POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Tecniche di cifratura dei dati nei database relazionali

Relatori
Prof. Antonio Lioy
Ing. Andrea Atzeni

Candidato
Lorenzo Ceccarelli

Tutor aziendale
Ing. Marco Mangiulli

Anno accademico 2020/2021

† A mia mamma

Sommario

Nel mondo di oggi la tematica della sicurezza informatica sta diventando sempre più importante sia per il consumatore che per il venditore di servizi. Ormai i servizi digitali fanno parte della quotidianità di tutte le persone e il problema della privacy sta assumendo sempre più rilevanza. Infatti il consumatore vuole avere garanzie sulla sicurezza dei propri dati memorizzati in un sistema remoto. Dall'altra parte le aziende devono attuare un certo livello di protezione sui propri sistemi per prevenire eventuali attacchi e furti di dati che possono rappresentare anche un danno d'immagine. Inoltre ogni organizzazione deve essere conforme agli standard e leggi che regolano il trattamento dei dati personali. Affidarsi alla sicurezza perimetrale o al controllo dell'accesso al database non fornisce una sicurezza adeguata. Infatti, il controllo degli accessi applicato al database può essere aggirato in diversi modi. Per esempio, un intruso può infiltrarsi nel sistema che ospita il database e cercare di estrarre i dati dal disco.

In questa trattazione si è effettuata inizialmente un'analisi dello stato dell'arte delle tecniche di cifratura a livello database vista come valida alternativa alle classiche tecniche di cifratura a livello applicazione e sistema operativo che possiedono problemi legati alle performance, sicurezza e flessibilità.. Di particolare interesse sono i database relazionali che, nonostante il grande successo recente avuto dai database NoSQL, sono ancora ampiamente diffusi per la memorizzazione dei dati.

La cifratura dei dati può fornire una forte sicurezza per i dati *a riposo* ma la vera sfida è rappresentata dallo sviluppo di una corretta infrastruttura che oltre all'utilizzo di un algoritmo sicuro deve anche implementare un sistema sicuro per la gestione delle chiavi crittografiche. La strategia di sviluppo deve infatti tenere in considerazione una moltitudine di fattori cercando di ottenere un buon compromesso tra requisiti di sicurezza e performance. Infatti i processi crittografici aggiungono un overhead computazionale significativo. Uno dei fattori fondamentali che caratterizzano la Database Encryption è la gestione delle chiavi crittografiche. In questo lavoro un capitolo è interamente dedicato alla descrizione di tale problematica che deve essere necessariamente gestita. Infatti, una pessima gestione delle chiavi può introdurre vulnerabilità sfruttabili da un attaccante che potrebbe ottenere la chiave di cifratura dei dati e potenzialmente effettuare operazioni illecite su di essi. Altri fattori che devono essere presi in considerazioni sono il livello a cui deve avvenire la cifratura e con quale granularità. Si effettuerà un'analisi delle varie soluzioni mostrandone vantaggi e svantaggi.

Una volta definita la strategia occorre scegliere il prodotto più conforme alle proprie esigenze. Molti vendor, sia open-source che proprietari, forniscono implementazioni per garantire la cifratura sui database come la *Transparent Data Encryption*. I vari prodotti sono stati analizzati e testati in modo da effettuare confronti in termini di impatto sulle performance e sulla dimensione del database. Al tal proposito è stato definito un processo di valutazione, chiamato *Benchmarking Framework*, che permette una comparazione il più critica e omogenea possibile.

L'ultimo obiettivo di questo lavoro di tesi è stato quello di sviluppare una proof-of-concept di una soluzione di *cifratura lato client* usando Java e la Java Cryptography Extension (JCE). Il software sviluppato permette ad una generica applicazione client di abilitare la cifratura dei dati apportando numerosi benefici in termini di sicurezza dato che le chiavi crittografiche vengono gestite dal client stesso e i dati vengono inviati al database già cifrati.

Indice

Capitolo 1

Introduzione

1.1 Motivazioni

Negli ultimi anni il problema della privacy è diventato sempre più importante e le recenti approvazioni di nuove leggi hanno portato le aziende ad utilizzare tecniche di cifratura per proteggere i dati sensibili memorizzati nei database. Tuttavia l'utilizzo della crittografia introduce problematiche come perdita di performance e protezione delle chiavi di cifratura. Tali problemi devono essere risolti in modo da garantire un sistema affidabile, sicuro e trasparente.

In questo lavoro di tesi si mostrerà come la Database Encryption sia un'utile e valida alternativa alle più utilizzate tecniche di cifratura a livello sistema operativo e applicazione. Infatti queste ultime nonostante apportino numerosi benefici possiedono intrinsecamente numerosi svantaggi. Problematiche come flessibilità, sicurezza, performance e supporto sono caratterizzanti delle soluzioni di cifratura a livello sistema operativo e applicazione mentre la Database Encryption risulta essere un buon compromesso introducendo un robusto sistema di sicurezza per i dati e gestione delle chiavi, permettendo una scelta della granularità del dato da cifrare, ottimizzando le performance e minimizzando il tempo e le risorse spese da una organizzazione per implementare il proprio database in modo sicuro.

Anche nel caso dei database, come in qualsiasi ambito di sicurezza, è necessario garantire tre proprietà principali, la cosiddetta triade "CIA"[1] [1]:

- Riservatezza (C);
- Integrità (I);
- Disponibilità (A).

Nel mondo della protezione dei dati memorizzati nei database queste caratteristiche assumono particolare rilievo.

Per riservatezza dei dati si intende la protezione delle informazioni dall'essere esposte ad una terza parte non autorizzata. Ogni informazione che un'azienda possiede ha un valore, specialmente nel mondo di oggi. Che si tratti di dati finanziari, numeri di carte di credito, segreti commerciali o documenti legali, tutto richiede la giusta protezione e riservatezza. In altre parole, solo le persone che sono autorizzate a farlo dovrebbero essere in grado di accedere ai dati sensibili. Un fallimento nel mantenere la riservatezza significa che qualcuno che non dovrebbe avere accesso è riuscito a ottenere le informazioni private. Ciò può causare gravi conseguenze. Un attaccante che ottiene i privilegi di accesso al sistema che ospita un database che memorizza dati sensibili può, ad esempio, divulgarli o ottenere informazioni riservate.

[1] Dall'inglese *Confidentiality*, *Integrity* e *Availability*.

Per integrità si indica l'accuratezza e la completezza dei dati. I controlli di sicurezza incentrati sull'integrità sono progettati in modo da impedire che i dati vengano modificati e danneggiati da una terza parte non autorizzata. L'integrità, quindi, implica il mantenimento della coerenza e dell'affidabilità dei dati durante il loro intero ciclo di vita. Procedure di backup e software per il rilevamento degli errori sono dei possibili controlli di sicurezza per garantire l'integrità dei dati.

Per disponibilità dei dati si intende che le informazioni sono accessibili agli utenti autorizzati. La disponibilità è tipicamente associata all'affidabilità e ai tempo di attività e risposta del sistema. Quest'ultimo perciò dovrà essere progettato per garantire un ottimo trade-off tra performance e sicurezza. Un database estremamente sicuro ma che non permetta disponibilità immediata è molto probabile che non venga utilizzato così come nel caso opposto.

Per preservare la riservatezza dei dati, prevalentemente si utilizzano delle politiche di controllo dell'accesso definite sul sistema di gestione del database (DBMS). Un controllo degli accessi, cioè un insieme di autorizzazioni, può assumere diverse forme a seconda del modello di dati sottostante (ad esempio, relazionale, XML), del modo in cui le autorizzazioni sono amministrate, seguendo un controllo di accesso discrezionale (DAC)[2] [2], controllo d'accesso basato sui ruoli (RBAC)[3] [3] o controllo d'accesso obbligatorio(MAC)[4] [1].

Qualunque sia il modello di controllo degli accessi, le autorizzazioni applicate al database server possono essere aggirate in diversi modi. Per esempio, un intruso può infiltrarsi nel sistema informativo e cercare di estrarre i dati dal disco. Un'altra fonte di minacce deriva dal fatto che molti database oggi sono gestiti da Database Service Provider (DSP) esterni. Quindi, i proprietari dei dati non hanno altra scelta che fidarsi dei DSP che sostengono che i loro sistemi sono completamente sicuri e i loro dipendenti sono al di là di ogni sospetto, un'ipotesi spesso smentita dai fatti. Infine, un amministratore di database (DBA) ha abbastanza privilegi per manomettere la definizione del controllo di accesso e per spiare il comportamento del DBMS [1, p. 1].

Il ricorso a tecniche crittografiche per integrare e rafforzare il controllo degli accessi ha recentemente ricevuto molta attenzione dalla comunità dei database. Lo scopo della crittografia dei database è quello di assicurare l'opacità del database mantenendo l'informazione nascosta a qualsiasi persona non autorizzata (per esempio degli intrusi). Anche se gli aggressori riuscissero a passare attraverso il firewall e bypassare le politiche di controllo dell'accesso, avrebbero ancora bisogno delle chiavi crittografiche per decifrare i dati [1].

Per **Database Encryption** si intende un processo che utilizza un algoritmo per trasformare i dati memorizzati in un database in un *testo cifrato* che è incomprensibile senza essere decifrato. Perciò lo scopo della *Database Encryption* è quello di proteggere i dati memorizzati in un database dall'accesso di individui che potenzialmente potrebbero avere intenzioni dannose e malevole. Inoltre cifrare un database riduce l'incentivo per gli individui a violare il suddetto database considerandolo "insignificante" in quanto contiene dati crittografati di poca o nessuna utilità [1].

La crittografia può fornire una forte sicurezza per i dati a riposo[5], ma lo sviluppo di una strategia di crittografia del database deve prendere in considerazione molti fattori. Per esempio:
dove dovrebbe essere eseguita la crittografia, nel livello di memorizzazione, nel database o nell'applicazione dove i dati sono stati prodotti?
Quanti dati dovrebbero essere crittografati per fornire una sicurezza adeguata?
Quale dovrebbe essere l'algoritmo di crittografia e modalità di funzionamento?
Chi dovrebbe avere accesso alle chiavi di crittografia?
Come minimizzare l'impatto della crittografia del database sulle prestazioni?

[2] Il proprietario della risorsa ha il controllo completo su chi può avere accesso a una risorsa specifica.

[3] Questo sistema è un approccio alla gestione della sicurezza e dei permessi d'accesso in cui i ruoli e le autorizzazioni sono assegnate all'interno dell'infrastruttura informatica di un'organizzazione. L'intero concetto è basato sui ruoli.

[4] L'accesso è determinato dal sistema, non dal proprietario del sistema.

[5] La definizione di dato a riposo è riportata nel Capitolo 2.

1.2 Struttura di un DBMS

Innanzitutto occorre descrivere la struttura di un **Database Management System (DBMS)** relazionale in quanto la sua composizione risulterà di notevole importanza durante la trattazione.

Il DBMS accetta dei comandi SQL generati da una varietà di interfacce utente, produce piani di valutazione delle query, esegue questi piani sul database e restituisce le risposte. La Figura 1.1 mostra i componenti che costituiscono un DBMS [4].

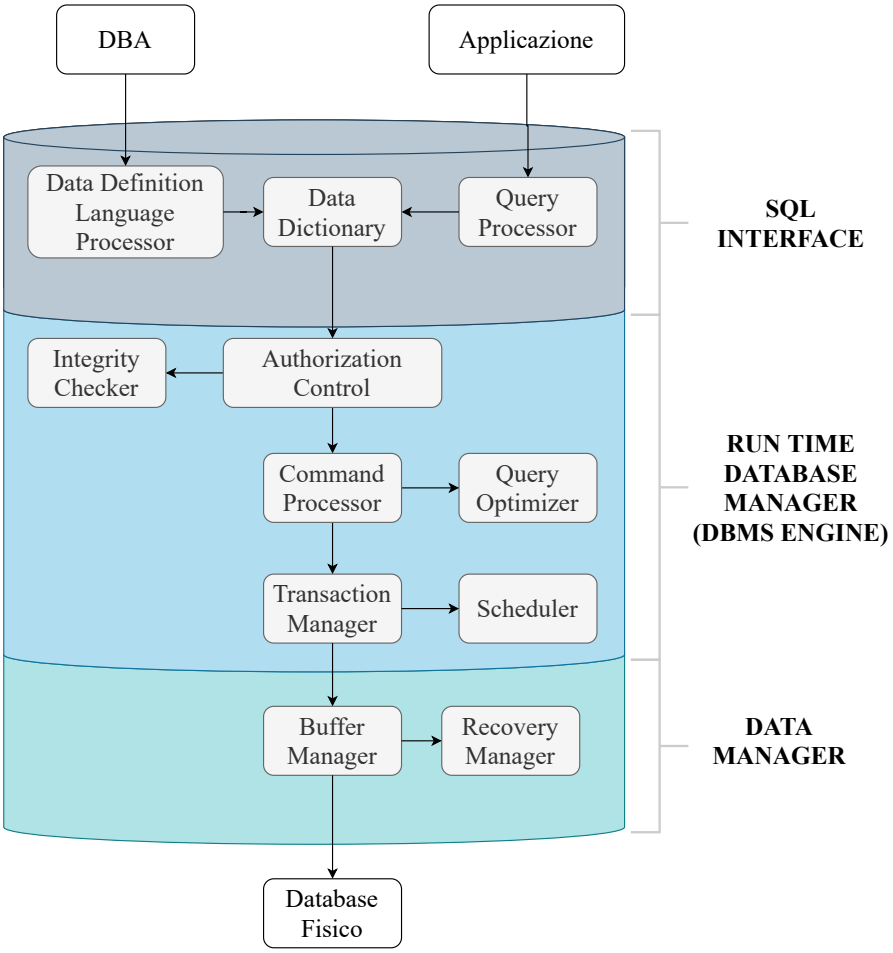


Figura 1.1: Struttura DBMS

Un DBMS è composto da tre componenti principali, ciascuno diviso in più sottocomponenti.

SQL Interface

Si tratta dell'interfaccia del DBMS che accetta comandi SQL provenienti da utenti, applicativi o DBA (Database Administrator), è divisa in:

- *Query Processor.* È un modulo che trasforma le query inserite dall'utente esterno in una serie di istruzioni a basso livello utilizzando il Data Dictionary [4]. Possiede al suo interno un *Data Manipulation Language Processor*. Questo processa le istruzioni DML (Data Maniplation Language). DML è una famiglia di linguaggi che consente di leggere, inserire, modificare o eliminare i dati in un database. Attualmente il linguaggio di manipolazione dati più utilizzato è l'SQL (Structured Query Language), che trova il suo maggiore campo di applicazione nel trattamento dei database relazionali. Inoltre esegue le cosiddette **Stored Procedures** qualora fossero presenti. Una stored procedure è una funzione scritta in un linguaggio procedurale (estensione SQL) che permette un maggior controllo del semplice SQL, che è linguaggio puramente descrittivo. Esempi di linguaggi sono PL/SQL (Oracle) [5], Transact-SQL (Microsoft)[6] e PL/pgSQL (PostgreSQL) [7];

- *Data Definition Language Processor.* Processa le istruzioni DDL (Data Definition Language). DDL è un linguaggio, parte del linguaggio SQL, che permette di creare, modificare o eliminare gli oggetti in un database ovvero agire sullo schema di database. Sono i comandi DDL a definire la struttura del database e quindi l'organizzazione logica dei dati in esso contenuti, ma non fornisce gli strumenti per modificare i valori assunti dai dati o per interrogare i dati stessi per il quale si usano rispettivamente il Data Manipulation Language e il Data Query Language. Viene utilizzato principalmente dal DBA;
- *Data Dictionary.* Contiene tutti i metadati che descrivono il DBMS.

Run Time Database Manager (DBMS Engine)

È il componente software principale del DBMS ed esegue il piano di esecuzione ricevuto dal *query processor*. È composto da [4]:

- *Authorization Control.* Verifica i permessi dell'utente che richiede l'esecuzione della query;
- *Integrity Checker.* Verifica i vincoli di integrità del piano di esecuzione in modo che solo i dati validi possano entrare nel database;
- *Command Processor.* Processa la query dopo essere stata ottimizzata dal *Query Optimizer*. Il *Query Optimizer* determina una strategia ottimale per l'esecuzione della query;
- *Transaction Manager.* Garantisce che il DBMS rimanga in uno stato consistente e coerente. Viene supportato dallo *Scheduler* che permette ad utenti multipli di lavorare contemporaneamente sugli stessi dati.

Data Manager

È il componente responsabile per la gestione fisica dei dati ed è composto da: [4]

- *Buffer Manager.* Si tratta di una memoria cache che si occupa di scrivere/leggere i dati sul disco;
- *Recovery Manager.* Si occupa della corretta gestione degli errori e fallimenti del DBMS.

Si prenda come esempio la seguente query:

*SELECT **
FROM UTENTI

L'applicazione invia al DBMS questa query che deve estrarre tutte le righe dalla tabella chiamata *UTENTI*. Il *Query Processor* utilizzando il *Data Dictionary* estrae i metadati della tabella UTENTI ed effettua un'analisi lessicale, sintattica e semantica della query. Se identifica un errore (tabella non esistente o errore nella sintassi della query) lo ritorna all'applicazione. Se l'analisi va a buon fine genera una rappresentazione interna della query basata sull'algebra relazionale (il cosiddetto "Query Tree").

Il *Query Tree* viene inviato al DBMS Engine che attraverso l'*Authorization Control* e *Integrity Checker* verifica se l'applicazione ha i permessi per eseguire la query SQL e se questa rispetti i vincoli di integrità richiesti dal database.

A questo punto il *Query Optimizer* identifica una possibile strategia di ottimizzazione utilizzando algoritmi di "Algebraic Optimization" e "Cost-based Optimization" che producono un *Canonical Query Tree* [8].

La query di esempio non rappresenta una transazione ma nel caso in cui lo fosse stato il *Transaction Manager* si sarebbe occupato di gestirla in modo da garantire le proprietà ACID [6].

[6] *A*tomicity, *C*onsistency, *I*solation e *D*urability

Infine il *Buffer Manager*, a partire dalla rappresentazione della query generata nei passi precedenti, restituisce al DBMS Engine le righe della tabella UTENTI estraendo i dati dalle pagine sul disco.

Un database *relazionale* è basato sul modello relazionale ed è composto da tabelle. Ciascuna riga di una tabella viene chiamata, in questa trattazione, **tupla**. Quindi, una tupla di una tabella rappresenta un record con un identificatore univoco, chiamato *chiave*.

Un concetto richiamato più volte in questo lavoro di tesi è l'**indice**. Si tratta di una struttura dati utilizzata per migliorare il tempo di ricerca di una query. Sostanzialmente in una tabella senza indici occorre leggere tutti i dati presenti in essa, invece, l'utilizzo di un indice permette di ridurre i dati in un sottoinsieme per completare la ricerca. Perciò questa tecnica viene utilizzata dai DBMS per ottimizzare l'esecuzione di una query e migliorare le performance.

1.3 Minacce

In questa sezione si vuole fornire una visione delle possibili minacce a cui i moderni DBMS sono soggetti ogni giorno, come motivazione aggiuntiva all'importanza della tematica della Database Encryption [9].

La seguente lista mette in evidenza i rischi di sicurezza legati al mondo dei database [10].

Abuso dei privilegi

Si può avere quando un singolo utente ha completo accesso al sistema e quindi può eseguire operazioni illecite. Questo tipo di attacco minaccia tutta la *triade CIA*. Infatti un potenziale attaccante può leggere e modificare i dati memorizzati violando le proprietà di riservatezza e integrità. Inoltre può effettuare delle operazioni che possono compromettere la disponibilità del database, come ad esempio metterlo offline.

Elevazione dei privilegi

Consiste in una eccessiva esposizione del sistema che porta alla scoperta di falle delle quali un intruso può approfittare per attuare un cambiamento dei privilegi a suo favore. In questo modo l'attaccante può accedere al sistema con privilegi di amministratore ed effettuare qualsiasi tipo di operazione. Si tratta di una minaccia molto simile alla precedente che può compromettere tutte e tre le proprietà principali di sicurezza.

Vulnerabilità della piattaforma che ospita il DBMS

Il sistema operativo che giace sotto al database può essere affetto da vulnerabilità che potrebbero compromettere la sicurezza dei dati memorizzati nel DBMS e in generale mettere a rischio tutta la triade CIA.

SQL Injection

Un attaccante può inviare delle query SQL malevole al sistema. Se il server non effettua opportune operazioni di validazione e sanificazione degli input provenienti dall'esterno, l'attaccante può compromettere i dati memorizzati nel DBMS. Questo tipo di attacco minaccia le proprietà di riservatezza e integrità. Infatti un attaccante può inviare delle query malevole sia per ottenere dei dati sensibili sia per alterare il loro stato sul database (modifica o eliminazione).

Debole tracciamento

Una politica di controllo dei database assicura una registrazione automatizzata, tempestiva delle transazioni che avvengono nel database. Tale politica dovrebbe essere presa in considerazione durante il progetto della sicurezza di un DBMS.

Denial Of Service

Si tratta di un attacco informatico in cui l'attaccante cerca di rendere una macchina o una risorsa di rete non disponibile per i suoi utenti previsti, interrompendo temporaneamente

o indefinitamente i servizi di un host collegato a Internet. Le tipologie di attacchi DoS riguardano principalmente Network Flooding e Buffer Overflow. Questo tipo di minaccia mette a rischio la proprietà di disponibilità in quanto può interrompere il servizio erogato dal sistema.

Vulnerabilità nel protocollo di comunicazione del DBMS

Un numero crescente di vulnerabilità di sicurezza viene identificato nei protocolli di comunicazione dei database di tutti i fornitori di database.

Autenticazione debole

Una strategia debole di autenticazione introduce vulnerabilità sfruttabili da un attaccante. Le proprietà di riservatezza e integrità possono essere compromesse in quanto un attaccante potrebbe accedere ai dati sensibili.

Esposizione dei backup dei dati

L'esposizione dei backup dei dati è un'importante minaccia che deve essere trattata con cura. Infatti i backup devono essere protetti dal furto o dalla distruzione. Ovviamente questa minaccia può mettere a rischio tutte le proprietà principali di sicurezza. Infatti un attaccante può leggere, modificare ed eliminare i backup non rendendoli più disponibili.

Inferenza

L'inferenza è una tecnica utilizzata per attaccare i database in cui un attaccante può dedurre informazioni sensibili da un database esaminando i dati ad alto livello. Sinteticamente si può classificare l'inferenza come una tecnica di *data mining*, basata su algoritmi di "data aggregation" e "data association", utilizzata per trovare informazioni nascoste agli utenti normali [11, 12]. Un esempio è l'attacco a inferenza basato sugli *Attributi* [11, p. 4], che nella prima fase preliminare raccoglie informazioni pubbliche con le relative informazioni private per allenare una rete neurale e nella seconda fase di attacco utilizza la rete neurale per prevedere le informazioni private a partire da quelle pubbliche. Un *Data Broker*[7] può effettuare questo tipo di attacco.

1.3.1 Considerazioni generali di sicurezza

Per eliminare le minacce sopra citate, ogni organizzazione deve definire e implementare delle politiche di sicurezza [9].

Controllo degli accessi

Il controllo degli accessi assicura che tutte le comunicazioni con i database e altri oggetti del sistema siano conformi alle politiche e ai controlli definiti. Questo assicura che non si verifichino interferenze da parte di qualsiasi aggressore né internamente né esternamente e quindi, protegge i database da potenziali errori che possono avere un impatto così grande da fermare le operazioni dell'azienda. Il controllo degli accessi aiuta anche a minimizzare i rischi che possono impattare la sicurezza del database sui server principali. Per esempio, se una qualsiasi tabella viene accidentalmente cancellata o l'accesso viene modificato, i risultati possono essere sottoposti a rollback o, per certi file, il controllo degli accessi può limitare la loro cancellazione.

Politica di inferenza

Un attacco basato su inferenza può mettere in pericolo l'integrità di un intero database. Più complesso è il database, maggiore dovrebbe essere la sicurezza implementata in associazione ad esso. Se i problemi di inferenza non sono risolti in modo efficiente, le informazioni sensibili possono essere trafugate da persone esterne. Occorre perciò implementare un sistema per l'individuazione delle inferenze che analizza e decide se una query è valida oppure no [11, 12].

[7] I *Data Broker* sono aziende che raccolgono direttamente o acquistano i dati da altre aziende e aggregano tali informazioni con dati provenienti da altre fonti per effettuare statistiche e tendenze. Sono molto presenti nell'ambito dei social network.

Identificazione e Autorizzazione degli utenti

L'identificazione e l'autenticazione degli utenti è la necessità di base per garantire la sicurezza, poiché il metodo di identificazione definisce un insieme di persone che sono autorizzate ad accedere ai dati e fornisce un completo meccanismo di accessibilità.

Cifratura

Per cifratura si intende quel processo che permette di convertire i dati originali in un formato che non possa essere comunemente riconosciuto. I dati così trasformati vengono detti "cifrati". Solo chi possiede la chiave di cifratura può decifrare e ottenere i dati originali (assumendo che l'algoritmo di cifratura sia crittograficamente sicuro). Questo aspetto sarà approfondito durante questo lavoro di tesi.

1.4 Sfide della Database Encryption

La crittografia può fornire una forte sicurezza per i dati a riposo, ma lo sviluppo di una soluzione di crittografia del database deve prendere in considerazione molti fattori. Di seguito è specificato un insieme di requisiti che gli schemi di crittografia dei database dovrebbero soddisfare, compresa la necessità di proteggere la riservatezza dei dati, rilevare modifiche non autorizzate e mantenere alte prestazioni [13].

Modelli dell'attaccante [13]

Gli attaccanti possono essere categorizzati in tre principali classi:

- *Intruso.* È una persona che ottiene l'accesso al sistema e cerca di estrarre informazioni riservate;
- *Addetto interno.* È una persona che appartiene all'organizzazione e cerca di ottenere informazioni a cui non ha accesso autorizzato;
- *Amministratore.* È una persona che possiede i privilegi di amministratore del sistema, ma abusa dei suoi diritti per estrarre informazioni riservate.

Inoltre queste tipologie di attaccanti possono usare diverse strategie di attacco:

- *Diretto.* È un attacco contro lo storage e può essere effettuato accedendo ai file del database con mezzi diversi dal software del database, come la rimozione fisica del supporto di memorizzazione o l'accesso ai file di backup del database;
- *Indiretto.* Negli attacchi indiretti allo storage, l'avversario può accedere alle informazioni dello schema e ai metadati, come i nomi delle tabelle e delle colonne, le statistiche delle colonne e i valori scritti nei log di recupero, al fine di stimare le distribuzioni dei dati;
- *In Memoria.* L'avversario può accedere direttamente alla memoria del software del database. In molti casi, la memoria contiene la cache del database che può contenere grandi quantità di dati in chiaro.

