimostra invece il secondo scenario ipotizzato. In tal caso, infatti, il salvataggio delle chiavi all'interno di apposite strutture del database è in grado di facilitarne il recupero ma pone dei problemi di sicurezza, sia nel caso in cui il sistema di controllo degli accessi venga aggirato, sia nei confronti degli utenti amministratori, normalmente in possesso di privilegi di sistema particolarmente ampi.

Una prima alternativa può essere perciò costituita dalla memorizzazione di singole porzioni di chiavi all'interno di un package e dall'intervento di un opportuna procedura in grado di ricostruire il segreto nella sua interezza.

Ovviamente, adottando questa soluzione, l'algoritmo di ricostruzione della chiave deve essere adeguatamente protetto (vedi a tale riguardo quanto detto nel paragrafo precedente relativamente all'utility wrap).

Altrimenti si può anche ricorrere ad una tabella separata, di proprietà di un utente non applicativo, nella quale memorizzare le chiavi in forma cifrata, mediante il ricorso ad una master key, oppure in chiaro.

Nel primo caso si tratta di prediligere le esigenze di sicurezza a fronte delle quali, però, si pone il problema di una maggiore complessità nel recupero mentre, nella seconda ipotesi, vengono beneficiate le esigenze applicative.

Con riferimento a quest'ultima soluzione tuttavia il meccanismo dovrebbe essere basato su una limitazione dell'accesso alla tabella contenente i dati sensibili, attraverso una procedura incaricata di selezionare i dati criptati, recuperare la chiave di decrittatura e sottoporla ad un processo di trasformazione prima dell'utilizzo effettivo.

³ L'utility wrap nelle versioni 8i/9i non offusca le costanti presenti nel codice. Questo inconveniente è stato superato nella successiva versione 10g che ha inoltre introdotto dei cambiamenti diretti rendere più ardui gli eventuali tentativi di reverse engineering del codice offuscato.

Così facendo gli utenti che conservano un accesso diretto alla tabella dei dati continueranno a vedere le informazioni offuscate e non saranno in grado di recuperare la chiave per la decifrazione.

Analogamente quelli che conservano un accesso diretto alla tabella delle chiavi non potranno utilizzarle fruttuosamente a causa dell'intervento del processo di trasformazione delle stesse, sempre che il relativo algoritmo non sia prevedibile e sia adeguatamente protetto.

Infine l'ultima soluzione prospettata, consistente nella conservazione diretta delle chiavi da parte dei singoli utenti, dovrebbe essere scartata in quanto non idonea a fornire adeguate garanzie soprattutto in considerazione delle problematiche e delle vulnerabilità che tale approccio può comportare (smarrimento, necessità di adottare meccanismi per impedirne la trasmissione in chiaro sulla rete, sensibilizzazione degli utenti negli aspetti attinenti la gestione, ecc...).

L'ultima considerazione deve essere riservata alla necessità di prevedere ed implementare un meccanismo di modifica periodica delle chiavi per evitare che il loro utilizzo, protratto per un periodo di tempo troppo elevato, possa aumentare le probabilità di una loro compromissione.

Le soluzioni crittografiche offerte dagli RDBMS Oracle

A partire dalla versione 8i Release 3 (8.1.7) del suo RDBMS Oracle ha messo a disposizione una serie di procedure utilizzabili direttamente per far fronte ad eventuali esigenze di protezione dei dati mediante crittografia.

Tali funzionalità sono state racchiuse nel package **DBMS_OBFUSCATION_TOOLKIT** che espone i metodi **DESEncrypt** e **DESDecrypt**, rispettivamente per la cifratura/decifrazione con algoritmo DES, nonché **DES3Encrypt** e **DES3Decrypt** rispettivamente per la cifratura/decifrazione dei dati con algoritmo 3DES.

Entrambi gli algoritmi possono essere utilizzati per la cifratura dei soli tipi di dati RAW e VARCHAR2, in modalità CBC (Cipher Block Chaining), con una lunghezza di chiave rispettivamente di 56 e 128/192 bit ed una lunghezza di input che deve essere un multiplo di 8 byte (non sono ammesse più passate simultanee di cifratura).

Le procedure per il 3DES utilizzano, per default, l'implementazione a doppia chiave con una lunghezza complessiva di quest'ultima di 128 bit (per abilitare l'implementazione a 3 chiavi occorre esplicitare, nella chiamata, un parametro specifico e precisare una chiave di lunghezza pari a 192 bit).

Con la versione 10g release 1 ha fatto la sua comparsa un ulteriore package crittografico denominato **DBMS_CRYPTO** che ha reso disponibili ulteriori funzionalità per la protezione delle informazioni.

Tra gli algoritmi implementati ora compaiono anche l'AES con lunghezza di chiave a 128, 192 e 256 bit ed il RC4.

Per quanto concerne le modalità di codifica sono supportati, oltre a Cipher Block Chaining, anche Cipher-Feedback, Electronic Codebook ed Output-Feedback.

Infine, a livello di data type, sono supportati, oltre a RAW, anche BLOB e CLOB e sono presenti apposite funzioni per il padding con zero o conformi allo standard PKCS #5.

Per concludere va menzionato che, con la versione 10g release 2, è stato introdotto un meccanismo per la cifratura trasparente ed automatica dei dati nelle tabelle denominato TDE acronimo di Transparent Data Encryption.

Riferimenti

[1] Transparent Data Encryption, <http://www.oracle.com/technology/oramag/oracle/05-sep/o55security.html>;

[2] Encrypt Your Data Assets, Oracle Magazine January/February 2005
(<http://www.oracle.com/technology/oramag/oracle/05-jan/o15security.html>);
[3] Database Encryption in Oracle 9iR2,
<http://otn.oracle.com/deploy/security/oracle9ir2/pdf/dbcrypt9ir2.pdf>;