Information Leakage - Attacchi passivi [13]

Un database sicuro non dovrebbe rivelare alcuna informazione sui valori in chiaro del database a utenti non autorizzati. Si possono categorizzare i diversi tipi di attacchi passivi:

- *Static Leakage.* Consiste nell'ottenere informazioni sui valori in chiaro del database osservando un'istantanea del database in un certo momento. Per esempio, se i valori di una tabella sono cifrati in modo tale che valori di testo in chiaro uguali sono crittografati in uguali valori di testo cifrato, è possibile raccogliere facilmente statistiche sui valori in chiaro, come la loro frequenza;
- *Linkage Leakage.* Consiste nell'ottenere informazioni sui valori in chiaro del database collegando un valore del database alla sua posizione nell'indice. Per esempio, se il valore del database e il valore dell'indice sono cifrati allo stesso modo (entrambi i valori cifrati sono uguali), un osservatore può cercare il valore cifrato del database nell'indice, determinare la sua posizione e stimare il suo valore in chiaro;

- *Dynamic Leakage.* Consiste nell'ottenere informazioni sui valori del testo in chiaro del database osservando e analizzando i modelli di accesso e i cambiamenti nel database per un periodo di tempo. Per esempio, se un utente monitora l'indice per un periodo di tempo e, se in questo periodo di tempo solo un valore viene inserito (nessun valore viene aggiornato o cancellato), l'osservatore può stimare il suo valore in chiaro in base alla sua nuova posizione nell'indice.

Modifiche non autorizzate - Attacchi attivi [13]

Oltre agli attacchi passivi in cui i dati sono compromessi come risultato di osservazioni, si descrivono ora gli attacchi attivi che modificano il database:

- *Spoofing.* Consiste nella sostituzione di un valore cifrato con un nuovo valore. Supponendo che le chiavi di crittografia non siano state compromesse, questo attacco è raramente eseguito;
- *Splicing.* Consiste nella sostituzione di un valore di testo cifrato con un altro valore di testo cifrato. In questo tipo di attacco, il contenuto cifrato da una posizione diversa viene copiato in una nuova posizione. Per esempio, se il valore cifrato dello stipendio massimo è scoperto attraverso un attacco di leakage, scambiandolo con lo stipendio criptato dell'attaccante genererà un valore valido come nuovo stipendio;
- *Replay.* Consiste nella sostituzione di un valore cifrato con una vecchia versione precedentemente aggiornata o cancellata.

Chiavi di cifratura [13]

Un aspetto importante della sicurezza dei dati riguarda il supporto del controllo di accesso multiutente di un ambiente di database cifrato in cui ogni utente può accedere (decifrare) solo agli oggetti del database (ad esempio, gruppi di celle, righe e colonne) a cui è stato concesso l'accesso. Criptare l'intero database usando la stessa stessa chiave, anche se si utilizzano i tradizionali meccanismi di controllo dell'accesso, non introduce un livello sufficiente di sicurezza. Criptare gli oggetti di diversi gruppi di sicurezza utilizzando varie chiavi assicura che un utente che possiede una chiave specifica possa decifrare solo gli oggetti del suo gruppo di sicurezza. Un altro importante punto che riguarda le chiavi di crittografia è la loro gestione: dove e come le chiavi di crittografia dovrebbero essere conservate; come vengono distribuite agli utenti; e come recuperare le chiavi di crittografia nel caso in cui vengano perse. In seguito si riferisce a questa proprietà come *Key Management*.

Performance [13]

I meccanismi di sicurezza tipicamente introducono un significativo overhead computazionale. Questo overhead può costituire un problema fondamentale per il DBMS, poiché le prestazioni del DBMS hanno un'influenza diretta sulle prestazioni dell'intero sistema informativo. Quando si cerca di minimizzare l'overhead delle prestazioni legato alla crittografia del database, le seguenti questioni dovrebbero essere considerate:

- *Cifratura selettiva.* Sarebbe auspicabile cifrare solo i dati sensibili mantenendo i dati non riservati non cifrati. Inoltre, solo i dati rilevanti dovrebbero essere cifrati/decifrati durante l'esecuzione di una query. Per esempio, se solo un attributo partecipa ad una query, non è necessario criptare/decriptare l'intero record;
- *Indici e altri meccanismi di ottimizzazione del DBMS.* Cifrare il contenuto del database può impedire ad alcuni meccanismi cruciali di ottimizzazione del DBMS di funzionare correttamente. Per esempio, alcuni fornitori di DBMS non permettono di costruire indici su colonne cifrate, mentre altri lo permettono in base ai valori cifrati della colonna (nel caso non siano salati). Quest'ultimo approccio si traduce in una perdita di alcune delle caratteristiche più ovvie degli indici (le ricerche per range) poiché un tipico algoritmo di crittografia non conserva l'ordine;
- *Encryption Overhead.* È auspicabile che il tempo impiegato per criptare/decriptare i dati sia ridotto al minimo. Per esempio, criptare la stessa quantità di dati usando una singola operazione crittografica è più efficiente che crittografarli in parti usando diverse operazioni crittografiche.

Deployment [13]

Incorporare una soluzione di crittografia su un DBMS esistente dovrebbe essere facile da integrare, vale a dire, è desiderabile minimizzare quanto segue:

- *L'influenza sul livello dell'applicazione.* Alcune soluzioni di crittografia richiedono di modificare l'implementazione del livello dell'applicazione, per esempio cambiando le query SQL per includere operazioni di crittografia. Tali modifiche possono costituire un problema fondamentale per le applicazioni legacy, dove nella maggior parte dei casi, il processo di modifica della loro implementazione è estremamente costoso e, in alcuni casi, potrebbe non essere possibile. Pertanto, una soluzione pratica di crittografia di database non dovrebbe richiedere una modifica importante all'implementazione del livello dell'applicazione;
- *L'influenza sull'architettura del DBMS.* È auspicabile evitare cambiamenti fondamentali all'implementazione del DBMS. La tecnologia dei database esiste da più di 30 anni. Riprogettare il modello relazionale per supportare un nuovo modello di crittografia è inaccettabile. È fondamentale, per quanto riguarda la praticità di una soluzione di crittografia DBMS, che essa sia costruita sopra un'implementazione DBMS esistente, incluse tutte le sue funzionalità, come l'indicizzazione, i meccanismi delle chiavi esterne e gli schemi di blocco;
- *L'influenza sul DBA.* È desiderabile permettere al DBA di eseguire i suoi compiti amministrativi direttamente sui dati criptati, senza la necessità di decifrarli prima (e di conseguenza, evitare che i dati sensibili siano rivelati al DBA);
- *L'overhead dello storage.* Anche se lo stoccaggio al giorno d'oggi è relativamente economico, è preferibile che il database criptato non richieda molto più di memoria rispetto al database originale non crittografato.

1.5 Organizzazione della tesi

La metodologia utilizzata in questo lavoro di tesi è il ***Design Research Metodology (DRM)*** [14, p. 8]. Il DRM si compone di sei fasi, che non sono puramente sequenziali, mostrate nella Figura 1.2.

Il primo step, "Identify Problem & Motivate", consiste nell'identificazione e alla motivazione del problema da risolvere. Il Capitolo 1 evidenzia le motivazioni riguardo allo sviluppo di tecnologie come la Database Encryption per la protezione dei dati su database. Il Capitolo 3 approfondisce la problematica della Key Management che è una tematica critica per quanto riguarda la cifratura dei dati.

Il secondo step, "Define Objectives of a Solution", include le definizione degli obiettivi da raggiungere. Il Capitolo 2 analizza lo stato dell'arte delle tecniche di cifratura. Il Capitolo 4 approfondisce le soluzioni dei principali vendor, sia open-source che proprietari, riguardo alla Database Encryption.

Il terzo step, "Desing & Development", consiste nel progettare e sviluppare una soluzione che vada a risolvere i problemi definiti nei precedenti step. Il Capitolo 6 mostra lo sviluppo di una soluzione client-side encryption.

Il quarto step, "Demonstration", include il progetto e lo sviluppo del processo di valutazione sull'oggetto del problema basato sulle soluzioni individuate. Coinvolge, perciò, l'esecuzione del Benchmarking Framework per la valutazione dei differenti prodotti. Inoltre vengono analizzate le metriche utilizzate durante la valutazione.

Il quinto step, "Evaluation", consiste nel confrontare in modo quantitativo i prodotti esaminati.

L'ultimo step, "Communication", consiste nel mostrare e commentare i risultati ottenuti sotto forma di grafici.

Gli ultimi tre step sono sono racchiusi nei Capitoli 5 e 7.

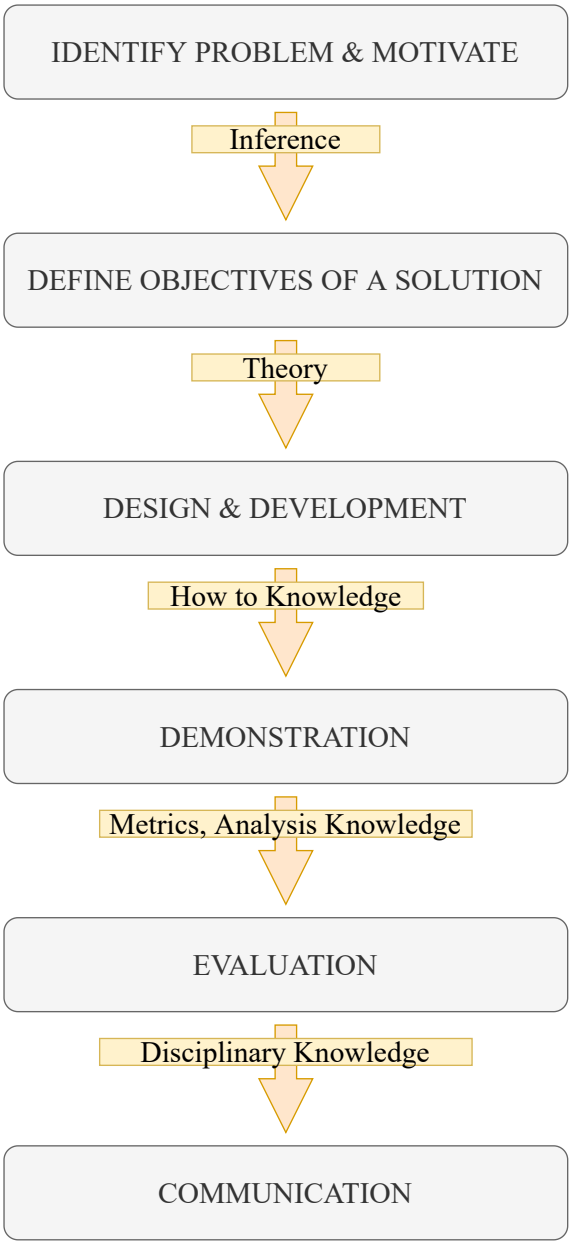


Figura 1.2: Metodologia DRM

Capitolo 2

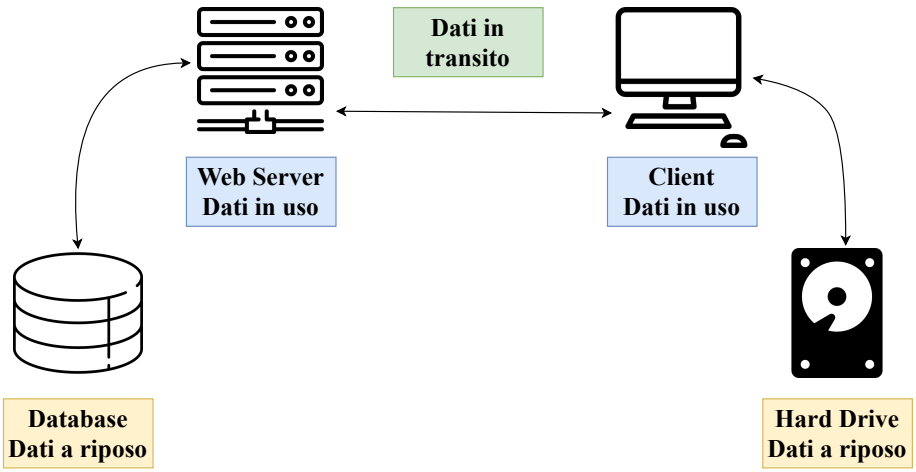
Tecniche di cifratura nei database relazionali

In questo capitolo si approfondirà lo stato dell'arte delle tecniche di crittografia nei database relazionali. Si commenteranno i vari livelli possibili di cifratura e le possibili granularità dei dati cifrati mostrando vantaggi e svantaggi di ciascuna soluzione.

2.1 I tre stati dei dati digitali

Innanzitutto occorre identificare i possibili "stati" che i dati digitali possono assumere durante il loro ciclo di vita [15]:

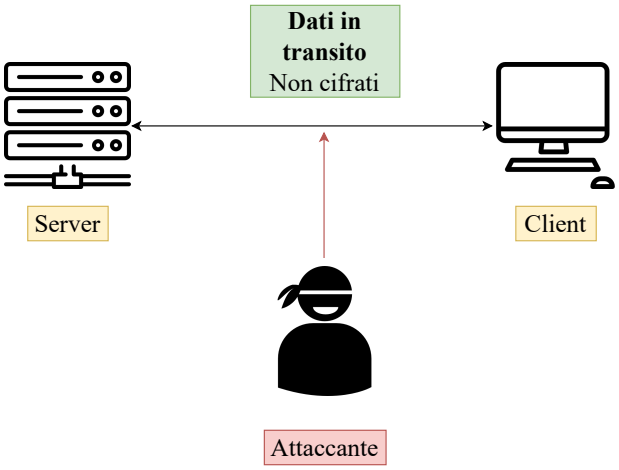
- **In transito (*In transit*)**;
- **In uso (*In use*)**;
- **A riposo (*At rest*)**.



2.1.1 Dati in transito

Come suggerisce la parola, si tratta dello stato che i dati acquisiscono nel momento in cui vengono trasmessi a qualcuno o a qualcosa, diventando così oggetto di un trasferimento di dati.

Occorre proteggere i dati in modo opportuno quando sono in questo stato in quanto un attaccante può leggerli, modificarli o eliminarli durante la connessione. Le strategie più utilizzate oggi per proteggere i dati in transito riguardano **TLS** [16] o **SSH** [17].



2.1.2 Dati in uso

I dati presenti in questo particolare "stato" vengono definiti *in uso* nel momento in cui vengono caricati in memoria di un qualsiasi dispositivo elettronico al fine di essere "letti e interpretati" da applicativi presenti su quello stesso sistema.

Ovviamente, si tratta di una condizione che può essere resa più o meno attaccabile a seconda delle particolari modalità di caricamento e lettura utilizzate dal software e/o dalle componenti hardware e software del sistema utilizzato [15].

2.1.3 Dati a riposo

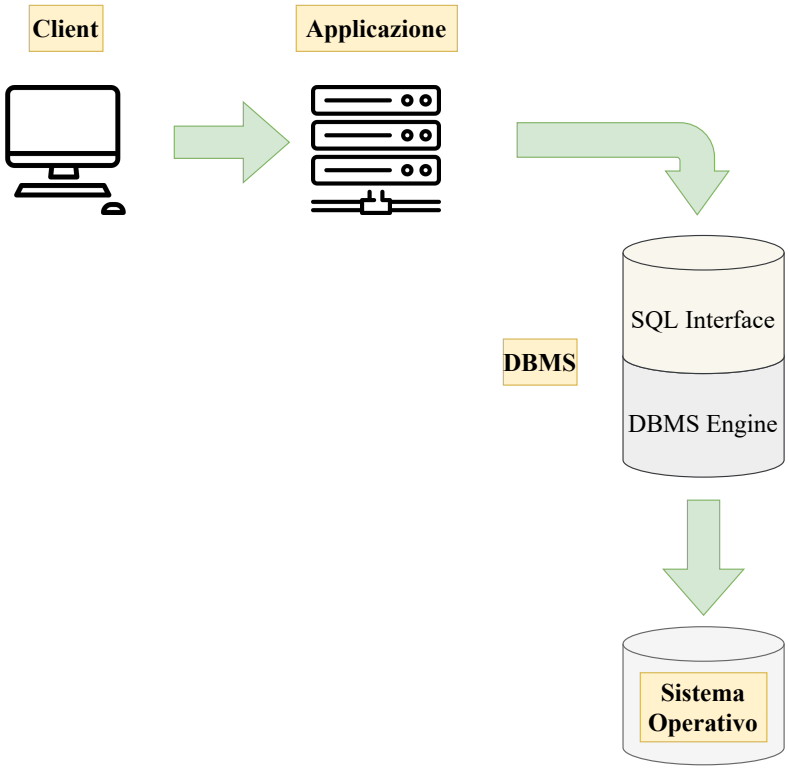
Si tratta dello stato iniziale (e spesso terminale) della stragrande maggioranza dei dati digitali. Sinteticamente, possiamo considerare a riposo tutti i dati memorizzati da qualche parte senza essere utilizzati da e/o trasmessi a altri utenti, programmi e applicativi, servizi di terze parti, strumenti per la riproduzione di media digitali, e così via. Tutti i dati memorizzati su qualsiasi unità di archiviazione locale o remota appartengono a questa categoria. Più in generale, possiamo ragionevolmente affermare che un dato è *a riposo* in tutte le situazioni in cui non sta venendo utilizzato e/o trasmesso [15].

Questo lavoro di tesi si concentrerà sulle tecniche di sicurezza per proteggere questi tipi di dati.

2.2 Tecniche di cifratura per dati a riposo

È possibile cifrare i dati a riposo a diversi livelli [18, 13]:

- **Livello *Sistema Operativo***;
- **Livello *DBMS***:
  - SQL Interface.
  - DBMS Engine;
- **Livello *Applicazione***;
- **Livello *Client***.

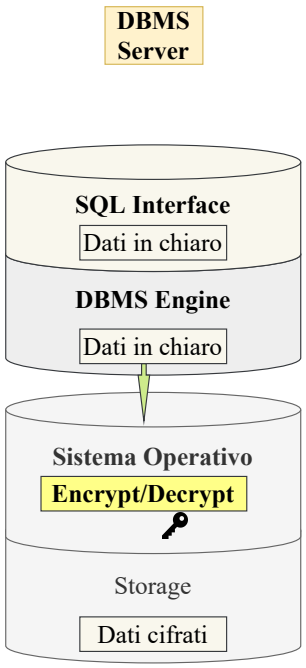


2.2.1 Cifratura a livello *Sistema Operativo*

Funzionamento e obiettivi

È importante notare che le tradizionali tecniche di cifratura di un database normalmente consistono nel cifrare e decifrare il contenuto di un database. I database vengono gestiti da *DBMS* che lavorano sopra ad un sistema operativo. Questo causa una potenziale minaccia e problema di sicurezza, infatti il sistema operativo potrebbe essere potenzialmente vulnerabile.

A questo livello, le pagine sono cifrate/decifrate dal sistema operativo quando vengono scritte/lette dal disco.



Vantaggi

- Trasparente agli utenti e alle applicazioni, i software vendor non devono personalizzare e riadattare gli applicativi o modificare i propri processi di business [13];
- Stabilisce forti controlli utilizzando delle specifiche policy.

Svantaggi

- Chi ha i privilegi di amministratore può leggere i dati cifrati;
- Key Management;
- Poca flessibilità nella scelta della granularità del dato [1, 13].
- La cache del database, che contiene un gran numero di copie di pagine del disco (per aumentare le performance), viene tenuta in chiaro e perciò vulnerabile agli attacchi su *Dati in Uso* [13].

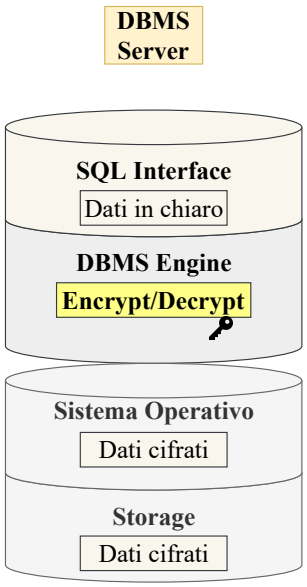
2.2.2 Cifratura a livello *DBMS Engine*

Si tratta di una cifratura simile al livello del sistema operativo, le pagine vengono cifrate/decifrate quando vengono lette/scritte dal disco. Ma in questo caso le operazioni crittografiche vengono eseguite dal DBMS Engine.

Il *DBMS Engine* è il modulo centrale di un DBMS che gestisce le operazioni di input/output e il data storage nel database. Molti DBMS offrono delle funzionalità di cifratura integrate.

Ogni volta che delle informazioni vengono inserite o lette nel database queste vengono rispettivamente cifrate o decifrate in modo del tutto automatico. Per questo motivo questa soluzione non comporta nessuna modifica alle esistenti applicazioni.

Gli articoli 33 [19] e 34 [20] del GDPR definiscono le linee guida per quanto riguarda la notifica al supervisore e al proprietario dei dai nel caso di *Data breach* [21]. La comunicazione al proprietario dei dati può non avvenire se "il titolare del trattamento ha messo in atto le misure tecniche e organizzative adeguate di protezione e tali misure erano state applicate ai dati personali oggetto della violazione, in particolare quelle destinate a rendere i dati personali incomprensibili a chiunque non sia autorizzato ad accedervi, quali la cifratura". Visto che il furto dei dati rappresenta un danno di immagine per l'azienda, l'utilizzo della cifratura a questo livello (e anche a livello client) può prevenire questo tipo di problemi.



La **Transparent Data Encryption** è la tecnica più utilizzata [18].

Transparent Data Encryption: *Funzionamento e obiettivi*

Transparent Data Encryption (spesso abbreviata come TDE) è usata per cifrare e decifrare dati e file di log. La cifratura utilizza una **Database Encryption Key (DEK)** che è una chiave simmetrica protetta da un certificato memorizzato nel DBMS. La TDE è stata introdotta da Microsoft SQL Server 2008 per cifrare il contenuto del database.

Si tratta di una tecnologia che può essere utilizzata per fornire alti livelli di sicurezza delle colonne, tabelle e tablespace.

La TDE cifra i dati prima che questi vengano scritti sul disco e li decifra prima di essere ritornati all'applicazione. I processi di cifratura/decifratura vengono eseguiti nel DBMS Engine, in modo completamente trasparente alle applicazioni e agli utenti [22]. Inoltre, assicura che i dati a riposo non possano essere letti da individui malintenzionati che hanno l'intenzione di rubarli fisicamente o che hanno ottenuto i privilegi sulla macchina che ospita il DBMS. Infatti anche se i file vengono compromessi o rubati tutti i dati rimangono illeggibili, solo gli utenti autorizzati possono farlo. Quindi solo chi possiede la chiave di cifratura è in grado di comprendere i dati. I dati che non possono essere letti sono inutili riducendo così l'incentivo per il furto.

La TDE effettua operazioni crittografiche a livello I/O all'interno del DBMS Engine. Inoltre è possibile aggiungere ai dati cifrati del *sale* per evitare attacchi crittografici sul testo cifrato, come gli attacchi a dizionario [1, 23]. Un dizionario è una mappa chiave-valore dove la chiave corrisponde ad una stringa (generalmente una password) mentre il valore è l'hash della stringa corrispondente, calcolato con uno specifico algoritmo. Un attacco a dizionario consiste nel prendere come input un hash (corrispondente all'hash di una password) e controllare nel dizionario se esiste una coppia avente come valore l'hash di input, la password corrispondente è la chiave associata a quel valore. Il sale è una sequenza casuale di bit che viene aggiunto ai dati originali prima di calcolare l'hash rendendolo più casuale. Il sale viene generato casualmente ogni volta che viene ricevuto o calcolato un dato. Perciò un attaccante dovrebbe calcolare il dizionario per ogni possibile valore del sale e se questo ha grandi dimensioni rende infattibile questo tipo di attacco.

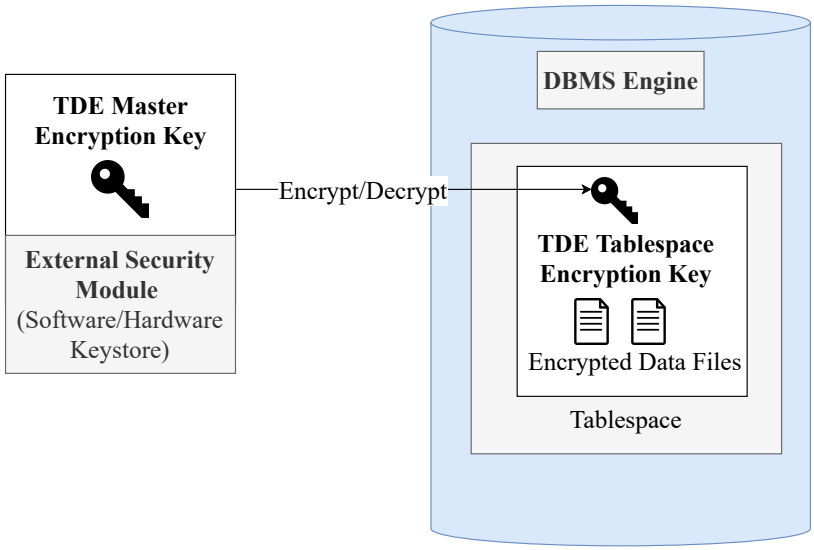


Figura 2.1: Transparent Data Encryption con granularità *Tablespace*

La maggior parte delle implementazioni permettono di cifrare i dati (a livello colonna o tablespace) utilizzando una chiave chiamata *TDE Encryption Key* mantenuta dal DBMS stesso. Tale chiave viene cifrata utilizzando una *TDE Master Encryption Key* che viene mantenuta in un modulo sicuro esterno [1].

[1] Questi concetti verranno approfonditi nei capitoli successivi.

Transparent Data Encryption: *Vantaggi*

- Non c'è bisogno di creare tabelle ausiliarie, trigger o viste per decifrare i dati per gli utenti autorizzati. I dati vengono decifrati in modo trasparente. Un'applicazione che processa dati sensibili può usare la TDE per fornire una forte cifratura dei dati senza cambiare l'applicativo;
- Essendo i dati decifrati in modo trasparente, gli utenti che utilizzano il database possono non essere consapevoli che i dati a cui stanno accedendo sono memorizzati in forma cifrata;
- Flessibilità nella scelta della granularità del dato da cifrare [1];
- Burocrazia e immagine. Nel caso di data breach in cui i dati rubati siano cifrati non c'è bisogno di notificare il cliente, come detto negli articoli 33 e 34 del GDPR citati nella Sezione 2.2.2.

Transparent Data Encryption: *Svantaggi*

- Key Management [2];
- Performance overhead;
  Le operazioni crittografiche aggiungono del calcolo computazionale al DBMS rendendolo tendenzialmente più lento rispetto ad una soluzione che non effettua operazioni crittografiche.
- Storage overhead;
  Quando i dati vengono cifrati vengono memorizzati anche MAC, IV aumentando la dimensione dei dati memorizzati. Occorre gestire ciò in modo opportuno.
- Vulnerabile a SQL Injection, Cross-site Scripting , bug nel codice che generano vulnerabilità non previste, attacchi al sistema operativo e attacchi di privilege escalation. Ad esempio nel caso dell'SQL Injection quando viene effettuata una query i dati ritornati vengono, in modo trasparente, decifrati.

Implementazioni esistenti

- Oracle SQL Transparent Data Encryption (proprietaria) [24];
- Microsoft SQL Server Transparent Data Encryption (proprietaria) [25];
- MySQL Transparent Data Encryption (open-source) [26];
- MariaDB Transparent Data Encryption (open-source) [27].

2.2.3 Cifratura a livello *SQL Interface*

Funzionamento e obiettivi

La **SQL Interface** fornisce meccanismi, funzioni e librerie per elaborare i dati nel DBMS, come procedure, viste e trigger. Operare a questo livello permette ad un programmatore di essere totalmente flessibile riguardo le operazioni che deve compiere sui dati.

A questo livello è possibile connettere al sistema dei *Plug-In* che permettono di cifrare e decifrare i dati installando un opportuno modulo sul DBMS. Cifrare i dati a questo livello permette al DBMS Engine di ricevere i dati già cifrati e inoltre permette di selezionare la granularità che si vuole ottenere beneficiando in prestazioni [18, 28, 13].

[2] È una problematica comune a tutte le tecniche analizzate. Verrà affrontato in dettaglio nel Capitolo 3.

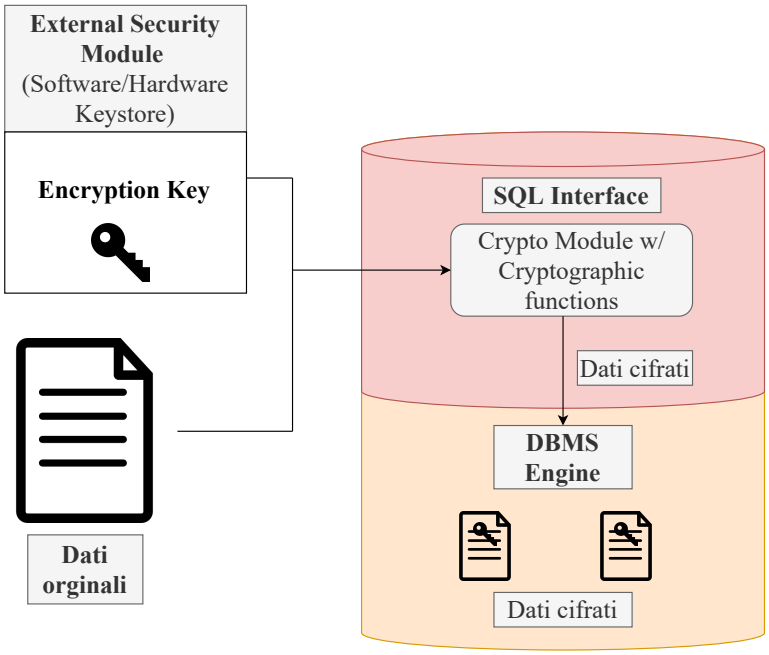


Figura 2.2: Cifratura a livello SQL Interface

Vantaggi

- *Performance.* Soluzioni ad-hoc possono essere adottate per diminuire l'impatto sulle performance delle operazioni crittografiche. Teoricamente si può cerca di ottimizzare la query in modo da ottimizzare le performance. Però, ciò non succede nella realtà, come mostrato nei risultati del Capitolo 5;
- *Flessibilità sia su DBMS commerciali e DBMS open-source.* Flessibilità in termini di granularità e gestione delle chiavi. Si possono cifrare dati diversi con diverse chiavi o con la stessa chiave. Massima libertà al programmatore.

Svantaggi

- Key Management esplicito. L'utente deve implementare un corretto sistema di Key Management in quanto è lui stesso il proprietario delle chiavi crittografiche inserite nelle query;
- Non è completamente trasparente. Infatti si può pensare di utilizzare dei trigger per automatizzare le operazioni crittografiche per i diversi statement SQL ma la maggior parte dei vendor supportano ciò solo parzialmente;
- Aumenta la complessità del codice (aumentando la probabilità di inserire nuove vulnerabilità) in quanto l'utente deve scrivere delle nuove query che contengono le chiamate alle funzioni crittografiche. È responsabilità dell'utente gestire correttamente le operazioni crittografiche;
- La cifratura avviene all'esterno del DBMS Engine e quindi alcuni meccanismi (indexing, foreign keys) potrebbero non funzionare correttamente [13].

Implementazioni esistenti

- Oracle DBMS_CRYPTO (proprietaria) [29];
- Microsoft Cryptographic functions in Transact-SQL (proprietaria) [30];
- PostgreSQL PGCRYPTO (open-source) [31];
- MariaDB Security Functions (open-source) [32];
- MySQL Security Functions (open-source) [33];

2.2.4 Cifratura a livello *Applicazione*

La cifratura a questo livello viene eseguita nell'applicazione che genera/riceve i dati. Le funzioni di cifratura e decifratura vengono eseguite esternamente al DB lasciandolo più leggero (il database non effettua operazioni di cifratura/decifratura dei dati).

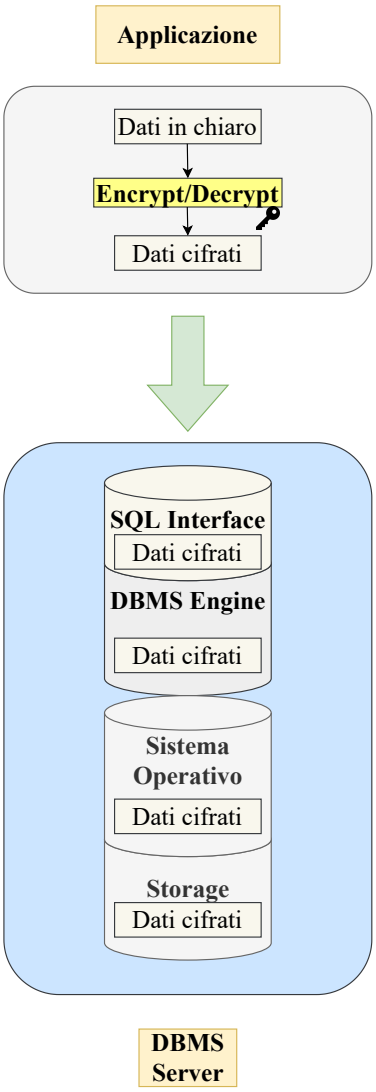


Figura 2.3: Cifratura a livello *Applicazione*

Vantaggi

- Separazione delle chiavi di cifratura dai dati cifrati memorizzati nel database, la chiave non esce mai dall'applicazione;
- Flessibilità nella scelta della granularità dei dati;
- Flessibilità nella Key Management.

Svantaggi

- La cifratura a questo livello comporta modifiche dell'applicazione;
- Non è possibile sfruttare i meccanismi del DBMS per l'ottimizzazione delle query (per aumentare le performance), come ad esempio gli indici (per ottimizzare le operazioni di ricerca);
- Implementare il supporto per la cifratura a livello applicazione ad un sistema legacy comporta un enorme utilizzo di tempo e risorse;

- A seconda della granularità, l'applicazione potrebbe dover recuperare un insieme di dati più grande di quello concesso all'utente effettivo, aprendo così una breccia nella sicurezza. Infatti, l'utente (o qualsiasi aggressore che abbia accesso alla macchina su cui gira l'applicazione) può violare l'applicazione per accedere a dati non autorizzati [13].

Implementazioni esistenti

Trattandosi di cifratura a livello applicazione, vengono utilizzate delle librerie crittografiche sviluppate per i più comuni linguaggi di programmazione (OpenSSL per C/C++, Java Cryptography o pycryptodome per Python) e framework (.NET o Java Spring).

2.2.5 Cifratura *Client-side*

Funzionamento e obiettivi

La **Client-side Encryption** è una tecnica crittografica che consiste nel cifrare i dati client-side ovvero sul client prima di essere trasmessi ad un server online.

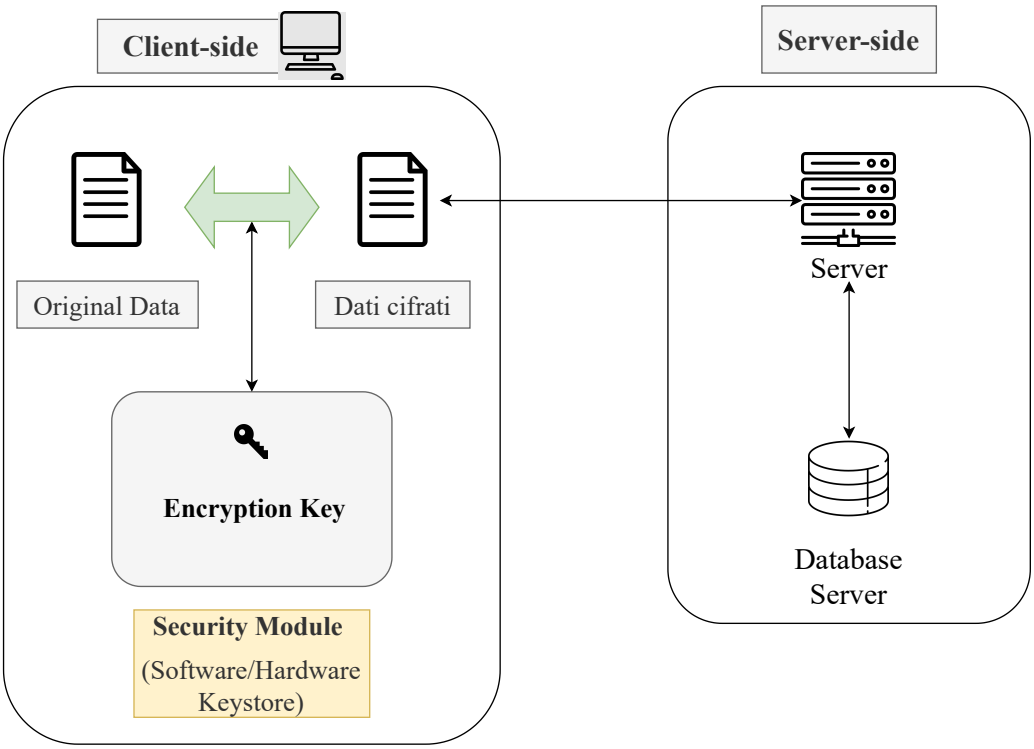


Figura 2.4: Cifratura *Client-Side*

Quando si cifrano i dati nel database, occorre fare le seguenti considerazioni: come influenzerà la ricerca dei dati? Il DBMS sarà in grado di determinare quali dati si cercano? E come la crittografia influenzerà gli indici e le prestazioni di ricerca?

Questo non è un problema con la crittografia che è trasparente al DBMS, come la crittografia a livello DBMS Engine. Infatti schemi in cui il DBMS ha accesso alla chiave di crittografia e può decifrare i dati non dovrebbe essere un problema. Il problema sorge nel caso in cui le operazioni crittografiche siano eseguite sull'applicazione o sul client. In questo caso il DBMS non ha conoscenza riguardo i dati memorizzati.

Tutte le query nel DBMS si basano su diverse operazioni di base. I sistemi SQL si basano principalmente sull'algebra relazionale e le sue operazioni come l'unione degli insiemi, la differenza, la proiezione, la selezione, ecc. I sistemi NoSQL si basano principalmente sugli array associativi e

le loro operazioni come costruzione, ricerca, aggiunta. Se il DBMS fosse in grado di eseguire tutte queste operazioni su dati criptati, sarebbe possibile eseguire anche intere query su dati cifrati.

La soluzione che permette di fare questo è chiamata *crittografia omomorfica*. La crittografia completamente omomorfica (FHE) è un sistema di crittografia che permette di fare operazioni di calcolo arbitrarie su dati cifrati e ottenere un risultato cifrato. L'idea di base di idea è nata nel 1978. A quel tempo non era certo se fosse possibile creare un FHE. Moltissimi criptosistemi parzialmente omomorfi sono stati creati da allora. Questi permettevano di eseguire diverse operazioni sui dati criptati, ma il resto delle operazione doveva essere fatta sul testo in chiaro. La prima costruzione di FHE è stata introdotta nel 2009. Da allora sono stati fatti molti miglioramenti e usi di questi criptosistemi. Di fatto non è ancora utilizzata perché questa comporta un'enorme potenza di calcolo.

Un altro approccio a questo problema è chiamato *Searchable Symmetric Encryption (SSE)*. La SSE permette al client di esternalizzare la memorizzazione dei suoi dati su un server remoto, pur mantenendo la capacità di cercare selettivamente su di esso [34].

Questi approcci ci permettono di separare la fornitura, l'amministrazione e l'interrogazione dei dati a entità diverse senza difetti di sicurezza. Il DBMS può ricevere i dati criptati dal fornitore e amministrarli. Con l'amministrazione dei dati si intende l'archiviazione, la ricerca ma anche far rispettare le regole di accesso. Con tutto ciò realizzato, il DBMS è in grado di accettare una query dal richiedente e restituire esattamente i dati che sono stati richiesti. La decifratura dei dati viene eseguita dal richiedente. Questo approccio non rivela i dati al DBMS, e inoltre non rivela alcun dato aggiuntivo al richiedente.

Crittografare i dati presso il fornitore e memorizzarli nel database solo in forma criptata, rende anche i dati più sicuri in caso di attacco. Anche se l'attaccante prende il controllo del DBMS, non è in grado di decifrare i dati. Lo svantaggio di questo è un overhead legato alle performance ed è stimato essere tra il 30%-500% rispetto ai database SQL standard [35].

Vantaggi

- Sicurezza. La chiave di cifratura non è disponibile al service provider rendendo più difficile o impossibile la decifratura dei dati da parte di entità non autorizzate;
- Flessibilità nello scegliere la granularità dei dati da cifrare.
- Burocrazia e immagine. Nel caso di *data breach* in cui i dati rubati siano cifrati non c'è bisogno di notificare il cliente, come detto negli articoli 33 e 34 del GDPR citati nella Sezione 2.2.2.

Svantaggi

- Aumenta la complessità del codice e dell'applicazione client;
- Dal momento che la cifratura parte dalla generazione dei dati si hanno basse performance e le operazioni non possono essere indicizzate con semplicità.

Implementazioni esistenti

- Microsoft Always Encrypted [36].

2.3 Granularità dei dati cifrati nei database

2.3.1 Intero table-space

Questo metodo consiste in cifrare interi table-space. Ogni table-space ha una chiave univoca che protegge tutto il suo contenuto [37, 18]. Questa modalità è supportata dalla TDE e dalle soluzioni di client-side encryption, infatti è possibile programmare il sistema che implementi la cifratura a livello client in modo da supportare questa granularità.

Vantaggi

- Semplice da implementare;
- Basso impatto sulle performance. Più efficiente della cifratura a singola colonna;
- Protegge tutte le informazioni nel table-space inclusi gli schemi, procedure e dati.

Svantaggi

- Una singola chiave per l'intero tablespace. Ciò rende più semplice la gestione della chiave eliminando però la possibilità di cifrare i dati nel tablespace con chiavi diverse.

2.3.2 Singola colonna

La **Column-level encryption** permette di cifrare delle singole colonne del database. È importante notare che la granularità della crittografia a livello di colonna fa emergere punti di forza e di debolezza rispetto alla crittografia di un intero database. Innanzitutto, l'abilità di cifrare delle colonne individualmente permette alla column-level encryption più flessibilità rispetto alla TDE. Inoltre è possibile usare un'unica e separata chiave di cifratura per ogni colonna. Il principale svantaggio di questa tecnica è relativo alla velocità. Infatti cifrare colonne separate con differenti chiave può causare una perdita di performance [18, 37, 38]. Come la granularità ad intero table-space anche questa modalità è supportata dalla TDE e dalla client-side encryption.

Vantaggi

- Maggior flessibilità nello scegliere quali pezzi di dati cifrare;
- Diverse colonne possono essere cifrate con chiavi diverse;
- Supporto delle implementazioni della Transparent Data Encryption fornite dai DBMS. Ciò permette un utilizzo e configurazione semplice e supportata dal DBMS.

Svantaggi

- Alto impatto sulle performance (5%-6% in media più lento nell'accesso/aggiornamento rispetto a colonne in chiaro). Più alto è il numero di colonne cifrate più alto è l'impatto sulle performance;
- Limitazioni e perdita di performance sulle tecniche di ricerca del database;
- Non protegge le "proprietà intellettuali" del database (schemi, viste).

2.3.3 Singolo campo

In questo caso ciascuna cella può essere cifrata con una sua particolare chiave. Questa soluzione può essere appropriata quando si vuole ottenere un alto livello di protezione a costo di pagare un enorme costo in termini di performance. Per questo proposito si stanno svolgendo dei lavori sperimentali (*Homomorphic Encryption*) per aggiornare le operazioni sul database (come ricerca o operazioni aritmetiche) in modo che operino senza la necessità di decifrare i dati. Queste tecniche sono deboli ma permettono di verificare l'uguaglianza senza decifrare i dati [37, 18]. Questa modalità è supportata dalle API crittografiche SQL Interface (ad esempio DBMS_CRYPTO) e dalla client-side encryption. La soluzione di cifratura a livello client sviluppata nel Capitolo 6 è stata programmata per supportare questa granularità.

Vantaggi

- Alta flessibilità nello scegliere quali campi di un dato cifrare;
- Ogni campo può essere cifrato con chiavi diverse.

Svantaggi

- Altissimo impatto sulle performance. Può far risultare le operazioni sul database molto più lente rispetto alle precedenti soluzioni;
- Difficile compatibilità con la Transparent Data Encryption. La maggior parte dei vendor che implementano soluzioni di TDE nei loro prodotti non offrono supporto a questa soluzione. Uno dei motivi di questa scelta è legato al fatto che la cifratura a singolo campo richiede una Key Management più complessa in quanto il numero di chiavi coinvolte è maggiore delle altre soluzioni. Inoltre, dato che la TDE cifra e decifra delle singole pagine su disco, cifrare e decifrare un singolo campo comporterebbe un'enorme penalità nella dimensione dello storage in quanto in ogni pagina occorrerebbe memorizzare un solo dato. Infine i DBMS supportano questa cifratura utilizzando delle specifiche API crittografiche a livello SQL Interface.
- Non protegge le "proprietà intellettuali" del database (schemi, viste).

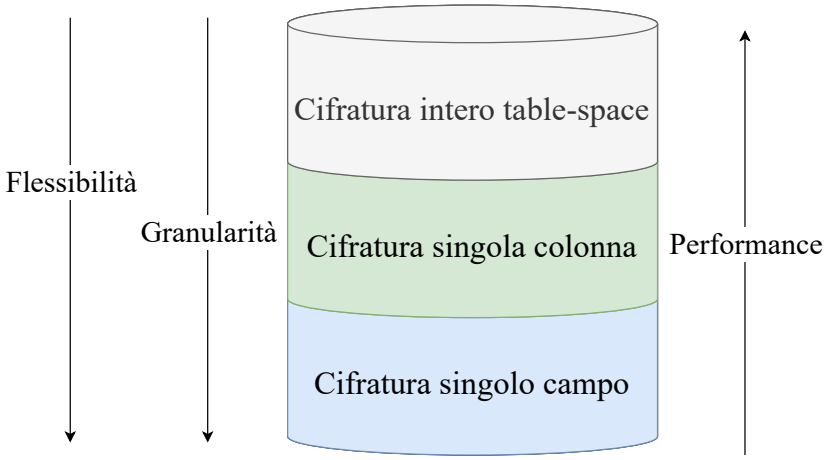


Figura 2.5: Confronto granularità.

2.4 Algoritmi di cifratura

Indipendentemente dalla strategia di cifratura (livello e granularità), la sicurezza dei dati cifrati dipende dall'algoritmo di cifratura, dalla dimensione della chiave di cifratura e dalla sua protezione. Inoltre anche adottando un forte algoritmo, come AES, il testo cifrato potrebbe essere compromesso e violato se non viene utilizzato una modalità sicura. Per esempio, se l'algoritmo di crittografia è implementato in modalità Electronic Codebook (ECB), blocchi di testo identici in chiaro sono cifrati in blocchi identici, rivelando così schemi ripetitivi. Nel contesto del database, gli schemi ripetitivi sono comuni in quanto molti record potrebbero avere gli stessi valori di attributi, quindi bisogna fare molta attenzione quando si sceglie la modalità. Inoltre, soluzioni semplici che possono funzionare in altri contesti possono fallire nel contesto di un DBMS.

Tutte le specificità del contesto del database dovrebbero essere prese in considerazione per guidare la scelta di un algoritmo di crittografia adeguato e della sua modalità di funzionamento. Inoltre, la protezione dovrebbe essere abbastanza forte poiché i dati possono essere validi per un tempo molto lungo (diversi anni). Quindi occorre scegliere un algoritmo di crittografia e modalità di funzionamento all'avanguardia [1].

Capitolo 3

Key Management

Come visto nel precedente capitolo la **Key Management**, è una problematica comune a tutte le diverse tecniche di cifratura. In questo capitolo si approfondirà questa tematica.

Nella **Database Encryption** il sistema deve gestire la memorizzazione, lo scambio e il ciclo di vita delle chiavi crittografiche. Questo processo è chiamato *Key Management*. Se le chiavi di cifratura non vengono correttamente gestite i dati sensibili possono essere divulgati e trafugati. Inoltre, se il *Key Management System*[1] perde o elimina la chiave, le informazioni che sono state cifrate con quella chiave sono essenzialmente "perse". Perciò la *Key Management* può rappresentare un punto vulnerabile se non correttamente gestita e quindi rappresenta una problematica da prendere seriamente in considerazione.

Con l'aumentare del numero di applicazioni che un'azienda utilizza aumenta anche il numero di chiavi che devono essere memorizzate e gestite. È quindi necessario stabilire un modo in cui le chiavi di tutte le applicazioni possono essere gestite attraverso un unico canale, noto anche come *Enterprise Key Management*. Soluzioni di *Enterprise Key Management* sono vendute da un gran numero di fornitori del settore IT. Essenzialmente questi sistemi forniscono una soluzione di gestione centralizzata delle chiavi che consente agli amministratori di gestire tutte le chiavi di un sistema attraverso un hub[2]. Si può quindi affermare che l'introduzione di soluzioni di gestione delle chiavi aziendali ha il potenziale di ridurre i rischi associati alla *Key Management* nel contesto della **Database Encryption** nonchè per ridurre i problemi logistici che sorgono quando molti individui tentano di condividere manualmente le chiavi.

Un *Key Management System* robusto deve gestire correttamente [39]:

- **Key Lifecycle**: Pre-operatività, operatività, post-operatività e distruzione. Queste fasi verranno definite e approfondite nella sezione 3.2;
- **Accesso fisico alla chiave**;
- **Accesso logico alla chiave**;
- **Determinazione degli utenti e ruoli che possono accedere alla chiave**.

Attualmente la problematica della Key Management ha delle sfide da risolvere che vanno oltre alla crittografia e che in un certo senso coinvolgono aspetti dell'ingegneria sociale come lo *user training*, *interazioni e coordinamento tra organizzazioni* in contrasto ai comuni processi matematici che possono essere automatizzati.

Infine come motivazione aggiuntiva all'importanza della problematica della Key Management si riporta una lista contenente gli standard nazionali e internazionali (ciascuno contenente dei requisiti) a cui una organizzazione deve essere conforme.

---

[1] Si tratta del sistema che gestisce le chiavi crittografiche.

[2] Nei prossimi capitoli verranno approfondite specificamente le soluzioni proposte dai vari vendor.

PCI DSS

Payment Card Industry Data Security Standard (PCI DSS) è un insieme di regolamenti per transazioni sicure basate su carte di credito, debito e prepagate. PCI DSS richiede che il venditore protegga le informazioni riservate del possessore della carta usando delle buone pratiche di sicurezza [40].

Nella sezione 3.5 del PCI DSS, tutte le organizzazioni che processano, memorizzano o trasmettono informazioni di una carta dovrebbero documentare ed implementare procedure per proteggere le chiavi usate per la memorizzazione sicura dei dati della carta. Queste procedure includono:

- Mantenere una descrizione documentata dell'architettura crittografica usata per proteggere i dati;
- Restringere l'accesso alle chiavi crittografiche ad un limitato numero di utenti;
- Memorizzare le chiavi di cifratura rispettando le due seguenti accortezze:
  - Cifrare la Data Encryption Key con una Master Key Encryption;
  - Memorizzare all'interno di un dispositivo crittograficamente sicuro.

Nella sezione 3.6 il PCI DSS richiede una piena documentazione e implementazione dei processi e procedure per la Key Management: Questo deve includere:

- Generazione di forti chiavi crittografiche;
- Distribuzione sicura delle chiavi;
- Memorizzazione sicura delle chiavi;
- Definizione del Crypto Period per tutte le chiavi. Il Crypto Period viene illustrato nella Sezione 3.2;
- Definizione delle procedure di ritiro e distruzione delle chiavi.

HIPAA TECH

Health Insurance Portability and Accountability Act (HIPAA) e Health Information Technology for Economic and Clinical Health (HITECH) sono leggi statunitensi che cercano una maggiore adozione e un uso significativo della Key Management per la protezione delle informazioni sanitarie. Entrambie stabiliscono delle linee guida e regolamenti per una corretta sicurezza dei dati e degli Electronic Protected Health Information (ePHI) [41]. La conformità a HIPAA Security Rules e HIPAA Privacy Rules per ePHI richiedono l'uso di tecnologie e best practices per dimostrare una forte adozione di misure di sicurezza per la Key Management [42].

SOX

La legge statunitense Sarbanes-Oxley (SOX) è stata approvata per proteggere gli investitori dalla possibilità di attività contabili fraudolente da parte delle società. Questa legge ha imposto severe riforme per migliorare le informazioni finanziarie delle società e prevenire frodi contabili. Le sezioni 302, 304 e 404 obbligano le organizzazioni a costruire e mantenere un sistema di Data Security e Key Management usati per salvaguardare i loro dati riservati da frodi [43].

Cloud Security Alliance

Cloud Security Alliance non è una agenzia governativa e quindi non in grado di imporre multe per la non conformità dei loro standard ma è una organizzazione no-profit per i cloud vendor, utenti ed esperti di sicurezza e la sua missione è quella di promuovere l'uso di best practice per fornire sicurezza all'interno del Cloud Computing [44].

Attualmente ha più di 80 mila membri perciò la conformità ai suoi standard è nell'interesse della maggior parte delle organizzazioni mondiali. Cloud Security Alliance ha pubblicato un documento, "Security Guidance For Critical Areas of Focus In Cloud Computing" [45], per aiutare i venditori e i clienti ad ottenere una maggiore sicurezza delle applicazioni in ambienti cloud. La parte che riguarda la Key Management è la sezione "Domain 11 - Encryption and Key Management".

In questa sezione ci sono i tre punti principali della Key Management per CSA:

- **Key Store Sicuri.** I dispositivi di Key Storage devono essere protetti come ogni altro dato sensibile. Devono essere protetti in storage, in transito e in backup. Un uso improprio del Key Storage potrebbe compromettere tutti i dati cifrati;
- **Accesso al Key Storage.** Gli accessi al Key Storage devono essere limitati alle entità che hanno specificamente bisogno delle chiavi individuali. Devono inoltre esserci delle politiche che amministrano il Key Storage e che usano la separazione dei ruoli e funzioni per aiutare l'access control;
- **Backup e ripristino della chiave.** La perdita di una chiave immediatamente si traduce in una perdita dei dati che la chiave protegge. Perciò devono essere implementate sicure misure di backup e ripristino.

EU GDPR

Negli articoli 32 e 34 del GDPR (General Data Protection Regulation) si definisce una alta priorità nel proteggere i dati a riposo con la cifratura. Dato che la Encryption Key Management è parte della strategia della cifratura dei dati a riposo, deve anch'essa avere alta priorità nella conformità con le leggi dell'Unione Europea [46].

CAP 486

L'ordinanza CAP 486 di Hong Kong sui dati personali richiede che vengano prese tutte le misure pratiche per garantire che le informazioni di identificazione personale, detenute da un proprietario dei dati, siano protette da accessi non autorizzati o accidentali. Fornisce indicazioni su come dovrebbero essere immagazzinati i dati e come comportarsi in caso di data breach [47].

APPI

Act on the Protection of Personal Information del Giappone contiene policy che sono delle linee guida, ma non leggi, che governano la protezione delle informazioni personali. Richiede tutte le misure necessarie per prevenire perdite e danni [48].

PA 1988 & PA 2000

Privacy Act del 1988 e il Privacy Amendament Act del 2000 del governo australiano sono leggi che riguardano la Data Protection. Tutte le organizzazioni devono costruire e mantenere dei sistemi per proteggere le informazioni personali in un database da violazioni. Un organizzazione deve inoltre distruggere permanentemente le informazioni riservate se non vengono più usate da tempo [49].

3.1 Funzionamento di un *Key Management System*

Innanzitutto occorre definire [39]:

Data Encryption Key (DEK). Chiave che ha la funzione di cifrare e decifrare i dati.

Key Encryption Key (KEK). Chiave che ha la funzione di cifrare e decifrare la DEK.

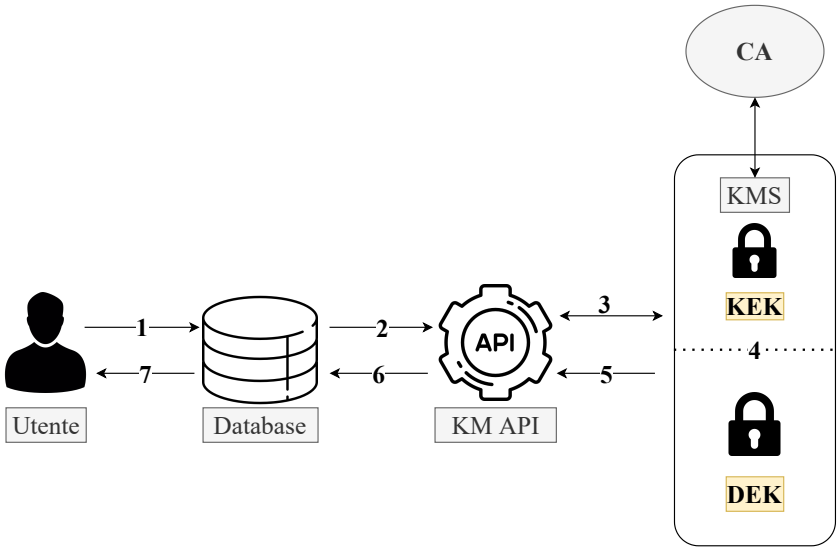
Key Management Application Program Intergace (KM API). Interfaccia progettata per lo scambio sicuro delle chiavi di cifratura tra il *Key Management Server* e l'*Utente*.

Certification Authority (CA). Entità che crea i certificati a partire da una chiave pubblica, verifica i certificati ed effettua altre funzioni di *PKI* (Public Key Infrastracture).

Transport Layer Security (TLS). Protocollo crittografico che fornisce proprietà di sicurezza come autenticazione (anche *Mutua*), integrità e confidenzialità per i *Dati in transito* su una rete.

Key Management System (KMS). Sistema che contiene il *Key Management Software*.

Il processo segue questo schema:



1. L'utente richiede l'accesso a dati cifrati.
2. Il database invia un messaggio *DEK Retrieval Request* al client (KM API) per ottenere la chiave di (de)cifratura.
3. Il client (KM API) e il KMS verificano i loro certificati (mutua autenticazione) e stabiliscono una connessione TLS.
4. Il KMS decifra la DEK richiesta usando la KEK.
5. Il KMS invia la DEK al client(KM API) utilizzando TLS per proteggere la chiave durante la comunicazione.
6. Il client(KM API) invia la DEK al database.
7. Il database ritorna i dati decifrati all'utente.

Seguendo questo schema si possono individuare i tre step principali della Key Management:

Key Exchange

Come visto la DEK scambiata tra il client e il KMS deve essere opportunamente protetta. Le soluzioni maggiormente adottate per lo scambio delle chiavi sono:

- Diffie-Hellman;
- Public Key Infrastructure (PKI): su cui si basa TLS;
- Web of Trust: su cui si basa OpenPGP;
- Password-authenticated key agreement;
- Quantum Key Exchange;
- Smart-card;

Key Storage

Le chiavi (DEKs e KEKs) devono essere memorizzate in sistemi sicuri usando tecnologie come *Hardware Security Module* [3] o *Trusted Execution Environment* (ad esempio Intel SGX) in modo da verificare l'integrità della chiave memorizzata.

Key Use

Il problema principale è la durata del tempo di utilizzo di una chiave e quindi la frequenza di sostituzione. Alla base di ciò c'è il principio della **Key Rotation**.

[3] Verrà approfondito successivamente in questo capitolo.

3.2 Ciclo di vita delle chiavi

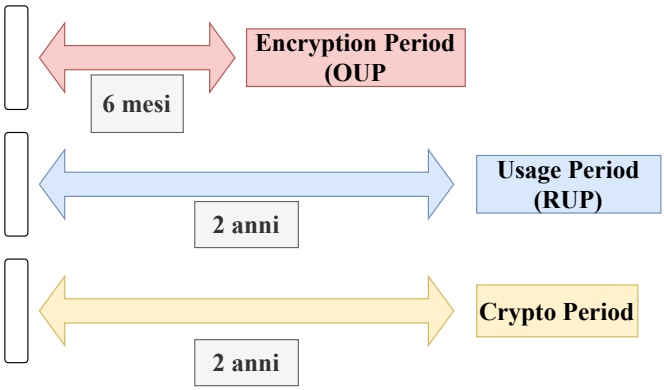
Il ciclo di vita delle chiavi di cifratura definito dal *NIST* [4] è composto dalle seguenti fasi [39, 50]:

1. **Pre-operatività.** Le chiavi non sono ancora disponibili per le normali operazioni crittografiche;
2. **Operatività.** Le chiavi sono disponibili e sono in normale uso;
3. **Post-operatività.** Le chiavi non sono più in uso ma è ancora possibile accedervi;
4. **Eliminazione.** Le chiavi non sono più disponibili.

Occorre definire l'**Operational Crypto Period** [39].

Un *Crypto Period* è il "lasso di tempo durante il quale una chiave specifica è autorizzata per l'uso" e nella sezione 5.3 della Guida del NIST [51] il Crypto Period è determinato combinando il tempo stimato durante il quale il dato è cifrato (*Originator Usage Period (OUP)*) e il tempo per cui sarà decifrato durante l'uso (*Recipient Usage Period (RUP)*).
Per esempio consideriamo il seguente caso :

- Un database viene cifrato e per i prossimi 6 mesi vengono aggiunti nuovi dati. Allora :
  - *OUP* è pari a 6 mesi.
- Per i prossimi 2 anni il database viene consultato da utenti autorizzati. Allora:
  - *RUP* è pari a 2 anni (e si interlaccia con il OUP).
- Pertanto il periodo di "cifratura" dei dati sarebbe pari a 2 anni e la chiave deve essere attiva durante questo lasso di tempo.



Ma dal momento che una organizzazione può ragionevolmente voler cifrare e decifrare gli stessi dati per anni, altri fattori possono entrare in gioco quando si valuta il *Crypto Period* ed occorre limitare [39]:

- *La quantità di informazioni protette da una singola chiave*;
- *Il periodo di esposizione se la chiave viene compromessa*;
- *Il tempo disponibile per provare a violare l'accesso fisico e logico da parte di un attaccante.*

Questa analisi può essere riassunta nelle seguenti domande chiave:

[4] National Institute of Standards and Technology.

- Per quanto i dati verranno usati?
- Come vengono utilizzati i dati?
- Quanti dati ci sono?
- Quanto sono sensibili i dati?
- Quanto danno sarà creato se i dati verranno esposti o le chiavi verranno perse ?

La regola generale da seguire è la seguente:

"*Con l'aumentare della sensibilità dei dati occorre diminuire la durata del ciclo di vita della chiave che li protegge favorendo la cosiddetta* ***Key Rotation***".

Sulla base delle precedenti osservazioni, la chiave di cifratura dovrebbe avere una "Active life" minore del tempo totale di utilizzo dei dati. Questo significa che c'è bisogno di archiviare le chiavi *non attive* e usarle solo per decifrare i dati. Una volta che i dati sono stati decifrati da una vecchia chiave occorre nuovamente cifrarli con la nuova chiave e quando la vecchia chiave non viene più usata per decifrare deve essere eliminata [39]. La Figura 3.1 mostra un esempio di applicazione del concetto di rotazione delle chiavi.

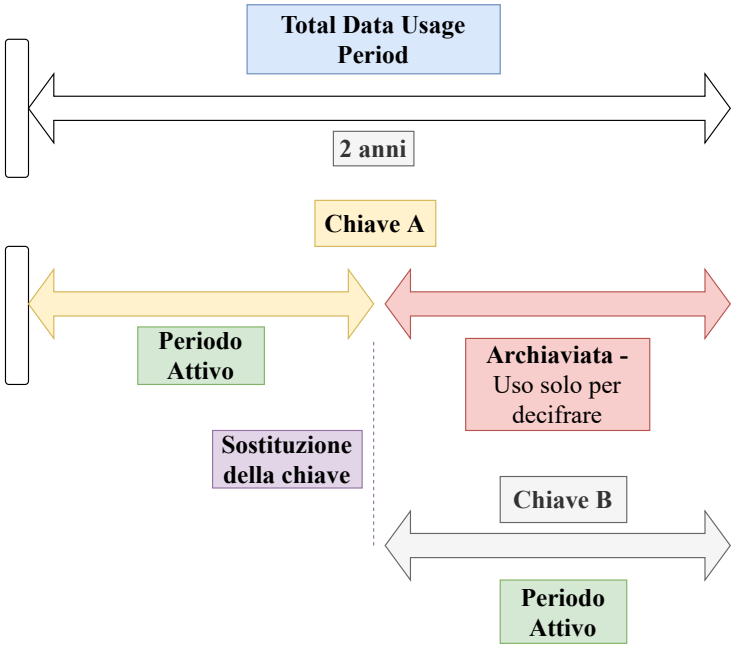


Figura 3.1: Key Rotation - Esempio.

La Figura 3.2 mostra l'intero ciclo di vita della chiave di cifratura [39].

Pre-Operation & Key Creation

In questa fase la chiave viene creata e memorizzata nel *Key Management Server*. Il *Key Manager* crea l'**Encryption Key** attraverso l'uso di un generatore di numeri casuali crittografico (sicuro) e memorizza la chiave (insieme ai metadati) nel Key Storage Database. I metadati memorizzati con la chiave includono [39]:

- Nome;
- Data di attivazione;
- Dimensione;
- Possibilità di cancellare la chiave così come di essere dismessa.

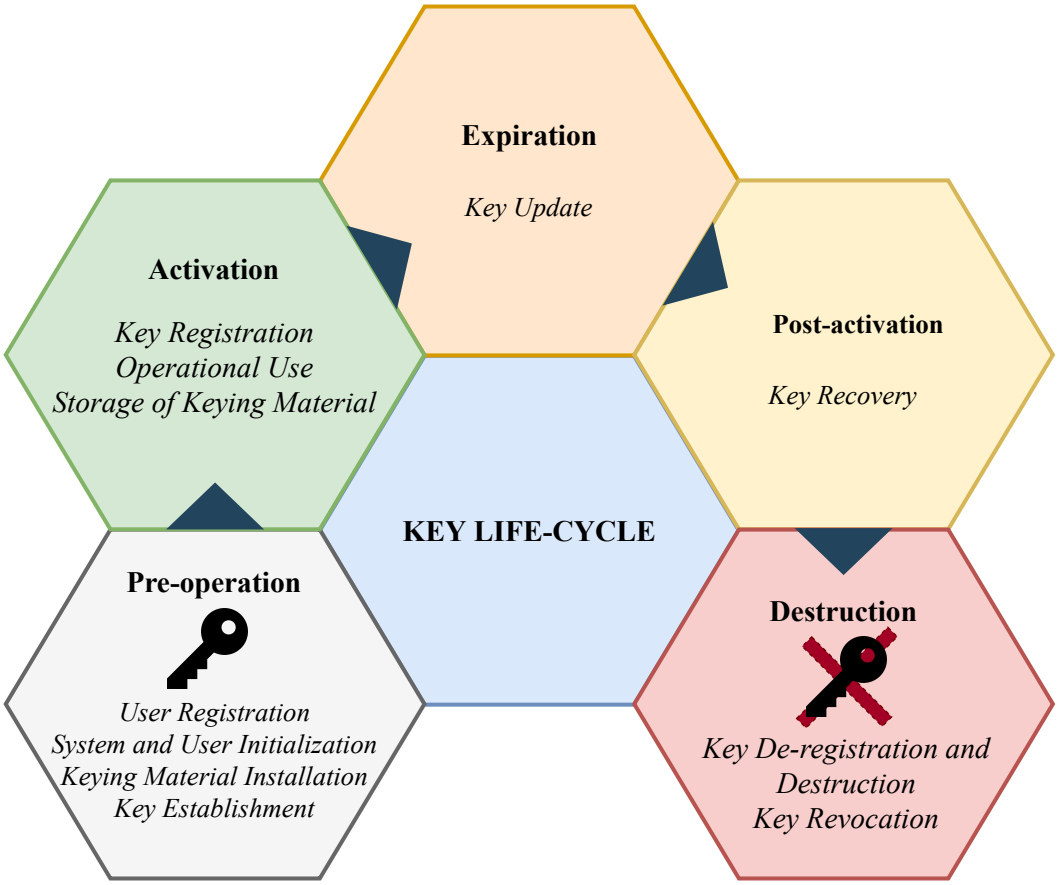


Figura 3.2: Ciclo di vita di una chiave crittografica.

La chiave può essere attivata subito dopo la sua creazione o dopo un lasso di tempo in modo automatico o manuale. Occorre stabilire se la chiave può essere cancellata o dismessa e il Key Manager deve essere in grado di cambiare i metadati associati alla chiave in qualsiasi momento. La fase di *Pre-Operation* è suddivisa in [50]:

- **User Registration.** Durante la registrazione, un'entità diventa un membro autorizzato all'interno del dominio di sicurezza. Quindi può effettuare le operazioni di creazione o scambio delle chiavi;
- **System and User Initialization.**
  - *System Initialization.* Consiste nel configurare e inizializzare il sistema in modo sicuro;
  - *User Initialization.* Una entità inizializza la propria applicazione crittografica.
- **Keying Material Installation.** Il Keyring Material [5] viene installato;
- **Key Establishment.** La chiave viene creata.

Key Activation & Key Operation

Il Key Manager deve permettere ad un sistema autorizzato (o utenti) di ottenere la chiave per cifrare/decifrare i dati. Il Key Manager dovrebbe tenere traccia delle chiavi vecchie e della chiave corrente. Le versioni precedenti possono essere ancora ottenute per decifrare i dati cifrati con la chiave vecchia e non con quella nuova.
Questa fase è suddivisa in [39]:

[5] Si intende tutto il *materiale* necessario durante la creazione della chiave.

- **Key Registration.** Il Keying Material viene legato alle informazioni o agli attributi associati ad una particolare entità;
- **Operational Use.** La chiave è disponibile per essere utilizzata. In normali circostanze una chiave rimane operativa fino alla fine del suo cryptoperiod;
- **Storage of Keying Material.** Il tipo di storage della chiave dipende dai requisiti di protezione richiesti.

  Quando richiesta per normale uso, la chiave deve risiedere nell' operational storage.

  Quando l'operation storage è corrotto o perso, la chiave deve essere recuperata da un backup storage.

  Dopo la fine del cryptoperiod la chiave deve essere posta in un archive storage.

  Le chiavi possono essere memorizzate in modo da essere subito disponibili all'applicazione (sull'hard disk del server) oppure possono essere memorizzate in forma elettronica in un dispositivo rimovibile e piazzato in un posto sicuro (questo metodo è soprattutto usato per backup o archivi). Lo storage deve garantire le seguenti proprietà di sicurezza [50]:

  – *Confidenzialità.*
    La chiave deve risiedere in un Approved Cryptographic Module (ACM). L'ACM deve essere progettato per essere conforme allo standard FIPS-140[6] e deve essere stato accreditato da un laboratorio CVMP (Cryptographic Module Validation Program) [52]. La chiave deve inoltre essere memorizzata in un ambiente sicuro e fidato e deve essere divisa in più componenti (principi di *Split Knowledge e Dual Control*[7]);
  – *Integrità.*
    L'integrità riguarda la prevenzione modifiche non autorizzate della chiave. Tutte le chiavi richiedono integrity protection. L'integrity protection è fornita da un fidato e sicuro sistema operativo o da un ACM utilizzando meccanismi crittografici come MAC o firma digitale;
  – *Association with Usage or Application.*
    La chiave deve essere associata ad una particolare applicazione e tenuta separata dalle applicazioni in modo da evitare usi incorretti e non autorizzati;
  – *Association with Other Entity.*
    Molte chiavi richiedono di essere correttamente associate tra diverse entità;
  – *Long Term Availability.*
    Le chiavi devono essere facilmente rimpiazzabili senza serie conseguenze se queste diventano non disponibili con sistemi efficienti di backup;
  – *Association with Other Information.*
    Deve essere memorizzata ciascuna associazione tra chiave e informazione protetta da quella chiave.

  Lo storage può essere dei seguenti tipi [50]:

  – **Operational Storage.**
    La chiave deve rimanere memorizzata e tenuta sicura per tutto il cryptoperiod;
  – **Backup Storage.**
    Il backup si riferisce a un tipo di storage durante l'operational use. Non tutte le chiavi richiedono backup;
  – **Key Archive Storage.**
    Si tratta di un archivio e contiene lo storico delle chiavi. Non tutte le chiavi richiedono di essere archiviate. Le chiavi archiviate possono essere statiche oppure possono essere cifrate con una nuova chiave. Le chiavi archiviate dovrebbero essere tenute separate delle chiavi attive e tenute in copie multiple. Le chiavi che non sono più richieste devono essere distrutte. Il Key Archive Storage deve garantire confidenzialità e integrità.

[6] Tale standard verrà approfondito successivamente nel capitolo.

[7] Saranno approfonditi successivamente.

Key Revocation & Key Update

Un amministratore dovrebbe usare il Key Manager per poter revocare una chiave che non è più usata per le richieste di cifratura e decifratura. Una chiave revocata può essere, se necessario, riattivata da un amministratore per decifrare ad esempio vecchi backup [39].

Back Up & Key Recovery

La sezione 8.3.1 del NIST richiede che deve essere mantenuto un archivio per memorizzare le chiavi dismesse. L'archivio deve proteggere il suo contenuto da modifiche, diffusioni, inserimenti e cancellazioni da sistemi non autorizzati. Inoltre la chiave deve essere recuperabile dopo la fine del suo *Crypto Period* ed inoltre il sistema deve essere progettato per permettere la ricostruzione delle chiavi qualora queste debbano essere riattivate per essere utilizzate nella decifrazione dei dati che sono stati cifrati precedentemente [39].

Key Deletion (Distruzione)

Se una chiave non viene più usata o è stata compromessa, un amministratore può scegliere di eliminare interamente la chiave dal Key Storage Database del Key Manager. Il Key Manager quindi rimuoverà la chiave, tutte le sue instanze e i metadati rendendo così il *Recovery* della suddetta chiave impossibile. Questa opzione può essere presa in considerazione quando dati sensibili vengono compromessi. Se la chiave viene eliminata, i dati saranno completamente sicuri ma non ripristinabili dato che è impossibile ricreare la chiave che li ha cifrati [39].

3.3 Best practice nella gestione dei ruoli

3.3.1 Separazione delle funzioni

Nella "Recommendation for Key Management-Part2" [53] il NIST definisce la **Separazione delle funzioni** come:

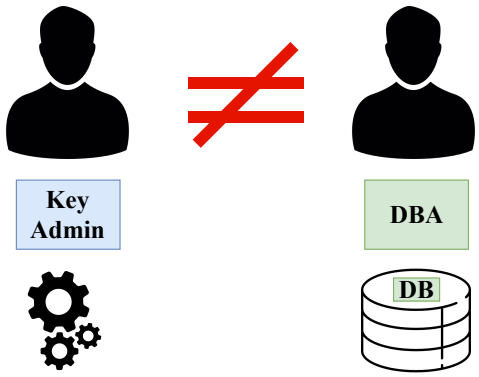
"*Un principio di sicurezza che divide le funzioni critiche tra differenti membri dell'organizzazione nel tentativo di garantire che nessun individuo abbia sufficienti informazioni o privilegi di accesso per effettuare frodi dannose.*"

La pratica della *Separazione delle funzioni* riduce il potenziale di possibili frodi o azioni illecite dividendo le responsabilità di task critici tra differenti individui di una organizzazione. Di solito i task critici per la sicurezza sono divisi in quattro categorie [39]:

- Autorizzazione;
- Memorizzazione;
- Registrazione;
- Riconciliazione.

In un sistema perfetto nessun individuo dovrebbe gestire più di un tipo di funzione.

Riguardo alla cybersecurity, l'implementazione di una *Separazione delle funzioni* è critica nell'area della **Key Management**. Per prevenire accessi non voluti a dati protetti è importante che l'individuo che gestisce le chiavi di cifratura non abbia l'accesso ai dati protetti, e viceversa.



3.3.2 Controllo Duale

Di nuovo, il NIST nella "Recommendation for Key Management-Part2" definisce il **Controllo Duale** come :

"Un processo che usa due o più entità separate (di solito persone) che operano insieme in modo da proteggere informazioni o funzioni sensibili. Nessuna singola entità deve avere la capacità di accesso o uso del materiale sensibile."

Mentre la *Separazione delle funzioni* consiste nel distribuire differenti parti di un processo a persone differenti, il *Controllo Duale* richiede che almeno due o più individui controllino un singolo processo.

Nella **Data Security** è comune trovare dei requisiti per il controllo duale delle funzioni di *Key Management* [39].

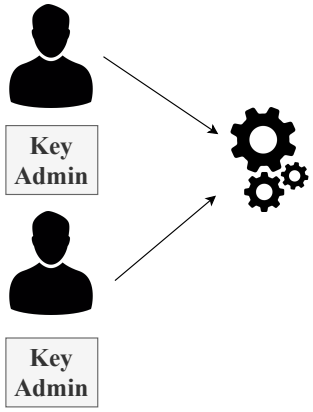
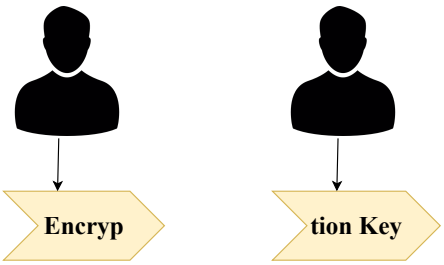


Figura 3.3: Esempio : Due o più Key Admin sono responsabili per la generazione delle chiavi di cifratura.

3.3.3 Dividere la conoscenza

Il concetto di *Dividere la conoscenza* viene applicato ad ogni accesso o gestione di materiale crittografico come chiavi di cifratura o passphrase utilizzate per creare chiavi di cifratura e richiede che nessuno conosca il valore completo della chiave. Se una passphrase viene usata per creare una chiave di cifratura nessuno dovrebbe conoscere l'intera passphrase. Piuttosto, due o più persone dovrebbero ciascuno conoscere una parte dell'intera passphrase e tutti dovrebbero essere presenti per creare o ricreare una chiave di cifratura [39].



3.4 Piattaforme

3.4.1 HSM

Hardware Security Module (HSM) è un dispositivo fisico (hardware + software/firmware) che gestisce e salvaguarda le chiavi digitali, effettua operazioni crittografiche (cifratura, decifratura, firma digitale e autenticazione) e protegge i dati riservati in transito, in uso e a riposo. Contiene uno o più *cryptoprocessor chip* e le sue funzioni principali sono [54]:

- Generazione sicura delle chiavi crittografiche *on-board*;
- Key Storage sicuro delle chiavi crittografiche *on-board*;
- **Key Management**;
- Effettuare operazioni crittografiche:
  - Crittografia simmetrica;
  - Crittografia asimmetrica;
  - Hashing;
  - Random Generator:
    - True Random Generator;
    - Pseudo Random Generator.
- Physical e Logical Security;
- Alleggerire il carico sul server applicativo effettuando operazioni di crittografia simmetrica e asimmetrica.

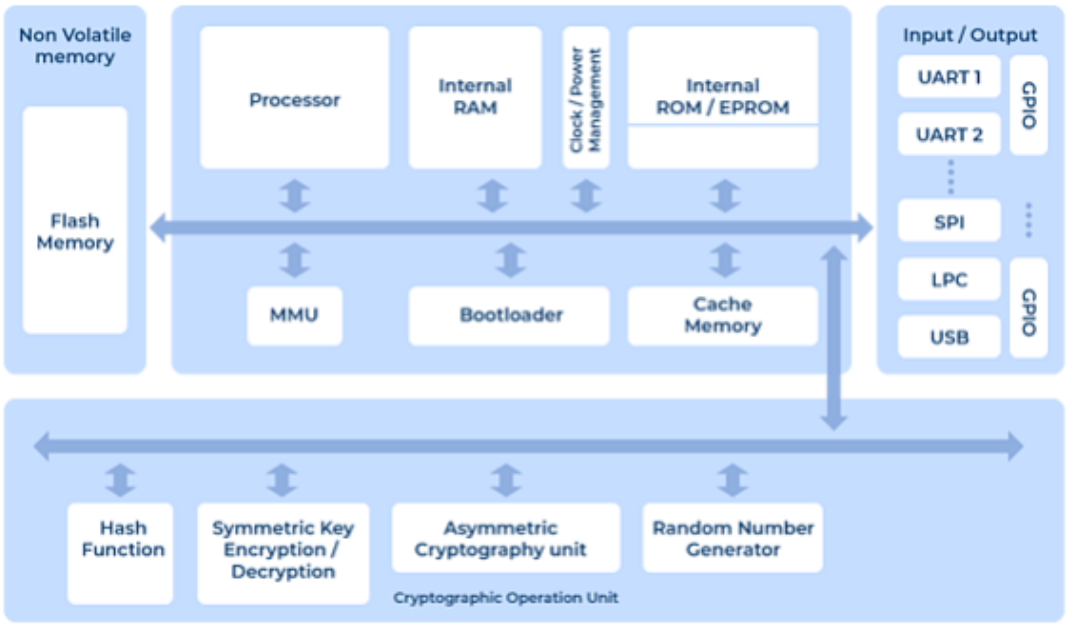


Figura 3.4: Schema di un Hardware Security Module

HSM è un dispositivo che può essere installato direttamente in un PC o server oppure può essere mobile e quindi separato dal PC o server.
Le interfacce di comunicazione sono [54]:

- Interne :
  - PCI Bridge (32 bit /64 bit);
  - PCI Express.
- Esterne :
  - Seriali: RS232;
  - Ethernet;
  - USB.

Tamper Security

Gli Hardware Security Module sono anche utilizzati per gestire e proteggere le chiavi utilizzate per la **Transparent Data Encryption**. Molti sistemi HSM sono anche degli acceleratori crittografici che permettono di aumentare le performance delle operazioni crittografiche.

Dal punto di vista della sicurezza gli Hardware Security Modules sono tipicamente certificati *Common Criteria* o *FIPS 140* e possiedono diversi livelli di protezioni contro tampering o perdite.

La *Tamper Security* consiste nel [54]:

- Individuazione di penetrazione meccanica;
- Individuazione di penetrazione chimica;
- Individuazione di manipolazione termica (freezing memory attack);
- Individuazione di manipolazione della batteria;
- Individuazione di variazione del voltaggio;

- Individuazione di movimento e luci.

Queste tecniche di protezione possono essere riassunte in [39]:

- **Tamper Evident.** Consiste nell'aggiungere rivestimenti o sigilli all'esterno del modulo;
- **Tamper Resistant.** Consiste nell'aggiungere dei circuiti per identificare e rispondere correttamente a violazioni sui dati sensibili come DEK e KEK;
- **Tamper Proof.** Consiste in una protezione completa del modulo con sistemi di antimanomissione e circuiti che cancellano i dati sensibili in presenza di violazioni.

Performance

Mediamente [54]:

- RSA 1024 bit Private Key Operation: 100- 7000 operations/seconds;
- ECC 160 bit ECDSA Signature: 250-2500 operations/second;
- 3DES : 2-8 Mbyte/second;
- AES: 6-40 Mbyte/second (256 bit key).

I dati citati si riferiscono a più di dieci anni fa e quindi non rappresentano le attuali performance, sicuramente maggiori visto la crescita tecnologica.

Tipi di Hardware Security Module

Gli HSMs possono essere classificati in due tipologie [55]:

1. **General Purpose HSM.** È un HSM che include una diversa varietà di algoritmi standard di cifratura (simmetrici, asimmetrici e funzioni di hash) con supporto a *Public-Key Cryptography Standard (PKCS) #11*, *Microsoft Cryptographic Application Programming Interface (CAPI)*, *Cryptography API Next Generation (CNG)*, *Java Cryptography Architecture (JCA)* e altri. Questi dispositivi sono tipicamente usati in ambienti PKI, canali HTTPS, DNSSEC, protezione di generici dati sensibili e crypto-wallets.
2. **Transaction and Payment HSM.** È un HSM specifico per la protezione delle transazioni di pagamento che includono l'uso di un PIN (ad esempio i POS o gli ATM), protezione di Electronic Fund Transfers (EFT) (ad esempio gli ATM) e la generazione di dati per le strisce magnetiche.

Validazione dei livelli di sicurezza di un HSM

Sono stati definiti una serie di standard internazionali per validare i livelli di sicurezza di questo tipo di dispositivo, tra questi troviamo [55, 39]:

FIPS (Federal Information Processing Standard) 140-2.

Il *Federal Information Processing Standard (FIPS)* ha identificato quattro livelli crescenti di sicurezza nel FIPS 140-2 che possono essere applicati ad un modulo fisico, ognuno corrispondente ad un livello di minaccia:

- **Livello 1:** *"Non è richiesto nessun meccanismo specifico di sicurezza a livello fisico sul modulo crittografico oltre ai requisiti base per un prodotto di qualità. Il livello 1 permette al software e al firmware di un modulo crittografico di essere eseguiti su un sistema general purpose che utilizza un sistema operativo non valutato";*

- **Livello 2:** *"Aumenta la sicurezza fisica del livello 1 aggiungendo al modulo crittografico requisiti di tamper-evidence (prove di manomissione)";*
- **Livello 3:** *"Tenta di impedire all'intruso di ottenere accesso ai parametri di sicurezza critici (CSPs) del modulo crittografico. I meccanismi di sicurezza fisici includono l'uso di involucri robusti e di circuiti di rilevamento delle manomissioni che azzerano tutti i CSPs quando il sistema si accorge di essere violato";*
- **Livello 4:** *"Fornisce il livello più alto di sicurezza definito in questo standard. A questo livello di sicurezza i meccanismi di sicurezza a livello fisico forniscono un completo involucro di protezione intorno al modulo crittografico con lo scopo di identificare e rispondere adeguatamente ad un tentativo di manomissione. La violazione dell'involucro del modulo crittografico in qualsiasi direzione ha una alta probabilità di essere rilevata con conseguente azzeramento di tutti i CSPs in chiaro".*

Common Criteria (ISO/IEC 15408).

Common Criteria for Information Technology Security Evaluation è uno standard di certificazione riconosciuto a livello internazionale per i prodotti IT e sistemi di sicurezza. I prodotti valutati da Common Criteria sono categorizzati in livelli (**Evaluation Assurance Level - EAL**) da EAL1 a EAL7 [56].

Payment Card Industry (PCI) PTS HSM Security Requirements (PCI HSM). Lo standard PCI HSM è parte dello standard PCI SSC Pin Transaction Security (PTS) e definisce i controlli di sicurezza richiesti durante la produzione, spedizione, uso e smantellamento di un HSM nelle transazione economiche.

Validazione di un HSM

Le organizzazioni hanno bisogno di garanzie nel momento in cui scelgono un prodotto per la protezione dei dati sensibili. Per risolvere questo problema il NIST ha progettato un sistema per convalidare i moduli crittografici e garantire che siano conformi allo standard FIPS 140-2. Ecco i passaggi che un Encryption Key Manager vendor deve adottare per dimostrare piena conformità allo standard [39]:

1. Prima di tutto deve contattare un laboratorio accreditato che ha superato con successo il National Voluntary Laboratory Accreditation Program (NVLAP) per condurre una adeguata fase di testing e validazione del modulo crittografico e dei suoi algoritmi crittografici rispetto agli standard stabiliti per cercare debolezze come progettazione scadente o algoritmi deboli;

2. Successivamente il laboratorio accreditato conduce il Cryptographic Algorithm Validation Program (CAVP). Questo test permette di validare l'adeguatezza del modulo a FIPS e NIST dal punto di vista degli algoritmi crittografici e delle componenti;

3. Una volta che il test è completato e il Key Manager sofddisfa tutti gli standard, il laboratorio effettua un test chiamato Cryptographic Module Validation Program (CMVP). Il laboratorio utilizza la Derived Test Requirements (DTR) [57] e Implementation Guidance (IG) [58] per testare il modulo crittografico;

4. Alla fine, una volta mostrato che il modulo soddisfa i requisiti dello standard FIPS 140-2, il laboratorio rilascia il FIPS 140-2 Validation Certificate e il Key Manager viene inserito nella FIPS 140-2 Vendor List.

Conclusioni

Bruce Schneier, nel suo libro "Applied Cryptography" afferma :

"La gestione delle chiavi crittografiche è la parte più difficile della crittografia e spesso è il tallone d'Achille dei sistemi sicuri".

Un modo sicuro e non molto complicato i gestire le chiavi è quello di usare un Hardware Security Module che permette di ottenere un Key Management sicuro, processando i dati sensibili in modo conforme agli standard internazionali e perciò rappresenta una più sicura alternativa all'uso di generiche librerie crittografiche software.

3.4.2 Soluzioni Virtuali

Instanze virtuali di un Encryption Key Manager offrono molta più flessibilità rispetto alla controparte HSM. In molti casi un Virtual Key Manager può essere scaricato in pochi minuti e distribuito all'ambiente virtuale. Un HSM, dall'altra parte, può richiedere giorni o settimane per essere ricevuto e richiede una installazione fisica. Inoltre, instanze virtuali possono essere installate su qualsiasi sistema che supporta la virtualizzazione.

Lo svantaggio, ovviamente, è che trattandosi di un componente virtuale e non di un componente fisico, il software di un Virtual Key Manager può essere compatibile con FIPS 140-2 ma non validato. Perciò se si vuole avere più sicurezza e si richiede validazione FIPS 140-2 l'HSM è la scelta migliore [39].

Un esempio celebre è HashiCorp KeyVault [59] che offre un Key Manager virtuale che può interfacciarsi con diversi tipi di storage (compresi quelli cloud) ed offre anche una soluzione di memorizzazione integrata.

3.4.3 Soluzioni Cloud dedicate

I servizi cloud, come Amazon Web Services (AWS), Microsoft Azure ed altri hanno un marketplace che offre delle soluzioni di Key Management come un servizio (KMaaS). I KMaaS di AWS o Azure riguardando architetture multi-tenant, che significa che nella stessa instanza del Key Manager possono essere presenti più chiavi di utenti diversi. Questo fa si che le organizzazioni debbano avere dei servizi dedicati per mitigare gli accessi di diversi utenti sullo stesso key store. Per risolvere questo problema, la maggiorparte dei cloud provider offrono dei servizi dedicati che hanno gli stessi effetti di HSM fisici. Nel marketplace è possibile trovare anche soluzioni di tipo *Bring your own encryption (BYOK)* che permettono ai clienti che usufruiscono del prodotto cloud di usare il proprio encryption software e gestire da per sè le chiavi di cifratura [39].

3.4.4 PKCS#12

PKCS#12 definisce il formato per memorizzare egli oggetti crittografici come file singoli. È comunemente usato per raggruppare una chiave privata con il suo certificato X.509 o per raggruppare tutti i membri di una *chain of trust* [60].

Un file PKCS#12 può essere cifrato o firmato. PKCS#12 fa parte della famiglia *Public-Key Cryptography Standards* pubblicato da RSA Laboratories.
L'estensione per i file PKCS#12 è .p12 o .pfx (PKCS#12 è il successore di Microsft PFX). Il file può essere creato, letto e gestito utilizzando il comando *pkcs12* della libreria OpenSSL.

Questa soluzione non è sofisticata come le precedenti e può essere usata in ambiente di sviluppo e testing [61].

3.5 Protocolli

3.5.1 PKCS#11

PKCS#11 è uno standard (appartenente a *Public-Key Cryptography Standards*) che definisce una API platform-indipendent per la gestione di Hardware Security Modules (HSMs) e smartcard. Questa API definisce i tipi degli oggetti crittografici (chiavi RSA, certificati X.509 etc.) e le funzioni necessarie per l'uso, creazione, modifica ed eliminazione degli oggetti [62].

3.5.2 Key Management Interoperability Protocol (KMIP)

Key Management Interoperability Protocol (KMIP) [63] è un protocollo di comunicazione estensibile che definisce i formati dei messaggi per la manipolazione delle chiavi crittografiche su un *Key Management Server*. Questo facilita la *Data Encryption* semplificando la Key Management. Le chiavi possono essere create sul server e successivamente inviate al client possibilmente cifrate da altre chiavi. Sono supportate le chiavi simmetriche e asimmetriche, inclusa l'abilità di firmare i certificati. KMIP permette inoltre ai client di chiedere al server di cifrare/decifrare dei dati senza avere accesso diretto alla chiave. Lo standard KMIP è stato rilasciato nel 2010 ed è gestito da OASIS [39].

Descrizione

Un server KMIP memorizza e controlla *Managed Objects*, come ad esempio chiavi simmetriche e asimmetriche, certificati e User Defined Objects. I client possono usare KMIP per accedere a questi oggetti sul server in modo sicuro. Ogni *Managed Object* possiede un *Value* immutabile, degli *Attributi* mutabili che possono essere usati per memorizzare dei metadati. Alcuni Attributi sono derivati direttamente dal Value come ad esempio l'algoritmo crittografico e la lunghezza della chiave. Altri Attributi sono direttamente definiti dal server o client. Inoltre ogni *Managed Object* è identificato da un unico e immutabile Object Identifier che è generato dal server ed usato per le operazioni di GET.

Oggetti

I tipi dei Managed Objects che sono gestiti da KMIP includono:

- Chiavi simmetriche usate per algoritmi come AES;
- Chiavi pubbliche e private usate per algoritmi asimmetrici come RSA o ECDH;
- Certificati e chiavi PGP;
- Dati segreti come password;
- Dati opachi per il client e Server Defined Extensions;
- Certificate Signing Request.

Operazioni

Le operazioni fornite da KMIP includono:

- **Creazione.** Crea un nuovo Managed Object e ritorna l'identificatore;
- **Crazione di una coppia di chiavi.** Crea due oggetti che rappresentano le chiavi asimmetriche (pubblica e privata);
- **Get.** Ritorna un Managed Object dato il suo identificatore univoco. Il valore ritornato può essere cifrato con un altra chiave che è sul server per sicurezza aggiuntiva;
- **Registrare.** Memorizza un valore generato esternamente.
- **Aggiungere, Ritornare, Modificare e Settare un attributo.** Queste operazioni possono essere usate per manipolare attributi mutabili di un Managed Object.
- **Localizzare.** Ricerca, identifica e ritorna una lista di oggetti.
- **Re-Key.** Crea una nuova chiave che rimpiazza una chiave esistente. Ci sono anche attributi che possono essere usati per avere una rotazione automatica delle chiavi sul server dopo un certo periodo;

- **Divisione o unione di chiavi**;
- **Cifrare, decifrare, generare un MAC, etc etc**. Sono tutte le operazioni crittografiche effettuate sul Key Management Server. La chiave inoltre può essere marcata come *Non Estraibile* nei casi in cui il valore non deve uscire dal server.
- **Esportare o importare chiavi da un altro KMIP server**;
- **Effettuare le operazioni per implementare il ciclo di vita delle chiavi definito dal NIST** (operazioni viste precedentemente).

Struttura del messaggio

KMIP è un protocollo *stateless* nel quale i messaggi sono inviati dal client al server e il client normalmente aspetta una risposta dal server. Ogni richiesta può contenere molte operazioni che abilitano il protocollo a gestire in modo efficiente un vasto numero di chiavi. Ci sono anche features avanzate per la gestione delle richieste asincrone.

KMIP specifica molti differenti tipi di codifica. La principale è la codifica *Type-Length-Value* chiamata **TTLV** (tag, type,length, value). TTLV annidate permettono la codifica di messaggi complessi e multi-operazione in un singolo messaggio binario. Ci sono inoltre codifiche basate su XML e JSON per protocolli dove la codifica binaria non è appropriata.

Tutti i messaggi devono essere trasmessi usando TLS in modo da garantire integrità, autenticazione e confidenzialità.

Gestione del sistema

KMIP fornisce dei meccanismi standard per gestire un server KMIP da parte di client amministrativi autorizzati usando dei *System Objects*. Oggetti utente possono essere creati e autorizzati ad effettuare operazioni specifiche sui Managed Objects. Sia Managed Objects che oggetti utente possono essere assegnati a gruppi e questi gruppi possono formare una gerarchia che facilita la gestione efficiente su ambienti complessi.

KMIP inoltre fornisce dei sistemi di provisioning che facilitano l'invio di credenziali agli end point usando delle One Time Password.

Possono essere forniti inoltre dei valori di default degli attributi in modo che client semplici non abbiano bisogno di specificare nulla riguardo a crittografia o altri parametri. Per esempio, un utente amministratore può specificare che tutte le chiavi con attributo "Secret Agent" siano cifrate usando chiavi AES a 192 bit in CBC mode. Un utente semplice che vuole solo creare una chiave con attributo "SecretAgent" non ha bisogno di specificare attributi riguardante la crittografia in quanto verranno usati quelli di default (definiti dall'utente amministratore).

Profili KMIP

KMIP definisce inoltre un insieme di **profili**, che sono sottoinsiemi di specifiche KMIP utilizzate per un particolare contesto. Una particolare implementazione KMIP è conforme ad un profilo quando soddisfa tutti i requisiti specificati nel documento del profilo. OASIS ha pubblicato vari profili ma lascia la possibilità alle organizzazioni di creare dei propri profili.

Relazione con PKCS#11

PKCS#11 è una API C usata per controllare un Hardware Security Module. PKCS#11 fornisce operazioni crittografiche di cifratura e decifratura e altre semplici operazioni per la Key Management. C'è una notevole sovrapposizione tra l'API PKCS#11 e il protocollo KMIP.

Questi due standard inizialmente furono sviluppati indipendentemente. PKCS#11 fu creato da RSA Security mentre KMIP da OASIS. Attualmente l'obiettivo è quello di allineare i due

standard. Per esempio, gli attributi PKCS#11 Sensitive and Extractable sono stati aggiunti nella versione 1.4 di KMIP. KMIP 2.0 fornisce un meccanismo standardizzato per il trasporto dei messaggi PKCS#11 dal client al server.

Capitolo 4

Analisi teorica delle implementazioni esistenti

In questo capitolo si approfondiranno le soluzioni esistenti fornite dai principali DBMS vendor (Oracle, Microsoft) e open-source (MySQL,MariaDB e PostgreSQL). Sono stati scelti questi sistemi in quanto rappresentano le soluzioni più diffuse e meglio documentate. Oracle e Microsoft sono i principali venditori di DBMS relazionali e Microsoft è stato il primo ad implementare la TDE nei propri sistemi, inoltre è l'unico ad implementare una propria soluzione di client-side encryption (Always Encrypted). MySQL, MariaDB e PostgreSQL sono i più celebri DBMS open-source relazionali.

L'Appendice C contiene tutti i comandi e dettagli per configurare e testare i prodotti.

4.1 Oracle DBMS: *Transparent Data Encryption*

Come descritto nel Capitolo 2, la **TDE** è una tecnica di cifratura a livello DBMS Engine che permette di cifrare i dati sensibili (che vengono memorizzati in tabelle o tablespace) in modo che solo gli utenti autorizzati possano leggerli correttamente.

Una volta che i dati vengono cifrati, questi vengono automaticamente e in modo trasparente decifrati per gli utenti o applicazioni autorizzati quando accedono ai dati. La TDE aiuta nella protezione dei dati a riposo nel caso in cui lo storage media o i dati vengano rubati.

L'utilizzo standard del DBMS Oracle si basa su meccanismi di autenticazione, autorizzazione e auditing per proteggere l'accesso ai dati nel database ma non li protegge da chi potenzialmente ha il controllo della macchina dove i dati vengono memorizzati. Per proteggere i file, Oracle DBMS fornisce l'implementazione della TDE. Per prevenire la decifratura dei dati da utenti non autorizzati, la TDE memorizza le chiavi di cifratura in moduli sicuri esterni al database chiamati *Key Store* [24].

Per poter comprendere il funzionamento della TDE viene inserita in questa trattazione una breve descrizione dell'organizzazione interna del DBMS Oracle. La versione analizzata è Oracle Database 21c.

Il DBMS Oracle viene implementato come un *Database Server*. In generale, un server gestisce in modo affidabile una grande quantità di dati in un ambiente multiutente in modo che gli utenti possano accedere contemporaneamente agli stessi dati. Un database server impedisce anche l'accesso non autorizzato e fornisce soluzioni efficienti per il failure recovery.

L'Oracle Database Server è composto da almeno un database (un insieme di file memorizzati sul disco). Un database può essere definito sia dal punto di vista fisico che dal punto di vista logico.

Le strutture fisiche che compongono il database sono:

- **File di dati**. Contiene i dati che il database deve memorizzare;
- **File di controllo**. Contiene i metadati che descrivono la struttura del database e delle tabelle;
- **Redo log file**. Contiene dei log che descrivono i cambiamenti e la storia dei dati.

Le strutture logiche permettono il controllo a diverse granularità dello spazio usato sul disco e sono:

- **Blocchi di dato**. Corrisponde a un numero di byte su disco;
- **Extent**. Corrisponde ad un insieme di blocchi di dato contigui;
- **Segmento**. Corrisponde ad un insieme di extent.
- **Tablespace**. Corrisponde ad un contenitore di segmenti composti da almeno un file di dato.

A partire dalla versione 21c Oracle Database supporta solo l'architettura multitenant. L'architettura multitenant abilita un database Oracle a funzionare come un *multitenant container database (CDB)* [64].

Un container è una collezione di schemi, oggetti e strutture correlate posizionate all'interno del CDB. All'interno di un CDB, ogni contenitore ha un ID e un nome unico.

Un CDB include zero, uno o più database creati dal client, *pluggable-database (PDB)* e contenitori di applicazioni. Un PDB è una collezione portatile di schemi e oggetti che appare ad un client Oracle come un database separato. Un contenitore di applicazioni è un componente CDB opzionale, creato dall'utente, che memorizza dati e metadati per uno o più back-end di applicazioni. Un CDB include zero o più contenitori di applicazioni.

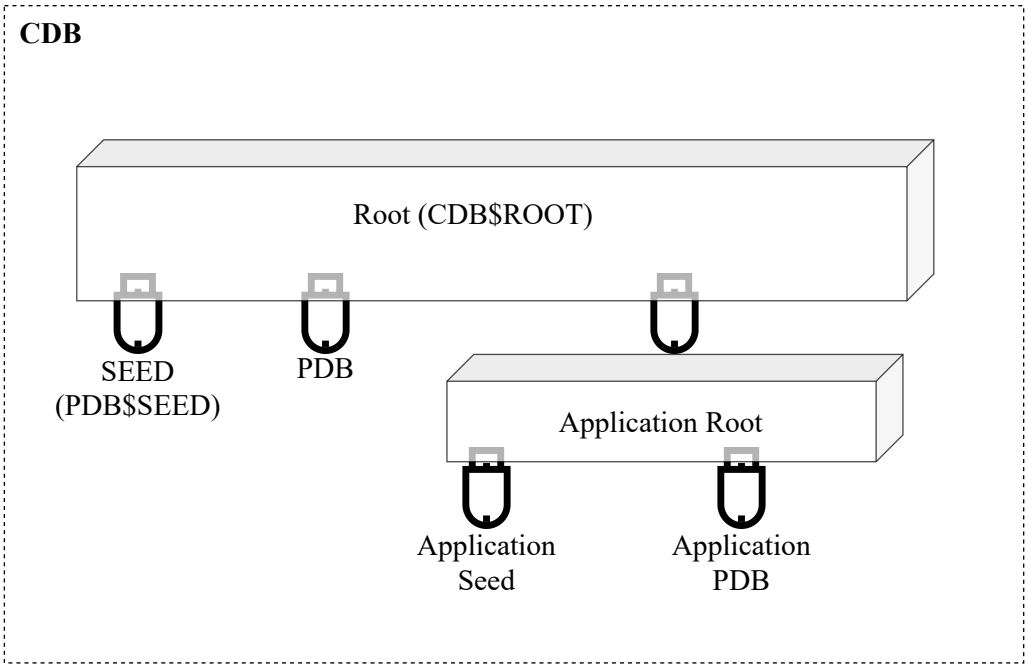


Figura 4.1: Composizione CDB.

Ogni CDB contiene i seguenti container:

- Esattamente un CDB root container (chiamato semplicemente *root*).

  Il CDB root è una collezione di schemi e oggetti comune a tutti i PDB. Il root container è identificato come CDB$ROOT;

- Esattamente un container di sistema.

  Include il root CDB e tutti i PDB;

- Zero o più container di applicazioni.

  Un contenitore di applicazioni è composto da esattamente una root di applicazione e dai PDB collegati a questa root. Mentre il contenitore di sistema contiene la radice del CDB e tutti i PDB all'interno del CDB, un contenitore di applicazioni include solo i PDB collegati alla radice dell'applicazione. Una root di applicazione appartiene alla root del CDB e a nessun altro contenitore;

- Zero o più PDB creati dagli utenti.

  Un PDB contiene i dati e il codice necessari per un insieme specifico di funzioni. Per esempio, un PDB può supportare un'applicazione specifica, come un'applicazione di risorse umane o di vendite. Nessun PDB esiste alla creazione del CDB. Si aggiungono i PDB in base ai requisiti aziendali.

  Un PDB appartiene esattamente a zero o un contenitore di applicazioni. Se un PDB appartiene ad un contenitore di applicazioni, allora è un PDB di applicazioni;

- Esattamente un PDB seed.

  Il PDB seed è un template fornito dal sistema che il CDB può usare per creare nuovi PDB. Il PDB iniziale è identificato come PDB$SEED. Non si possono aggiungere o modificare oggetti in PDB$SEED.

Oracle TDE permette di cifrare i dati sensibili a livello *Colonna* o a livello *Tablespace* [24].

Cifratura a livello colonna.

La cifratura a questo livello protegge i dati confidenziali, come i numeri delle carte di credito, che sono memorizzati nelle colonne delle tabelle.

Riguardo alle chiavi, viene utilizzata un'architettura a due livelli per cifrare e decifrare in modo trasparente le colonne sensibili della tabella. La TDE Master Encryption Key è memorizzata in un modulo di sicurezza esterno, che può essere un keystore software o un keystore hardware. Questa chiave cifra e decifra la TDE Table Key che a sua volta cifra e decifra i dati nella colonna della tabella.

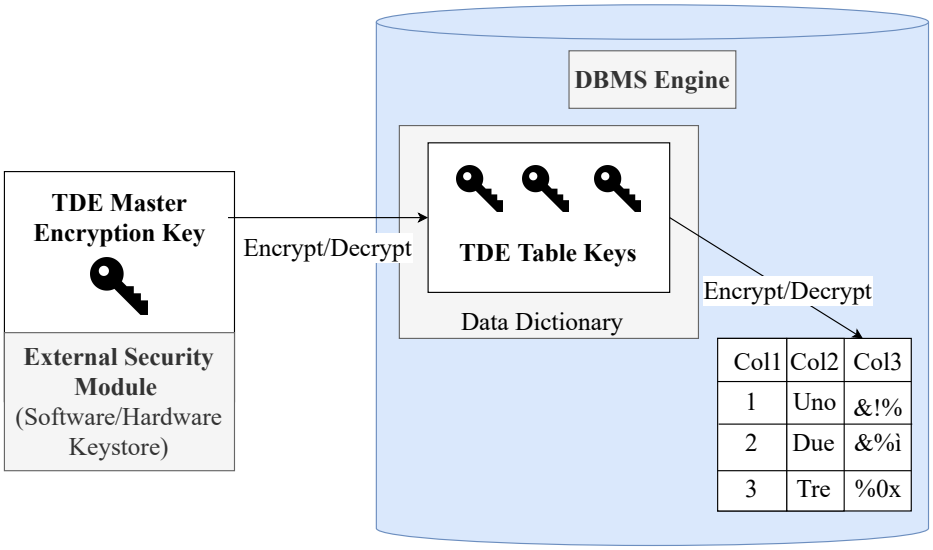


Figura 4.2: Oracle TDE: Cifratura a livello colonna (Col3 è cifrata).

Come mostrato nella Figura 4.2, la TDE Master Encryption Key è memorizzata in un modulo di sicurezza esterno che è fuori dal database e accessibile solo ad un utente a cui

sono stati concessi i privilegi appropriati. Per questo modulo di sicurezza esterno, Oracle Database usa un Oracle Software Keystore o un HSM. Memorizzare la chiave principale in questo modo impedisce il suo uso non autorizzato.

L'uso di un modulo di sicurezza esterno separa le funzioni ordinarie delle operazioni crittografiche, rendendo possibile l'assegnazione di compiti separati e distinti agli amministratori del database e agli amministratori della sicurezza. La sicurezza è migliorata perché la password del keystore può essere sconosciuta all'amministratore del database, richiedendo all'amministratore della sicurezza di fornire la password.

Quando una tabella contiene più colonne cifrate, TDE usa una singola TDE Table Key indipendentemente dal numero di colonne cifrate. Ogni TDE Table Key è cifrata individualmente con la TDE Master Key. Tutte le TDE Table Key si trovano insieme nella colonna *colklc* della tabella del dizionario dei dati *ENC$*. ENC$ è quindi un dizionario, ovvero un file di controllo, che memorizza le TDE Table Key nella colonna *colklc* che rappresentano i metadati necessari a decifrare i dati memorizzati nelle tabelle cifrate. Nessuna chiave viene memorizzata in chiaro all'interno di ENC$.

Riguardo agli indici, se si indicizza una colonna cifrata, l'indice viene creato sui valori cifrati. Quando si esegue una query per un valore nella colonna cifrata, Oracle Database cifra in modo trasparente il valore usato nella query SQL. Poi esegue una ricerca nell'indice usando il valore cifrato.

Cifratura a livello Tablespace.

La cifratura a questo livello permette di cifrare un intero tablespace.

Tutti gli oggetti creati nel tablespace vengono automaticamente cifrati, inclusi i dati Redo. Invece non vengono cifrati i dati memorizzati al di fuori del tablespace, come per esempio i dati BFILE[1] in quanto memorizzati al di fuori del database. La cifratura a livello tablespace è utile se le tabelle contengono dati sensibili in più colonne, o se si vuole proteggere l'intera tabella e non solo le singole colonne.

Inoltre, in questo caso viene sfruttata la crittografia di massa e il caching per fornire prestazioni migliorate. Per crittografia di massa si intende la *Bulk Encryption* che fa uso di algoritmi (bulk cipher) che sono usati per cifrare e decifrare un elevato numero di dati aumentando le performance rispetto a cifrare/decifrare singoli dati alla volta. Infatti nel caso della cifratura a livello tablespace tutti i dati vengono cifrati con la stessa chiave. Quindi una porzione del tablespace o l'intero tablespace può essere estratto dal database fisico, memorizzato nella cache del DBMS e poi decifrato usando un bulk cipher. Ovviamente si hanno vantaggi in termini di performance.

Tutti i dati in un tablespace cifrato sono memorizzati in formato cifrato sul disco. I dati sono decifrati in modo trasparente per un utente autorizzato che ha i privilegi necessari per visualizzare o modificare i dati. Un utente di database o un'applicazione non ha bisogno di sapere se i dati di una particolare tabella sono cifrati sul disco. Nel caso in cui i file di dati su un disco o un supporto di backup vengano rubati, i dati non vengono compromessi.

Riguardo alle chiavi, anche la crittografia a questo livello utilizza un'architettura a due livelli. Il Key Management è uguale alla cifratura a livello colonna.

La Figura 4.3 mostra una panoramica del processo di crittografia a livello tablespace. [2]

Oracle, nella documentazione, specifica anche che la cifratura a livello tablespace permette gli *index range scans* sui dati nei tablespace cifrati. Questo non è possibile con la cifratura a livello colonna. Il *Query Optimizer* sceglie il metodo d'accesso di tipo Index Range Scans per le query selettive. Ovvero le query che hanno condizioni di tipo <= e >= su una colonna indicizzata [65]. Un metodo di accesso è un modulo software che fornisce primitive per la

[1] L'Oracle BFILE è un tipo di dati LOB (*Large Object*) che contiene un riferimento a dati binari con una dimensione massima di 4 gigabyte. Un BFILE è diverso dagli altri tipi di dati LOB in quanto i dati vengono archiviati in un file fisico nel sistema operativo esterno al database invece che nel database stesso.

[2] Nota: i dati cifrati sono protetti durante operazioni come JOIN e SORT. Questo significa che i dati sono al sicuro quando vengono spostati in tablespace temporanei. Sono protetti anche i dati nei registri di undo e redo.

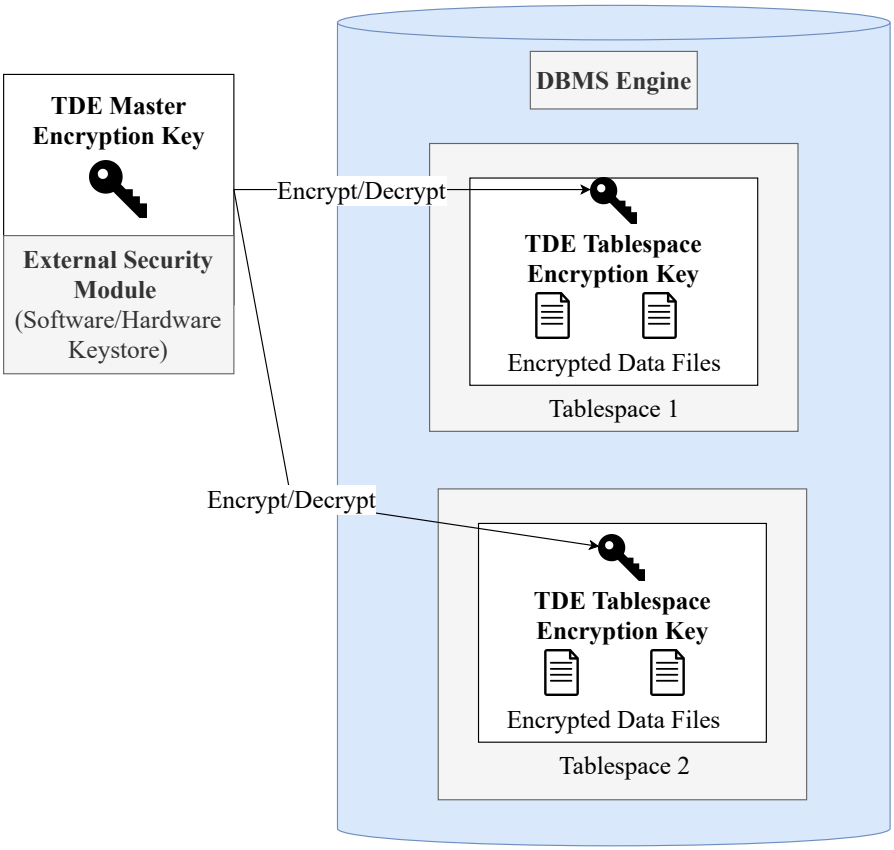


Figura 4.3: Cifratura a livello Tablespace.

lettura e scrittura dei dati. Seleziona i blocchi del file appropriati che devono essere caricati in memoria richiedendoli al *Buffer Manager* [66].

Gestione della TDE Master Encryption Key

Oracle Database fornisce un framework per la gestione delle chiavi utilizzate dalla Transparent Data Encryption. Ciò include il Keystore per memorizzare in modo sicuro la TDE Master Encryption Key ed inoltre permette un uso efficiente e sicuro delle operazioni svolte dal keystore [24].

L'Oracle Keystore memorizza inoltre tutte le TDE Master Encryption Key dismesse permettendo la decifratura anche di dati cifrati con vecchie chiavi. Inoltre permette la conformità alle best practice discusse nel Capitolo 3:

- Sepazione dei ruoli tra amminsitratore del database e amministratore di sicurezza. È possibile fornire i privilegi di ADMINISTER KEY MANAGEMENT o SYSKM agli utenti responsabili della gestione del keystore;
- Facilita le operazioni di Key Rotation e reset delle chiavi;
- Facilita e aiuta tutte le operazioni di backup e migrazione delle chiavi.

Oracle Database supporta implementazioni software dei keystore, Oracle Key Vault e PKCS #12. Schematicamente è possibile riassumere i tipi di keystore con la Figura 4.4.

Per quanto riguarda le implementazioni software, Oracle supporta:

Auto-login software keystore: si tratta di una implementazione software (file PKCS#12) protetta da una password generata dal sistema e non necessita di essere aperto esplicitamente

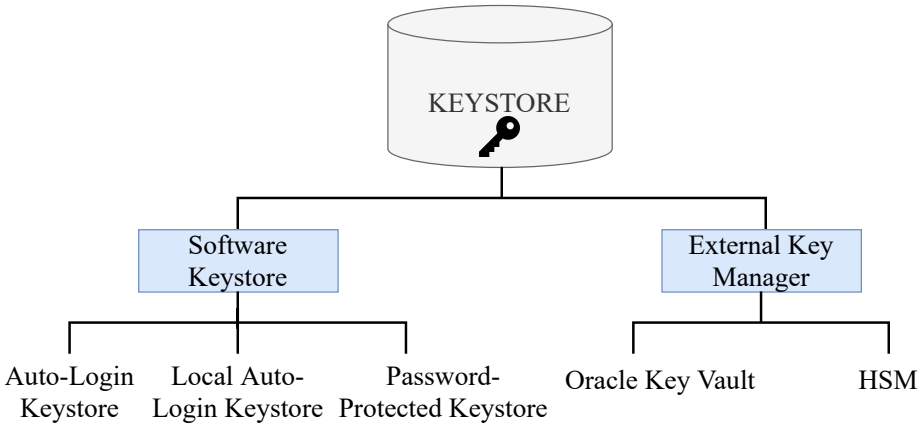


Figura 4.4: Keystore supportati da Oracle DBMS

da un amministratore di sicurezza. Infatti questo tipo di keystore sono automaticamente aperti quando vi si accede. Può essere usato, ad esempio, se il sistema non richiede requisiti extra sulla sicurezza del keystore. Di fatto non è la soluzione più sicura;

Local auto-login software keystore: si tratta di una implementazione auto-login che viene utilizzata in locale sul dispositivo che ha creato il keystore. Questi tipi di keystore non possono essere aperti da altri dispositivi oltre a quello che lo ha creato. È utile negli scenari dove sono richieste maggiori restrizioni di accesso;

Password-protected software keystore: si tratta di una implementazione software protetta da una password creata dall'utente e non automaticamente dal sistema. Per accedere al contenuto occorre conoscere la password.

Per quanto riguarda i dispositivi esterni, Oracle supporta:

Oracle Key Vault (OKV): si tratta di una implementazione software che fornisce una disponibilità continua e gestione scalabile delle chiavi attraverso il clustering con un massimo di 16 nodi Oracle Key Vault, potenzialmente distribuiti in data center geograficamente distribuiti.

Hardware Security Module (HSM): come detto nella Sezione 3.4.1, si tratta di dispositivi fisici di terze parti che forniscono un'archiviazione sicura per le chiavi crittografiche in keystore esterni. Gli HSM forniscono anche uno spazio computazionale sicuro (memoria) per eseguire operazioni di cifratura e decifratura. Se si utilizza un HSM, le chiavi di crittografia dei dati devono essere decifrate dalla TDE Master Encryption Key all'interno del perimetro FIPS dell'HSM. Questo significa che la TDE Master Encryption Key non lascia mai il confine FIPS interno dell'HSM, il che spiega l'alto carico (di rete) quando un database con un alto carico transazionale continua a inviare chiavi di crittografia dei dati all'HSM.

Nell'ambiente multitenant, è possibile configurare il keystore sia per l'intero CDB che per i singoli PDB.

Oracle Database supporta le due seguenti modalità per la gestione dei keystore multitenant:

- **United Mode.** Permette di configurare un keystore per il root CDB e qualsiasi PDB associato. Questa modalità gestisce la TDE come nelle precedenti versioni (quando non era implementato il concetto di CDB);
- **Isolated Mode.** Permette di creare e gestire il keystore in un particolare PDB. Differenti PDB isolati possono avere diversi tipi di keystore.

Si possono utilizzare queste due modalità per configurare i software keystore o keystore esterni.

Algoritmi crittografici supportati

Riguardo agli algoritmi supportati, Oracle Database supporta AES con chiavi crittografiche lunghe 128,192 o 256 bit. AES128 viene usato di default nella cifratura a livello tablespace mentre AES192 è l'algoritmo di cifratura di default per la cifratura a livello colonna. È possibile cambiare l'algoritmo di cifratura utilizzando il comando TABLE_SPACE_ENCRYPTION_DEFAULT_ALGORITHM.

Per la cifratura a livello colonna, il sale è aggiunto di default al testo in chiaro prima del processo di cifratura. Non è possibile aggiungere sale alle colonne indicizzate che si vogliono cifrare. Per le colonne indicizzate occorre inserire il parametro NO SALT nella clausola SQL ENCRYPT.

Infine è possibile cambiare algoritmo di cifratura o la chiave di cifratura per una colonna cifrata utilizzando la clausola SQL ENCRYPT.

Per quanto riguarda gli algoritmi per l'integrità, l'algoritmo SHA-1 è deprecato.

La tabella 4.1 mostra gli algoritmi supportati da Oracle DBMS.

Algoritmo	**Dimensione della chiave**	**Parameter Name**
AES	• 128 bit (default per la tablespace encryption) • 192 bit (default per la column encryption) • 256 bit	• AES128 • AES192 • AES256
ARIA	• 128 bit • 192 bit • 256 bit	• ARIA128 • ARIA192 • ARIA256
GOST	256 bit	GOST256
SEED	128 bit	SEED128
3DES	168 bit	3DES168

Tabella 4.1: Algoritmi simmetrici supportati da Oracle TDE.

Nel capitolo 5 vengono elencati i risultati sperimentali confrontati con i dati degli altri vendor.

4.2 Oracle DBMS: *DBMS_CRYPTO*

Riguardo a DBMS_CRYPTO, la versione analizzata è quella fornita da Oracle Database 21c.

DBMS_CRYPTO fornisce un'interfaccia per cifrare e decifrare i dati. Può essere usato nei programmi PL/SQL e per questo motivo è una soluzione a livello *SQL Interface*. Fornisce supporto per molti algoritmi crittografici e di hash. Inoltre è possibile scegliere molte opzioni di padding, come PKCS#15, e modalità di cifratura, come CBC [29].

DBMS_CRYPTO fornisce inoltre il supporto delle operazione crittografiche per i più comuni tipi di dato, inclusi *RAW* (tipo di dato binario) e *LOB* (tipo di dato usato per memorizzare un file di grandi dimensioni), che possono essere usati per memorizzare immagini o suoni. In particolare, per i LOB, DBMS_CRYPTO supporta i *BLOB* (LOB memorizzati esternamente al database) e *CLOB* (LOB memorizzati internamente al database) [67]. La Tabella 4.2 mostra le funzionalità offerte.

Oracle Database installa il pacchetto DBMS_CRYPTO nello schema SYS[3]. L'amministratore del DB, il DBA, può fornire i privilegi di utilizzo agli utenti in base alla necessità.

Un utente, che ha ottenuto i privilegi, può utilizzare le funzioni offerte da DBMS_CRYPTO direttamente nella query SQL. Ad esempio:

```
INSERT INTO UTENTI(ID,NOME,NUMEROCARTACREDITO)
VALUES(1,Lorenzo,
    dbms_crypto.encrypt(
    utl_i18n.string_to_raw(598371037,'AL32UTF8'),
     DBMS_CRYPTO.ENCRYPT_AES256 + DBMS_CRYPTO.CHAIN_CBC +
         DBMS_CRYPTO.PAD_PKCS5,
    utl_i18n.string_to_raw(password,'AL32UTF8')))
```

La query di esempio mostra un possibile utilizzo per l'operazione di INSERT. Utilizzando la funzione DBMS_CRYPTO.ENCRYPT è possibile cifrare un dato. La funzione accetta tre parametri: la stringa da cifrare, la modalità di cifratura e la password. La funzione UTL_I18N.STRING_TO_RAW esegue la conversione da stringa a RAW.

Come si può notare dalla query di esempio il Key Management è esplicito, ovvero è cura dell'utente gestire correttamente le chiavi. Questa soluzione infatti non supporta il Key Management implicito (come la TDE).

FEATURE	**DBMS_CRYPTO**
Algoritmi crittografici	DES, 3DES, AES, 3DES_2KEY
Padding	PKCS#5, zeroes
Block cipher chaining modes	CBC, CFB, ECB, OFB
Algoritmi crittografici di hash	MD5, MD4, SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512)
Algoritmi Keyed Hash (MAC)	HMAC_MD5, HMAC_SHA1, HMAC_SHA256, HMAC_SHA384, HMAC_SHA512
Cryptographic pseudo-random number generator	RAW, NUMBER, BINARY_INTEGER
Tipi supportati	RAW, CLOB, BLOB

Tabella 4.2: DBMS_CRYPTO: Funzionalità

4.3 Microsoft SQL Server: *Transparent Data Encryption*

Riguardo alla implementazione Microsoft della Transparent Data Encryption [25] (versione SQL Server 2019), la cifratura dei file del database è effettuata dal DBMS Engine a livello pagina. Le pagine nel database cifrato vengono cifrate prima di essere scritte sul disco e decifrate quando sono lette dalla memoria. Inoltre, come per l'implementazione Oracle, la TDE non aumenta la dimensione del database cifrato.

La figura 4.5 mostra il workflow della Microsoft SQL Server Transparent Data Encryption. La versione analizzata è SQL Server 2019.

La Service Master Key è la radice della gerarchia di chiavi di SQL Server. L'SMK viene generato automaticamente la prima volta che l'istanza di SQL Server viene avviata e viene utilizzata per crittografare la password del server collegata, le credenziali e la Database Master Key di ogni

[3]Tutte le tabelle di base e le viste per il dizionario dei dati del database sono memorizzate nello schema SYS. Per mantenere l'integrità del dizionario dei dati, le tabelle nello schema SYS sono manipolate solo dal database.

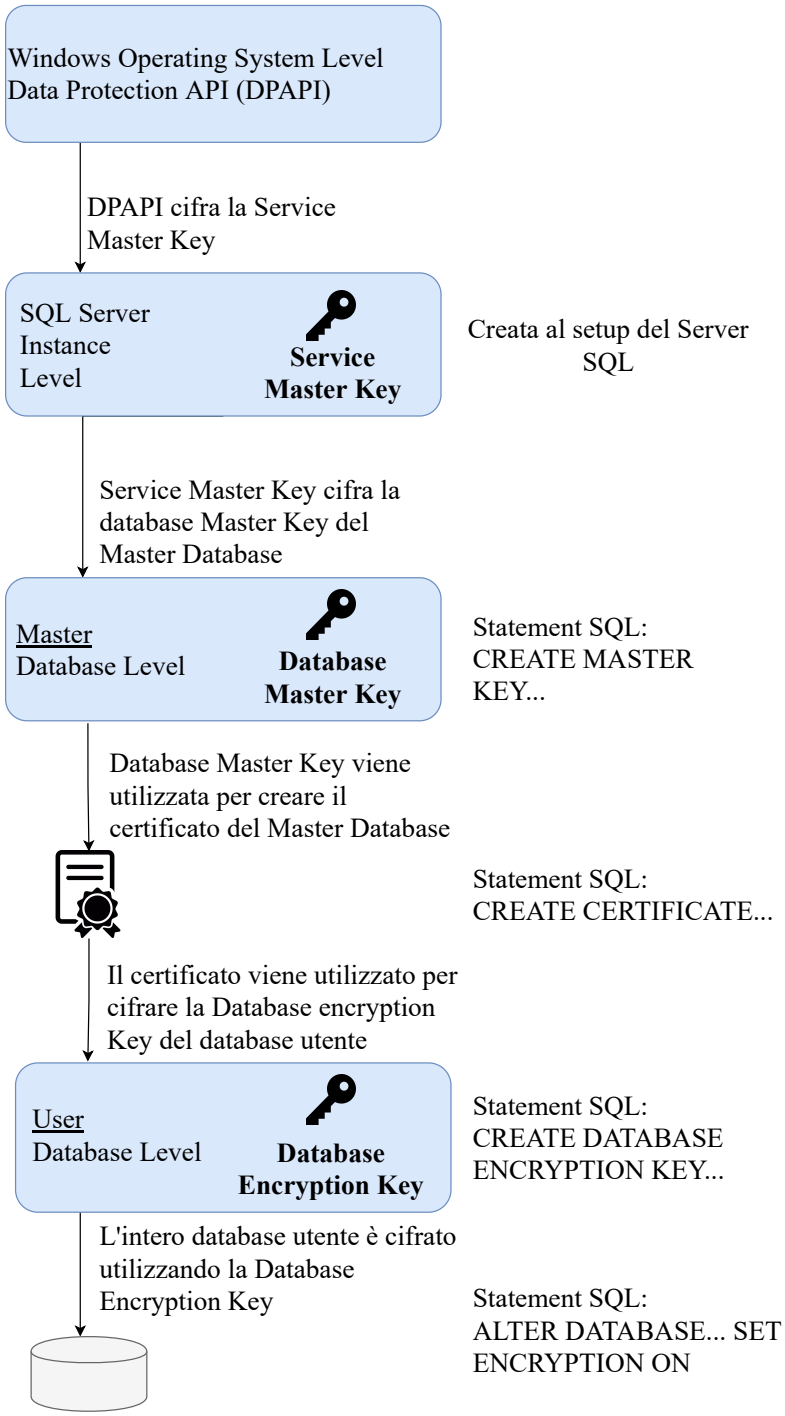


Figura 4.5: Microsoft TDE - Workflow.

database. La SMK è cifrata usando la chiave della macchina locale utilizzando l'API di protezione dei dati di Windows (DPAPI). La DPAPI usa una chiave che deriva dalle credenziali di Windows dell'account di SQL Server e dalle credenziali del computer [68]. La SMK può essere decifrata solo dall'account con cui è stata creata o da un utente che ha accesso alle credenziali della macchina. SQL Server utilizza l'algoritmo di crittografia AES per proteggere la Service Master Key (SMK) e la Database Master Key (DMK).

La Database Master Key è una chiave simmetrica che viene usata per proteggere le chiavi private dei certificati e le chiavi asimmetriche che sono presenti nel database [69].

Per utilizzare la TDE occorre effettuare i seguenti step:

1. Creare la Master Key;
2. Creare o ottenere il certificato protetto dalla Master Key;
3. Creare la Database Encryption Key e proteggerla usando il certificato;
4. Configurare il database per usare la TDE.

A differenza dell'implementazione Oracle, Microsoft SQL Server TDE non permette la cifratura a livello colonna. Si può ottenere questa tipologia di cifratura utilizzando la SQL Interface e le funzioni Transact-SQL messe a disposizione da SQL Server o utilizzando Always Encrypted.

SQL Server fornisce le funzionalità di crittografia dei dati anche insieme all' Extensible Key Management (EKM) [70]. Le chiavi di crittografia per i dati vengono generate all'interno di "contenitori" esterni al DBMS. Questo approccio consente la gestione delle chiavi, includendo una gerarchia delle chiavi di crittografia e un backup delle chiavi, in un modulo esterno a SQL Server, come un HSM.

Le implementazioni HSM variano da fornitore a fornitore e il loro uso con SQL Server richiede un'interfaccia comune. Anche se questa interfaccia è fornita da MSCAPI (le API crittografiche fornite da Microsoft) [71], questa supporta solo un sottoinsieme delle funzionalità HSM. Sono presenti inoltre altre limitazioni, ad esempio l'impossibilità di mantenere a livello nativo le chiavi simmetriche e la mancanza di supporto orientato alla sessione. Per sessione si intende un insieme di connessioni multiple con l'HSM. MSCAPI quindi fornisce supporto solo per le singolo connessioni con l'HSM.

SQL Server Extensible Key Management consente ai fornitori di EKM/HSM di terze parti di registrare i propri moduli in SQL Server. Quando registrati, gli utenti di SQL Server possono usare le chiavi di crittografia archiviate nei moduli EKM. Ciò consente a SQL Server di accedere alle funzionalità di crittografia avanzate supportate da tali moduli, quali le operazioni crittografiche a le funzioni di gestione delle chiavi quali il Crypto Period e la Key Rotation.

Gli unici algoritmi crittografici simmetrici supportati da SQL Server 2019 sono AES128, AES192 e AES256 [72].

4.4 Microsoft SQL Server: *Transact-SQL functions*

In questa sezione si analizzano le funzioni Transact-SQL crittografiche relative alla versione SQL Server 2019.

Le funzioni Transact-SQL crittografiche costituiscono la soluzione Microsoft per la cifratura a livello SQL Interface [30].

Fornisce il supporto per la cifratura simmetrica, asimmetrica, generazione e verifica di firme digitali e hash.

Transact-SQL funzioni crittografiche		
	Nome	Descrizione
Crittografia simmetrica	ENCRYPTBYKEY	Cifra i dati utilizzando una chiave simmetrica.
	DECRYPTBYKEY	Decifra i dati utilizzando una chiave simmetrica.
	ENCRYPTBYPASSPHRASE	Cifra i dati utilizzando una passphrase e l'algoritmo 3DES con una chiave di 128 bit.
	DECRYPTBYPASSPHRASE	Decifra i dati utilizzando una passphrase.
	KEY_ID	Ritorna l'ID della chiave simmetrica.

	KEY_GUID	Ritorna il GUID della chiave simmetrica.
	DECRYPTBYKEYAUTOASYMKEY	Decifra i dati cifrati. Per fare ciò, prima decifra la chiave simmetrica con una separata chiave asimmetrica e dopo decifra i dati cifrati con la chiave simmetrica estratta nel primo step.
	KEY_NAME	Ritorna il nome della chiave simmetrica.
	SYMKEYPROPERTY	Ritorna l'algoritmo associato alla chiave simmetrica.
Crittografia asimmetrica	ENCRYPTBYASYMKEY	Cifra i dati utilizzando una chiave asimmetrica.
	DECRYPTBYASYMKEY	Decifra i dati utilizzando una chiave asimmetrica.
	ENCRYPTBYCERT	Cifra i dati utilizzando la chiave pubblica di un certificato.
	DECRYPTBYCERT	Decifra i dati cifrati utilizzando la chiave privata associata ad un certificato.
	ASYMKEYPROPERTY	Ritorna le proprietà di una chiave asimmetrica.
	ASYMKEY_ID	Ritorna l'ID di una chiave asimmetrica.
Firma digitale	SIGNBYASYMKEY	Firma del testo in chiaro con una chiave asimmetrica.
	VERIFYSIGNEDBYASMKEY	Verifica la correttezza di una firma digitale.
	SIGNBYCERT	Firma del testo con un certificato.
	VERIGYSIGNEDBYCERT	Verifica la correttezza di una firma digitale calcolata con un certificato.
	IS_OBJECTSIGNED	Indica se un oggetto è firmato da una specifica chiave asimmetrica.
Hash	HASHBYTES	Ritorna l'hash di un input.

Per quanto riguarda la funzione SIGNBYCERT occorre fare una precisazione. Infatti permette di firmare dei dati utilizzando una chiave privata associata ad un certificato. La differenza con SIGNBYASYMKEY sta nel fatto che quest'ultima utilizza una generica chiave privata per firmare i dati. Riguardo alla creazione di un certificato in SQL Server 2019 occorre utilizzare la funzione CREATE CERTIFICATE [73]. Questa funzione permette di creare un certificato compatibile con SQL Server[4] in due modalità diverse. La prima modalità permette di creare un certificato a partire da un certificato, in formato DER[5] e da un file che contiene la chiave privata, già esistenti. Il secondo modo permette di creare un certificato nuovo generando una nuova coppia di chiavi (chiave pubblica e privata) e specificando opzionalmente la password per cifrare la chiave privata. Se questa opzione non è definita, in automatico viene usata la Database Master Key per cifrare la chiave privata. In questa modalità la chiave privata cifrata viene inserita all'interno di un file protetto dalla password o dalla DMK. È possibile estrarre la chiave privata associata ad un

[4]I certificati SQL Server seguono lo standard X.509 [74].

[5]Si tratta del formato per la rappresentazione binaria di un certificato.

certificato utilizzando la funzione CERTPRIVATEKEY [75] mentre CERTENCODED [76] permette di estrarre la parte pubblica.

4.5 Microsoft SQL Server: *Always Encrypted*

In questa sezione si analizza la soluzione Always Encrypted relativa alla versione SQL Server 2019.

Always Encrypted è una funzionalità progettata per proteggere i dati sensibili, come i numeri di carta di credito, memorizzati in Azure SQL Database o nei database SQL Server [36]. Always Encrypted rappresenta la soluzione client-side di Microsoft e permette ai client di crittografare i dati sensibili all'interno delle applicazioni client e non rivelare mai le chiavi di crittografia al Database Engine (SQL Database o SQL Server). Di conseguenza, Always Encrypted fornisce una separazione tra coloro che possiedono i dati e possono vederli, e coloro che gestiscono i dati, assicurandosi che gli amministratori dei database, gli operatori dei database cloud o altri utenti non autorizzati con alti privilegi, non possano accedere ai dati crittografati. Always Encrypted permette ai client di archiviare con fiducia i dati sensibili al di fuori del loro controllo diretto. Questo permette alle organizzazioni di archiviare i loro dati in Azure e consentire la delega dell'amministrazione del database a terzi o di ridurre i requisiti di autorizzazione di sicurezza per il proprio personale DBA.

La figura 4.6 mostra il funzionamento di Always Encrypted.

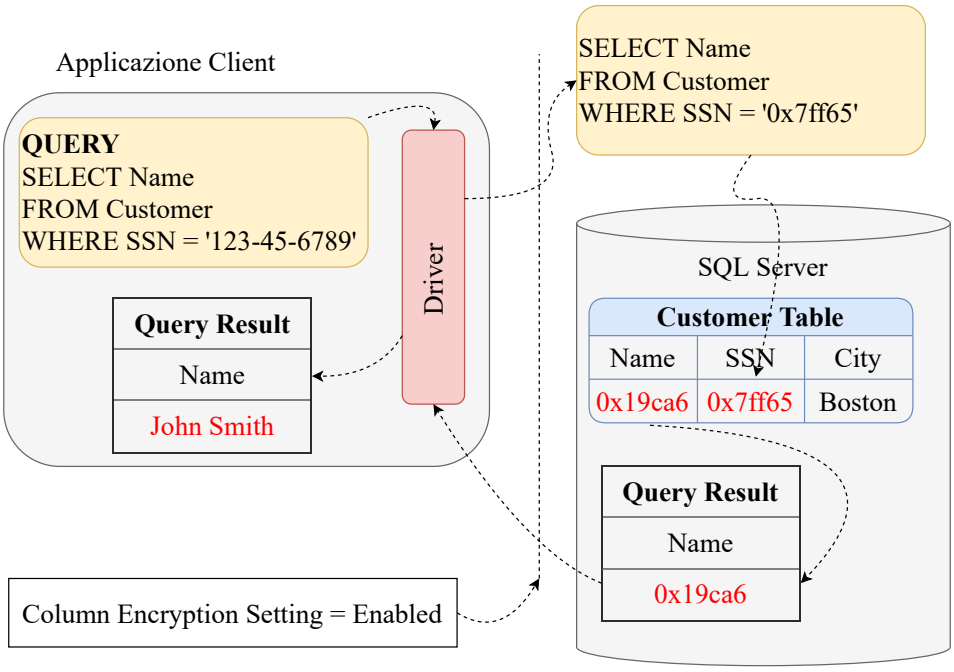


Figura 4.6: Microsoft SQL Server Always Encrypted Workflow.

Always Encrypted rende il processo crittografico trasparente alle applicazioni. Un driver Always Encrypted installato sul computer client ottiene questo risultato cifrando e decifrando automaticamente i dati sensibili nell'applicazione client. Il driver cifra i dati delle colonne sensibili prima di passare i dati al motore di database, e riscrive automaticamente le query in modo che la semantica dell'applicazione sia preservata. Allo stesso modo, il driver decifra in modo trasparente i dati, memorizzati nelle colonne del database cifrato, contenuti nei risultati delle query.

Si può configurare Always Encrypted per cifrare le singole colonne del database che contengono i dati sensibili. Quando si imposta la crittografia per una colonna, si specificano le informazioni sull'algoritmo di crittografia e le chiavi crittografiche utilizzate per proteggere i dati nella colonna. Always Encrypted usa due tipi di chiavi: le chiavi di crittografia della colonna e le chiavi master della colonna. Una chiave di crittografia di colonna è usata per crittografare i dati in una colonna

crittografata. Una chiave master di colonna è una chiave di protezione che cifra una o più chiavi di crittografia di colonna.

Il DBMS Engine memorizza la configurazione di crittografia per ogni colonna nei metadati del database. Si noti, tuttavia, che il motore di database non memorizza o utilizza mai le chiavi di entrambi i tipi in chiaro. Memorizza solo i valori crittografati delle chiavi di crittografia della colonna e le informazioni sulla posizione delle chiavi master della colonna, che sono memorizzate in archivi di chiavi esterne affidabili, come Azure Key Vault o un modulo di sicurezza hardware (HSM).

Per accedere ai dati memorizzati in una colonna crittografata in chiaro, un'applicazione deve utilizzare un driver client Always Encrypted. Quando un'applicazione emette una query parametrica, il driver collabora in modo trasparente con il Database Engine per determinare quali parametri mirano a colonne crittografate e, quindi, devono essere crittografate. Per ogni parametro che deve essere crittografato, il driver ottiene le informazioni sull'algoritmo di crittografia e il valore crittografato della chiave di crittografia della colonna, così come la posizione della corrispondente chiave master della colonna.

In seguito, il driver contatta l'archivio delle chiavi, che contiene la chiave principale della colonna, per decifrare il valore della chiave di crittografia della colonna e poi usa la chiave di crittografia della colonna in chiaro per crittografare il parametro. La chiave di crittografia di colonna in chiaro risultante viene memorizzata nella cache per ridurre il numero di viaggi di andata e ritorno verso l'archivio delle chiavi negli usi successivi della stessa chiave di crittografia di colonna. Il driver sostituisce i valori in chiaro dei parametri che mirano alle colonne cifrate con i loro valori cifrati, e invia la query al server per l'elaborazione.

Il server calcola l'insieme dei risultati, e per qualsiasi colonna crittografata inclusa nell'insieme dei risultati, il driver allega i metadati di crittografia per la colonna, incluse le informazioni sull'algoritmo di crittografia e le chiavi corrispondenti. Il driver prima cerca di trovare la chiave di crittografia della colonna in chiaro nella cache locale, e cerca la chiave principale della colonna solo se non riesce a trovare la chiave nella cache. Successivamente, il driver decifra i risultati e restituisce i valori in chiaro all'applicazione.

Un driver client interagisce con un deposito di chiavi, contenente una chiave principale di colonna, usando un fornitore di deposito di chiavi principale di colonna, che è un componente software lato client che incapsula un deposito di chiavi contenente la chiave principale di colonna. I provider per i tipi comuni di archivi di chiavi sono disponibili nelle librerie di driver lato client di Microsoft o come download autonomo. È anche possibile implementare il proprio provider. Le capacità Always Encrypted, inclusi i fornitori di archivi di chiavi master di colonna incorporati, variano a seconda della libreria del driver e della sua versione.

Always Encrypted supporta due tipi di crittografia: Randomized Encryption e Deterministic Encryption.

La *Deterministic Encryption* genera sempre lo stesso valore crittografato per qualsiasi valore di testo in chiaro. L'utilizzo della crittografia deterministica permette di effettuare ricerche, join di uguaglianza, raggruppamenti e indicizzazioni su colonne crittografate. Tuttavia, può anche permettere agli utenti non autorizzati di indovinare le informazioni sui valori criptati esaminando i modelli nella colonna criptata, specialmente se c'è un piccolo insieme di possibili valori criptati, come Vero/Falso, o Nord/Sud/Est/Ovest.

La *Randomized Encryption* usa un metodo che cifra i dati in modo meno prevedibile. Si tratta di una crittografia più sicura, ma impedisce la ricerca, il raggruppamento, l'indicizzazione e l'unione sulle colonne crittografate.

Perciò è preferibile usare la Deterministic Encryption per le colonne che saranno usate come parametri di ricerca o di raggruppamento. Per esempio, un numero identificativo governativo. Invece è preferibile utilizzare la Randomized Encryption per dati come i commenti di indagini riservate, che non sono raggruppati con altri record e non sono usati per unire le tabelle.

Always Encrypted utilizza l'algoritmo AEAD_AES_256_CBC_HMAC_SHA_256 per cifrare i dati nel database.

4.6 MariaDB: *Data-at-rest Encryption*

MariaDB è un DBMS open-source che fornisce la funzionalità Data-at-rest Encryption (anche chiamata Transparent Data Encryption) dalla versione 10.1 per evitare la diffusione di dati sensibili in caso di accesso fisico non autorizzato al database [27].

Riguardo la versione 10.5.11, MariaDB supporta la cifratura su tutti i motori di storage (DBMS Engine) compatibili con esso come InnoDB e Aria. Si può scegliere di cifrare un intero tablespace, una tabella singola o tutto il database. Opzionalmente si possono cifrare i file di log (è comunque un'operazione raccomandata). MariaDB dichiara che la cifratura aggiunga un overhead del 3-5% sulle prestazioni. Inoltre se si utilizza InnoDB come motore è possibile utilizzare la compressione della pagina insieme alla TDE in modo da diminuire l'impatto sullo storage. In questo caso i dati (o meglio le pagine) vengono prima compressi con un algoritmo di compressione, come ZLIB o ZL4 [77], e poi cifrati, così si diminuisce lo spazio usato senza penalità in termini di sicurezza.

Riguardo alla Key Management, MariaDB richiede l'utilizzo di specifici plugin che sono responsabili sia per la gestione delle chiavi di cifratura e sia per la cifratura e decifratura dei dati. Inoltre MariaDB supporta l'uso di chiavi di cifratura multiple[6]. Ogni chiave utilizza un intero a 32 bit come identificatore. Se un plugin specifico supporta la Key Rotation allora le chiavi di cifratura possono essere sostituite con una nuova versione. Riguardo alla Key Rotation, InnoDB ha a disposizione dei thread che girano in background che cifrano di nuovo tutte le pagine con la nuova chiave mentre Aria non possiede questo meccanismo automatico.

MariaDB supporta tre opzioni per la Key Management [78]:

File Key Management Plugin

Si tratta di un plugin base che gestisce le chiavi memorizzate in un file di testo in chiaro. Come protezione aggiuntiva si può cifrare questo file. Supporta chiavi di cifratura multiple mentre la key rotation non è supportata. AES_CBC e AES_CTR sono gli unici due algoritmi supportati. Può essere utile in fase di sviluppo.

AWS Key Management Plugin

Si tratta di un plugin che gestisce le chiavi utilizzando Amazon Web Services (AWS) Key Management Service (KMS) che permette di generare e memorizzare le chiavi AES sul disco, in forma cifrata, utilizzando la Customer Master Key (CMK) mantenuta in AWS KMS. All'avvio di MariaDB, il plugin decifra le chiavi di cifratura (mantenute cifrate sul disco) usando le API fornite da AWS. Successivamente MariaDB cifrerà e decifrerà i dati utilizzando la chiave AES. Questo meccanismo supporta chiavi di cifratura multiple e la key rotation.

Eperi Key Management Plugin

Si tratta di un plugin che utilizza eperi Gateway for Databases. Ciò permette di memorizzare le chiavi di cifratura su un server esterno al database fornendo così un ulteriore livello di sicurezza. Eperi Gateway inoltre supporta (opzionalmente) le operazioni di cifratura sul key server. Supporta chiavi di cifratura multiple e la key rotation.

La cifratura avviene ogni volta che vengono scritte pagine sul disco, ma solo dopo aver configurato correttamente un Key Management Plugin.

4.7 MariaDB: Security Functions

Come le precedenti implementazioni (Oracle e Microsoft), anche MariaDB fornisce delle funzioni che permettono di cifrare i dati a livello SQL Interface. Queste funzioni sono relative alla versione 10.5.11 di MariaDB [32].

[6]Le chiavi di cifratura multiple possono essere utili quando si vogliono ottenere diversi gradi di sicurezza a beneficio delle prestazioni.

MariaDB funzioni crittografiche	
Nome	Descrizione
AES_DECRYPT	Decifra i dati cifrati con AES_ENCRYPT.
AES_ENCRYPT	Cifra una stringa con l'algoritmo AES.
COMPRESS	Ritorna una stringa compressa in formato binario.
DES_DECRYPT	Decifra una strigna cifrata con DES_ENCRYPT.
DES_ENCRYPT	Cifra una stringa usando l'algoritmo 3DES.
ENCRYPT	Cifra una stringa con la funzione Unix crypt().
MD5	Calcola l'hash di una stringa usando l'algoritmo MD5.
PASSWORD	Calcola l'hash di una password.
SHA1	Calcola l'hash di una stringa usando l'algoritmo SHA-1.
SHA2	Calcola l'hash di una stringa usando l'algoritmo SHA-2.
UNCOMPRESS	Decomprime una stringa compressa con COMPRESS.
UNCOMPRESSED_LENGTH	Ritorna la lunghezza della stringa originale.

4.8 MySQL: *Data-at-Rest Encryption*

Questa sezione analizza la Transparent Data Encryption per quanto riguarda la versione 8.0 di MySQL [26].

MySQL utilizza come storage engine InnoDB (lo stesso di MariaDB). InnoDB supporta la cifratura per i dati a riposo a granularità tablespace e per la singola tabella includendo anche i log redo e undo.

InnoDB utilizza una architettura basata su due chiavi di cifratura, la Master Encryption Key e la Tablespace Key. Quando una tablespace viene cifrato con la Tablespace Key, questa chiave viene cifrata e memorizzata nell'header del tablespace. Quando una applicazione o un utente autenticato vuole accedere ai dati cifrati del tablespace, InnoDB utilizza la Master Encryption Key (MEK) per decifrare la Tablespace Key. InnoDB supporta inoltre la key rotation, per cui la MEK può cambiare se richiesto. Tutte il processo di cifratura si basa su un Key Management Plugin. Tutte le versioni di MySQL forniscono un KEYRING_FILE plugin per la gestione della Master Encryption Key. In aggiunta la MySQL Enterprise Edition offre i seguenti plugin addizionali:

- KEYRING_ENCRYPTED_FILE memorizza le chiavi in un file cifrato locale nel server;
- KEYRING_OKV include un client KMIP (KMIP 1.1) che utilizza un prodotto compatibile con KMIP come backend per la memorizzazione delle chiavi. I prodotti KMIP-compatible supportati includono tutte le soluzioni centralizzate per la Key Management come Oracle Key Vault [79], Thales Vormetric [80] e Fornetix VaultCore [81];
- KEYRING_AWS comunica con Amazon Web Services Key Management Service (AWS KMS) [82] come backend per la generazione delle chiavi mentre utilizza un file locale per la memorizzazione;
- KEYRING_HASHICORP comunica con HashiCorp Vault [59] per la memorizzazione delle chiavi.

È importante notare che nelle precedenti alternative il KEYRING_FILE e il KEYRING_ENCRYPTED_FILE plugin non sono conformi agli standard di sicurezza come PCI o FIPS che invece richiedono un sistema di gestione delle chiavi per gestire le chiavi di cifratura in modo sicuro come ad esempio

Key Vault o HSM. Nel caso di MySQL solo la versione commerciale MySQL Enterprise Transparent Data Encryption(TDE) utilizza un robusto sistema di gestione delle chiavi conforme agli standard di sicurezza.

La Data-at-Rest Encryption di MySQL supporta AES in modalità ECB (non sicura[7]), per la cifratura della Tablespace Key e in modalità CBC per la cifratura dei dati. Non è possibile usare altri tipi di algoritmi. Permette la cifratura delle colonne indicizzate in modo trasparente e a differenza di altre soluzioni supporta tutti i tipi di dato e qualsiasi lunghezza. Viene supportata la compressione dei dati prima di essere cifrati e infine non c'è un impatto sullo storage mentre l'impatto sulle performance è minimo (singola cifra percentuale) [83].

4.9 MySQL: *Security Functions*

MySQL fornisce delle API per le operazioni crittografiche a livello SQL Interface. Le funzioni sono relative alla versione 8.0 di MySQL [33].

MySQL funzioni crittografiche	
Nome	Descrizione
AES_DECRYPT	Decifra i dati cifrati con AES_ENCRYPT.
AES_ENCRYPT	Cifra una stringa con l'algoritmo AES.
COMPRESS	Ritorna una stringa compressa in formato binario.
MD5	Calcola l'hash di una stringa usando l'algoritmo MD5.
RANODM_BYTES	Ritorna un array contenenti valori random.
SHA1	Calcola l'hash di una stringa usando l'algoritmo SHA-1.
SHA2	Calcola l'hash di una stringa usando l'algoritmo SHA-2.
STATEMENT_DIGEST	Calcola l'hash di uno statement SQL.
STATEMENT_DIGEST_TEXT	Ritorna lo statement SQL normalizzato (parametrizzato con ?).
UNCOMPRESS	Decomprime una stringa compressa con COMPRESS.
UNCOMPRESSED_LENGTH	Ritorna la lunghezza della stringa originale.
VALIDATE_PASSWORD_STRENGTH	Determina la sicurezza di una password.

4.10 PostgreSQL: *PG_CRYPTO*

PostgreSQL non fornisce una implementazione per la Transparent Data Encryption ma mette a disposizione un package di funzioni crittografiche utilizzabili a livello SQL Interface. Le funzioni analizzate riguardano la versione 13 di PostgreSQL [31].

Funzioni PostgreSQL PG_CRYPTO		
	Nome	Descrizione
General Hashing Functions	Digest() Hmac()	Calcola l'hash di un dato. Calcola l'HMAC di un dato.

[7]Blocchi di testo identici in chiaro sono cifrati in blocchi identici, rivelando coì schemi ripetitivi.

Password Hashing Functions	Crypt() Gen_salt()	Calcola l'hash di una password. Genera un nuovo valore di sale random per essere usato in crypt().
PGP Encryption Functions	Pgp_sym_encrypt() Pgp_sym_decrypt() Pgp_pub_encrypt() Pgp_pub_decrypt() Pgp_key_id() Armor(), Dearmor() Pgp_armor_headers()	Cifra i dati con una chiave simmetrica PGP. Decifra un messaggio PGP con una chiave simmetrica. Cifra i dati con una chiave pubblica PGP. Decifra un messaggio con una chiave privata PGP. Estrae il Key ID della chiave PGP. Codificano/Decodificano in Base64. Estrae l'armor header dai dati.
Raw Encryption Functions	encrypt() decrypt()	Cifra dai dati. Decifra dei dati.
Random Data Functions	gen_random_bytes() gen_random_uuid()	Ritorna dei byte random crittograficamente sicuri. Ritorna un UUID (Obsoleto).

Capitolo 5

Risultati sperimentali

In questo capitolo si mostreranno i risultati sperimentali ottenuti testando le varie tecniche di cifratura sui prodotti discussi nel Capitolo 4 e facendo un confronto con le osservazioni teoriche discusse nel Capitolo 2.

5.1 Ambiente di valutazione

Tutti i dati sperimentali sono stati raccolti utilizzando un computer con le seguenti caratteristiche:

- CPU: Processore Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz;
- RAM: 8GB;
- Capacità SSD: 256 GB;
- Sistema Operativo: Windows 10.

Ogni tabella dei database di prova possiede le seguenti colonne:

Nome colonna	**Tipo di dato**
ID	int
Nome	varchar
Cognome	varchar
NumeroCartaCredito	varchar
Città	varchar

Ogni database di prova contiene due tabelle (con lo stesso schema definito sopra), una tabella con 10 mila tuple e l'altra con 100 mila tuple.

5.2 Installazione DBMS

In questa sezione si forniscono le istruzioni per installare correttamente i DBMS analizzati nel Capitolo 4. L'Appendice C contiene i comandi utilizzati per configurare e testare i vari prodotti.

Oracle

Riguardo al DBMS Oracle è stata utilizzata la versione Express Edition 18c. Innanzitutto occorre visitare la pagina web al seguente URL `https://www.oracle.com/database/technologies`

/xe-downloads.html e scegliere la piattaforma su cui si vuole installare il DBMS (Windows o Linux con architettura x86). Per poter procedere con il download bisogna effettuare il login con un account Oracle.

Dopo aver scaricato il file zip occorre estrarlo ed aprire la corrispondente cartella generata (in Windows chiamata come "OracleXE184_Win64"). Eseguire il programma di setup dell'installazione (in Windows "setup.exe") con privilegi di amministratore.

Occorre quindi accettare la licenza, scegliere il percorso di installazione e inserire la password di amministrazione del DBMS. Infine cliccare sul tasto *Installa*.

Per poter interagire con i database si può utilizzare il tool *sqlplus* da linea di comando (maggiori dettagli alla pagina https://docs.oracle.com/en/database/oracle/oracle-database/21/sqpug/index.html) oppure Oracle SQL Developer. Quest'ultimo è consigliato in quanto fornisce un'interfaccia grafica per interagire con il DBMS. È scaricabile alla pagina web al seguente URL https://www.oracle.com/tools/downloads/sqldev-downloads.html.

Microsoft

Riguardo al DBMS Microsoft è stata utilizzata la versione SQL Server Developer Edition 2019. Innanzitutto occore visitare la pagina web al seguente URL https://www.microsoft.com/it-it/sql-server/sql-server-downloads, scegliere la piattaforma e scaricare la versione Developer.

Nel caso di Windows, bisogna eseguire il programma eseguibile *.exe* scaricato con privilegi di amministratore per aprire il tool di installazione.

A questo punto occorre scegliere la modalità di installazione. Se non si hanno particolare esigenze si consiglia la modalità Standard. Accettare la licenza, scegliere il percorso di installazione e infine cliccare sul tasto *Installa*.

Per poter interagire con il DBMS installato si consiglia *Microsoft SQL Server Management Studio 2018*, l'ambiente di sviluppo integrato fornito da Microsoft. Il software è scaricabile da questa pagina web https://docs.microsoft.com/it-it/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15.

Sulla piattaforma Linux si può utilizzare il comando *sqlcmd* (maggiori dettagli alla pagina web https://docs.microsoft.com/it-it/sql/tools/sqlcmd-utility?view=sql-server-ver15) per interagire con Microsoft SQL Server.

MariaDB

Riguardo al DBMS MariaDB è stata utilizzata la versione 10.5. È possibile scaricarla al seguente URL https://mariadb.org/download/ dove occorre inserire versione, sistema operativo e architettura. Infine cliccando su Download si inizierà a scaricare i file.

Successivamente occorre aprire il programma scaricato e completare l'installazione seguendo i passi presentati dall'interfaccia.

Per interagire con MariaDB si consiglia l'utilizzo di HeidiSQL (scaricabile al seguente URL https://www.heidisql.com/download.php), uno strumento dotato di interfaccia grafica per interagire con MariaDB.

MySQL

Riguardo al DBMS MySQL è stata utilizzata la versione Community Edition 8. Innanzitutto occorre scaricare l'installer al seguente URL https://dev.mysql.com/downloads/mysql/.

Successivamente bisogna eseguire l'installer scaricato. Utilizzando l'interfaccia grafica si può installare MySQL Server ma anche altri tool come MySQL Workbench che è uno strumento molto utile e semplice da utilizzare per interagire con MySQL. Perciò, si consiglia la sua installazione. Una volta scelti i pacchetti da installare occorre accettare la licenza, inserire la password di sistema utilizzata dal DBMS ed infine cliccare su *Installa*.

PostgreSQL

Riguardo al DBMS PostgreSQL è stata utilizzata la versione 13. È possibile scaricarla al seguente URL https://www.postgresql.org/download/ dove occorre selezionare il sistema operativo. Nel caso in cui Windows sia la piattaforma dove installare il DBMS, si verrà reindirizzati su una nuova pagina contenente l'installer da scaricare.

Successivamente occorre eseguire l'installer scaricato ed eseguire i passi forniti dall'interfaccia grafica. Inoltre, è possibile scaricare altri software come PgAdmin che è uno strumento utile per interfacciarsi con PostgreSQL e perciò se ne consiglia l'installazione. Dopo aver installato i pacchetti è possibile iniziare ad utilizzare i database.

5.3 Benchmarking Framework

L'obiettivo del Benchmarking Framework è quello di fornire una metodologia per misurare l'impatto sulle performance delle tecniche crittografiche sulle operazioni del database. Il framework sviluppato confronta i vari prodotti dal punto di vista delle performance e l'impatto sullo storage sulla base di alcune considerazioni [14].

Metriche

Un database ha diverse metriche di valutazione che indicano l'efficienza e il suo stato. Le metriche di base sono l'Elapsed time, CPU time e l'occupazione su disco. Questi sono di particolare importanza in quanto i processi crittografici richiedono degli onerosi calcoli computazionali che vanno ad impattare sull'utilizzo e trasparenza del prodotto. Queste metriche verranno analizzate durante le operazioni principali sul database ovvero SELECT, INSERT, UPDATE e DELETE. L'Elapsed Time rappresenta il tempo di ritardo tra l'invio dell'input e la generazione dell'output mentre il CPU Time rappresenta il tempo effettivo in cui la CPU è stata impiegata per eseguire un determinato task. Per questo motivo ci si aspetta che l'Elapsed Time sia maggiore del CPU Time in quanto la CPU in tutto l'intervallo di tempo sarà assegnata anche ad altri task dallo scheduler.

Algoritmi

Gli algoritmi utilizzati per testare i prodotti sono AES, 3DES (solo Oracle) e Blowfish (solo PostgreSQL). Riguardo alla lunghezza della chiave, per AES 128,192 e 256 bit, per 3DES 168 bit mentre per Blowfish 128 bit.

Dimensione della tabella

Un importate criterio di valutazione è quello della dimensione della tabella. Tabelle più grandi fanno tendere il sistema test alla realtà dove è possibile apprezzare le latenze e le differenze tra i vari prodotti.

Granularità del dato da cifrare

La scelta della granularità dei dati è importante nella valutazione di un prodotto. In particolare verrà confrontata la cifratura a livello tablespace con la cifratura a livello colonna.

Stato del sistema

Per ogni test effettuato si fornisce lo stato del sistema prima, dopo e durante l'esecuzione. Ciò è fondamentale in quanto permette di stimare con maggiore precisione l'impatto sulla macchina che ospita il DBMS in termini di utilizzo CPU, RAM, disco e processi attivi. L'appendice D contiene tutte le tabelle che riassumono lo stato del sistema per ogni test eseguito.

L'obiettivo è quello di mostrare l'impatto sulle performance al variare della lunghezza della chiave, dell'algoritmo, soluzione adottata e alla granularità dei dati mettendo in relazione con lo stato del sistema.

Durante l'esecuzione dei test tutti i DBMS giravano sulla stessa macchina e i risultati ottenuti sono stati mediati tra tre campioni per ogni test in modo da attenuare il "rumore" di una singola rilevazione.

In questo lavoro di tesi si sono analizzate le soluzioni a livello DBMS Engine, SQL Interface e Client-side, occorre perciò specializzare il concetto generale di Benchmarking Framework [14] per ogni livello. Si ottengono perciò:

- DBMS Engine Benchmarking Framework;
- SQL Interface Benchmarking Framework;
- Client-side Benchmarking Framework [1].

5.3.1 DBMS Engine Benchmarking Framework

In questo caso il processo è composto da cinque step distinti come mostra la Tabella 5.1.

Step	**Procedura**
1	Creazione dell'ambiente di valutazione
2	Creazione dei database
3	Attivazione della TDE
4	Esecuzione degli scenari
5	Analisi dei risultati

Tabella 5.1: Passi del DBMS Engine Benchmarking Framework

1. Il primo step consiste nell'impostare l'ambiente di valutazione;
2. Il secondo step consiste nel costruire i database e definire gli scenari. Idealmente il database dovrebbe rappresentare un caso analogo alla teoria quindi con un numero significativo di record;
3. Il terzo step consiste nell'attivazione della TDE, la cui procedura è molto simile a tutti i prodotti analizzati;
4. Il quarto step consiste nell'eseguire le valutazioni, come precedentemente discusso;
5. Il quinto step consiste nell'analizzare e commentare i risultati ottenuti.

Gli scenari analizzati sono:

- Scenario 1: valutazione dell'impatto sullo storage della TDE per differenti: lunghezza delle chiavi, algoritmi di cifratura, dimensione della tabella e granularità;
- Scenario 2: valutazione delle performance della TDE per quanto riguarda l'Elapsed Time e CPU Time per differenti: lunghezza delle chiavi, algoritmi di cifratura, dimensione della tabella e granularità;

Le rilevazioni dello stato del sistema prima, durante e dopo sono state effettuate utilizzando i tool **Gestione Attività**, **Monitoraggio risorse** e **Tasklist** [84] messi a disposizione da Windows 10.

[1] Verrà approfondito nel Capitolo 6.

Configurazione dei DBMS

In questo paragrafo si mostra la configurazione scelta per ogni DBMS analizzato riguardo alla TDE.

Per la Microsoft SQL Server Transparent Data Encryption, è stata utilizzata la versione SQL Server Developer Edition 2019. Inoltre i database sono stati creati utilizzando la GUI fornita da Microsoft SQL Server Management Studio 18.

Riguardo alla Oracle Transparent Data Encryption, è stata utilizzata la versione Oracle Database 18c Express Edition.

Per la Data-at-rest Encryption di MariaDB è stata utilizzata la versione 10.5. Il motore utilizzato è ARIA [2],

Riguardo alla Data-at-rest Encryption di MySQL è stata utilizzata la versione MySQL Community Edition 8.0.23. Il motore scelto è InnoDB.

La Sezione 5.2 mostra i passaggi da effettuare per installare correttamente i DBMS.

La Tabella 5.2 mostra per ogni DBMS i database di prova creati.

	Microsoft	Oracle	MariaDB	MySQL
NO_TDE	x	x	x	x
TDE_AES128	x	x	x	x
TDE_AES192	x	x	x	x
TDE_AES256	x	x	x	x
TDE_AES256COLUMN		x		
TDE_3DES168		x		

Tabella 5.2: Database creati per ogni DBMS per la TDE.

- NO_TDE è il database dove la TDE non è abilitata;
- TDE_AES128 è il database dove la TDE è abilitata con AES128;
- TDE_AES192 è il database dove la TDE è abilitata con AES192;
- TDE_AES256 è il database dove la TDE è abilitata con AES256;
- TDE_AES256COLUMN è il database dove la TDE è abilitata solo sulla colonna NumeroCartaCredito con AES256;
- TDE_3DES168 è il database dove la TDE è abilitata con 3DES;

Microsoft SQL Server Developer 2019, MariaDB 10.5 TDE e MySQL 8 TDE non supportano la TDE a granularità colonna e l'algoritmo 3DES, per questa ragione non sono stati creati i corrispettivi database.

Riguardo alla Key Management, per Microsoft è stato utilizzato il workflow basato su Master Encryption Key e Certificato discusso nel Capitolo 4. Per Oracle è stato utilizzato un password-based keystore per memorizzare la Master Encryption Key. Per MariaDB è stato utilizzato il File Key Management Plugin. Infine per MySQL è stato utilizzato il Keyring File Plugin.

L'appedice C contiene tutte le istruzioni per attivare la TDE e le istruzioni SQL CRUD [3] utilizzate per effettuare i testi per ogni DBMS.

[2] Ciò permette un confronto con InnoDB (utilizzato invece in MySQL) che è un altro motore utilizzabile in MariaDB.

[3] SELECT, UPDATE, DELETE e INSERT

5.3.2 SQL Interface Benchmarking Framework

La specializzazione fatta in questo caso consiste nell'aggiungere l'approccio nell'utilizzo dell'interfaccia SQL, oltre alle metriche, agli algoritmi e dimensione della tabella ereditati dal concetto più generale del Benchmark Framework. Gli approcci possibili sono i trigger e la chiamata diretta delle API crittografiche. Come mostrato nel Capitolo 2 l'utilizzo dei trigger permette di automatizzare i processi crittografici ma a differenza della TDE non è completamente trasparente e si hanno degli svantaggi per le operazioni di SELECT. Invece chiamando direttamente le API crittografiche si annulla la trasparenza permettendo però il corretto funzionamento per tutte le operazioni.

In questo lavoro di tesi si utilizzano le chiamate dirette delle API crittografiche fornite dai DBMS in quanto così si annullano i vincoli di utilizzo permettendo un confronto con le soluzioni TDE. Inoltre utilizzando questo approccio si mettono in evidenza le problematiche principali di questa soluzione ovvero il Key Management esplicito, l'assenza di trasparenza e l'aumento significativo della complessità delle query come si può notare dalle query di test nell' appendice C.

Quindi gli scenari possibili sono:

- Scenario 1: valutazione dell'impatto sullo storage per differenti: lunghezza delle chiavi, algoritmi di cifratura, dimensione della tabella e granularità;
- Scenario 2: valutazione delle performance per quanto riguarda l'Elapsed Time e CPU Time per differenti: lunghezza delle chiavi, algoritmi di cifratura, dimensione della tabella e granularità;

Le rilevazioni sullo stato del sistema prima, durante e dopo l'esecuzione dei test sono state effettuate con gli stessi strumenti elencati nel DBMS Engine Benchmarking Framework (Sezione 5.3.1).

Configurazione dei DBMS

In questo paragrafo si mostra la configurazione scelta per ogni DBMS analizzato riguardo alle soluzioni SQL Interface.

Per Oracle è stata utilizzata la versione di DBMS_CRYPTO relativo Oracle Database 18c Express. Per quanto riguarda Microsoft è stata utilizzata la versione delle funzioni crittografiche TRANSACT-SQL relative a Microsoft SQL Developer 2019. Per MySQL è stata utilizzata la versione delle funzioni relativa a MySQL Comunnity Edition 8.0.23. Infine per quanto riguarda PostgreSQL è stata utilizzata la versione delle funzioni relativa a PostgreSQL 13. La Sezione 5.2 mostra i passaggi da effettuare per installare correttamente i DBMS.

La Tabella 5.3 mostra, per ogni DBMS, i database di prova creati per testare le soluzioni SQL Interface.

	Oracle	Microsoft	MySQL	PostgreSQL
NO_ENC	x	x	x	x
AES128	x	x	x	x
AES192	x	x	x	x
AES256	x	x	x	x
AES256COLUMN	x	x	x	x
THREE_DES	x			
BLOWFISH				x

Tabella 5.3: Database per testare le soluzioni SQL Interface creati per ogni DBMS.

- NO_ENC è il database di prova non cifrato;

- AES128 è il database di prova cifrato con AES128;
- AES192 è il database di prova cifrato con AES192;
- AES256 è il database di prova cifrato con AES256;
- AES256COLUMN è il database di prova dove è cifrata solo la colonna NumeroCartaCredito con AES256;
- THREE_DES è il database di prova cifrato con 3DES.
- BLOWFISH è il database di prova cifrato con Blowfish.

Solo Oracle Database 18c Express supporta 3DES per questa ragione non è stato creato il corrispettivo database negli altri DBMS. Stessa considerazione anche per Blowfish e PostgreSQL 13.

L'appendice C mostra le istruzioni eseguite per testare le varie soluzioni a livello SQL Interface.

5.4 Risultati sperimentali TDE

5.4.1 Valutazione dell'impatto sullo storage

La Figura 5.1 mostra la dimensione dei database di prova al variare dell'algoritmo utilizzato e della granularità (solo Oracle) per ogni prodotto analizzato. Di particolare interesse è il fatto che l'abilitazione della TDE a livello Tablespace non ha impatto sullo storage, infatti la dimensione rimane costante al variare dell'algoritmo in ogni prodotto. Mentre l'abilitazione della TDE a livello colonna (Oracle) ha un impatto considerevole (+60%) sullo storage in quanto i valori cifrati devono essere multipli di 16 bytes e ogni valore cifrato ha associato un integrity check di 20 byte. Inoltre, siccome la cifratura avviene a livello di pagina, se la dimensione dei dati è minore della dimensione della pagina non è possibile inserire altri dati in quella pagina. Ciò può comportare un elevato spazio inutilizzato nella pagina. Infine ogni DBMS associa per ogni database dei metadati diversi e ciò giustifica le diverse dimensioni occupate.

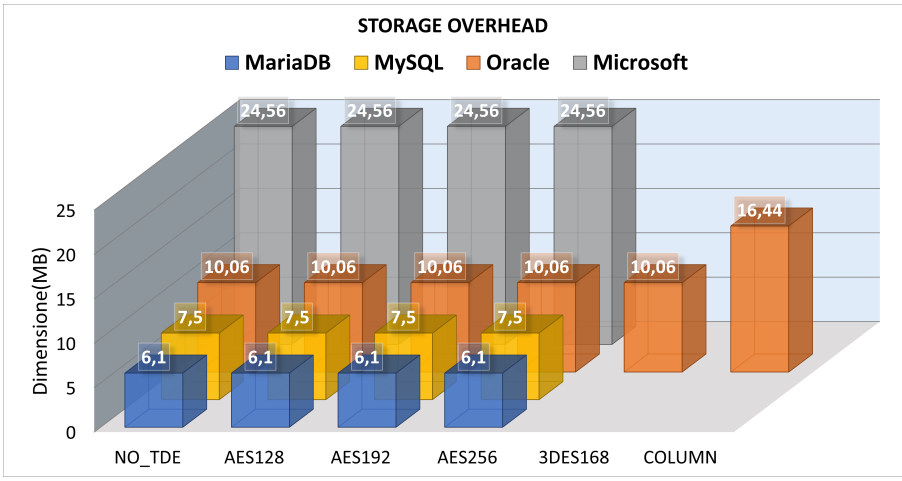


Figura 5.1: *TDE* - Confronto Storage Overhead tra i diversi prodotti

5.4.2 Valutazione dell'Elapsed Time e e CPU Time

SELECT

Le Figure 5.2 e 5.3 mostrano l'Elapsed Time e CPU Time sull'operazione SELECT al variare dell'algoritmo, granularità e dimensione della tabella. Si nota che le due metriche aumentano

all'aumentare della dimensione della tabella e della lunghezza della chiave in quanto gli algoritmi con chiave più lunga forniscono una maggiore sicurezza ma hanno un maggior impatto computazionale. Inoltre, come si può notare, 3DES è più lento di AES [85][4].

La cifratura a livello colonna comporta una minor perdita di prestazioni, come prevedibile, poichè il numero di operazioni crittografiche eseguite per decifrare i dati[5] si limitano ad una sola colonna piuttosto che a tutta la tabella. In questo caso l'incremento dell'Elapsed Time è negativo in quanto per casualità l'attesa è stata inferiore al caso in cui la TDE non è abilitata. In ogni caso l'incremento del CPU Time è positivo mettendo in evidenza il fatto che c'è stato un aumento della complessità della computazione.

Aria (il motore di storage usato in MariaDB) è il più performante, mentre Microsoft SQL Server ha le performance peggiori in termini assoluti mentre in termini percentuali Oracle ha il maggior impatto.

La Tabella 5.4 mostra l'incremento percentuale dell'Elapsed Time e CPU Time rispetto al database non cifrato per l'operazione di SELECT. I dati si riferiscono alle tabelle da 100k tuple. I valori di AES (AES128, AES192 e AES256) sono stati mediati [6].

Le Tabelle D.1 D.2 D.3 e D.4 (Appendice D) mostrano lo stato del sistema prima, durante e dopo l'esecuzione dei test.

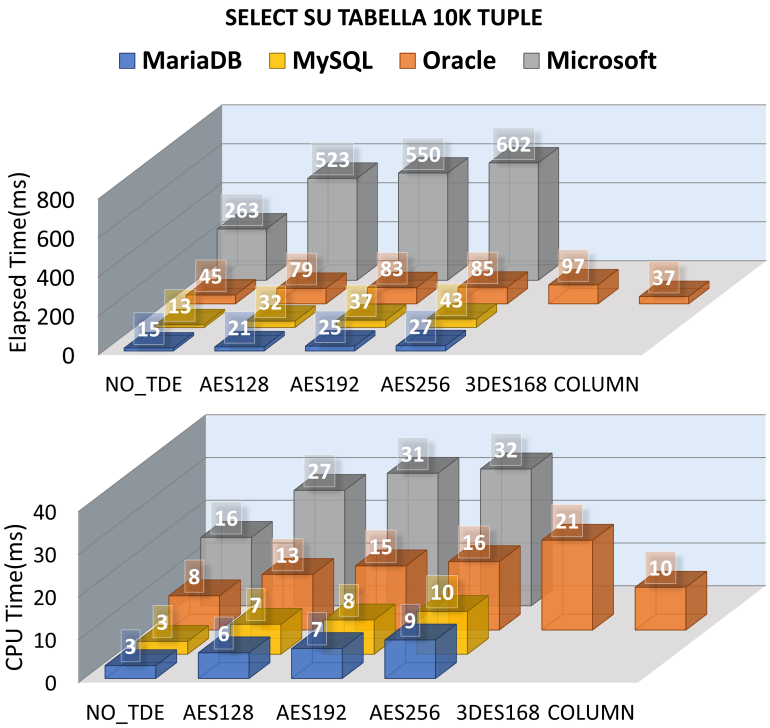


Figura 5.2: *TDE* - Confronto dell'Elapsed Time e CPU Time(SELECT) sulle tabelle da 10k tuple.

[4]In generale AES è più veloce e sicuro rispetto a 3DES.

[5]Nell'operazione di SELECT, i dati vengono in modo trasparente decifrati e restituiti all'utente.

[6]Anche nelle altre operazioni si fanno queste considerazioni. Per tale ragione questa spiegazione verrà omessa.

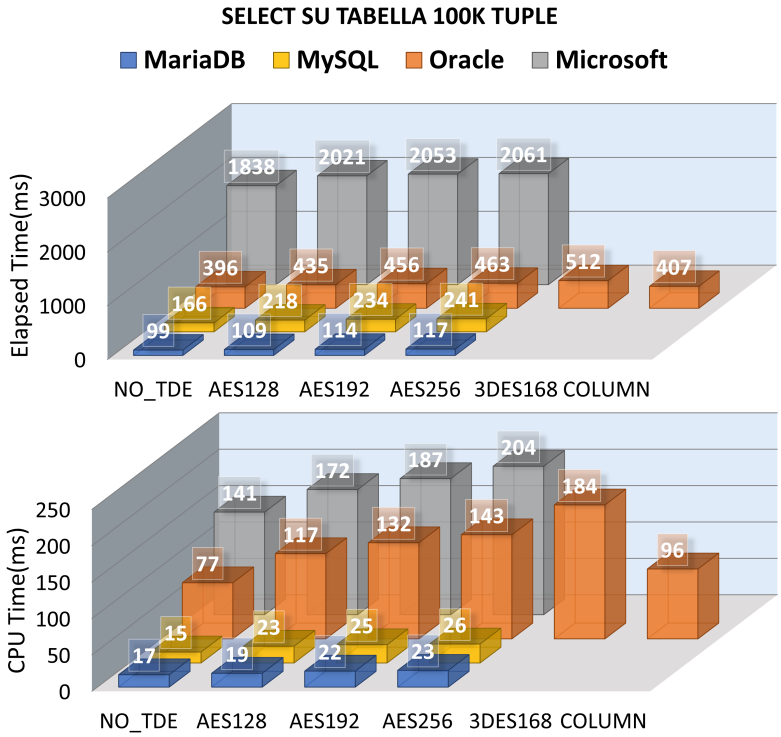


Figura 5.3: *TDE* - Confronto dell'Elapsed Time e CPU Time(SELECT) sulle tabelle da 100k tuple.

		MariaDB	MySQL	Microsoft	Oracle
AES	Elapsed Time	+14%	+39%	+11%	+14%
	CPU Time	+25%	+64%	+33%	+70%
COLUMN	Elapsed Time	-	-	-	-17%
	CPU Time	-	-	-	+25%
3DES	Elapsed Time	-	-	-	+116%
	CPU Time	-	-	-	+162%

Tabella 5.4

UPDATE

Le Figure 5.4 e 5.5 mostrano l'Elapsed Time e CPU Time sull'operazione UPDATE al variare dell'algoritmo, granularità e dimensione della tabella. Anche in questo caso 3DES è più lento rispetto ad AES e la cifratura a livello colonna ha un minor impatto sulle performance. L'incremento del CPU Time è giustificato dal fatto che nel caso dell'UPDATE i dati vengono, in modo trasparente, prima cifrati e poi memorizzati sul disco. Dato che la cifratura comporta un overhead computazionale, il CPU Time aumenta. Rispetto all'INSERT (analizzata successivamente) l'incremento del CPU Time è minore in quanto il numero di dati da cifrare è minore (solo un campo, NumeroCartaCredito).

Le Tabelle D.9 D.10 D.11 e D.12 (Appendice D) mostrano lo stato del sistema prima, durante e dopo l'esecuzione dei test.

MariaDB risulta essere il DBMS più veloce in termini assoluti mentre l'incremento del CPU Time in Oracle è più accentuato rispetto alle altre soluzioni.

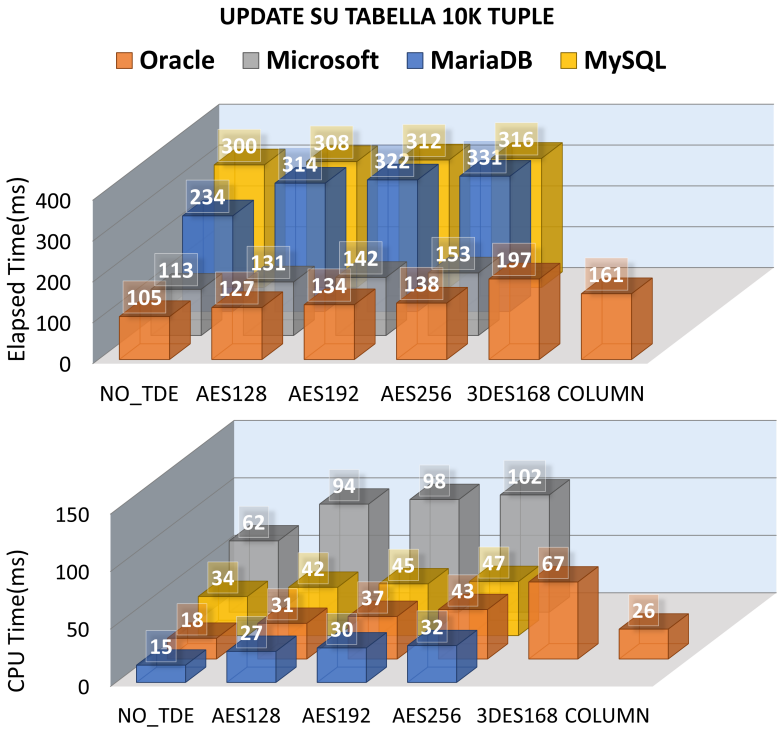


Figura 5.4: *TDE* - Confronto dell'Elapsed Time e CPU Time (UPDATE) sulle tabelle da 10k tuple.

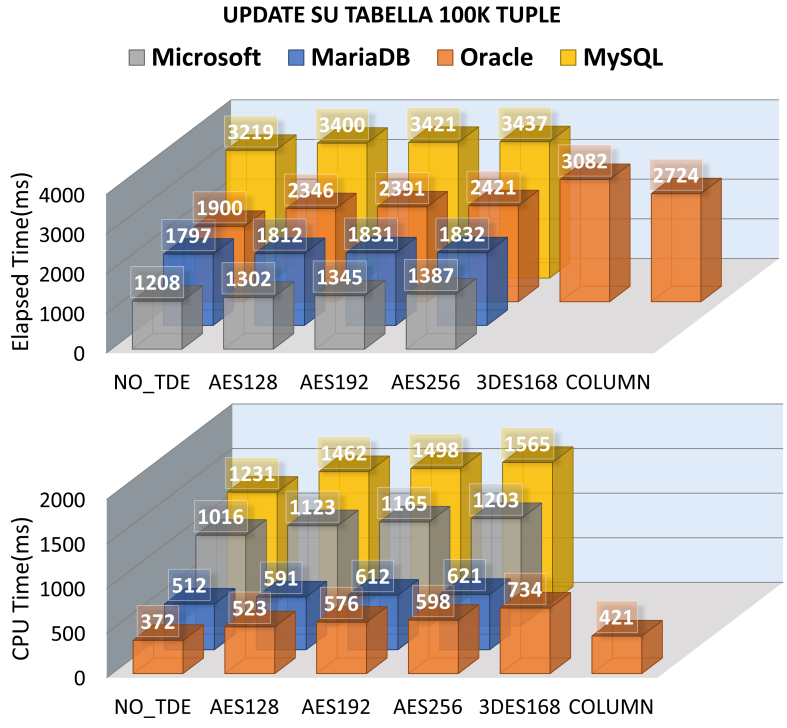


Figura 5.5: *TDE* - Confronto dell'Elapsed Time e CPU (UPDATE) sulle tabelle da 100k tuple.

		MariaDB	**MySQL**	**Microsoft**	**Oracle**
AES	Elapsed Time	+2%	+6%	+11%	+26%
	CPU Time	+18%	+23%	+15%	+52%
COLUMN	Elapsed Time	-	-	-	+43%
	CPU Time	-	-	-	+13%
3DES	Elapsed Time	-	-	-	+62%
	CPU Time	-	-	-	+97%

Tabella 5.5

INSERT

Le Figure 5.6 e 5.7 mostrano l'Elapsed Time sull'operazione INSERT al variare dell'algoritmo, granularità e dimensione della tabella. Si nota che per Microsoft e Oracle l'impatto della TDE è minimo all'aumentare della lunghezza della chiave, come mostrato nella tabella 5.6. In questo caso la cifratura a livello colonna ha un enorme impatto sulle performance. Ciò è dovuto al fatto che Oracle, con questa granularità, non supporta l'utilizzo degli algoritmi bulk per ottimizzare le performance, come discusso nella Sezione 4.1.

Per il resto, l'incremento del CPU Time è dovuto al fatto che nel caso dell'operazione INSERT (come nell'UPDATE) i dati vengono prima cifrati, in modo trasparente, e poi memorizzati sul disco. Dato che la cifratura ha un overhead computazionale, il CPU Time aumenta.

Le Tabelle D.5 D.6 D.7 e D.8 (Appendice D) mostrano lo stato del sistema prima, durante e dopo l'esecuzione dei test.

Anche in questo caso, MariaDB (e quindi Aria) risulta essere il DBMS più veloce in termini assoluti.

		MariaDB	MySQL	Microsoft	Oracle
AES	Elapsed Time	+6%	+21%	+16%	+5%
	CPU Time	+32%	+61%	+18%	+15%
COLUMN	Elapsed Time	-	-	-	+57%
	CPU Time	-	-	-	+157%
3DES	Elapsed Time	-	-	-	+27%
	CPU Time	-	-	-	+21%

Tabella 5.6

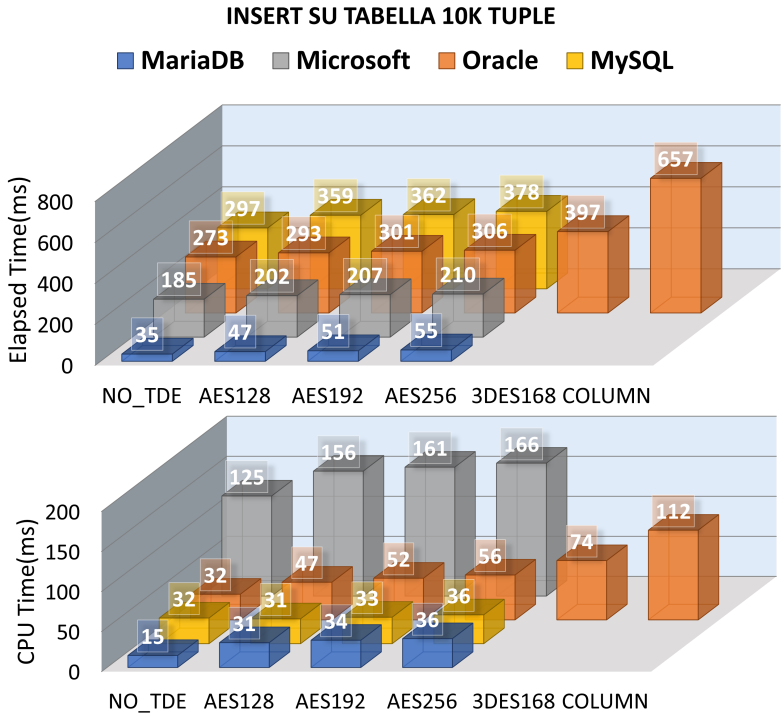


Figura 5.6: *TDE* - Confronto dell'Elapsed Time e CPU Time (INSERT) sulle tabelle da 10k tuple.

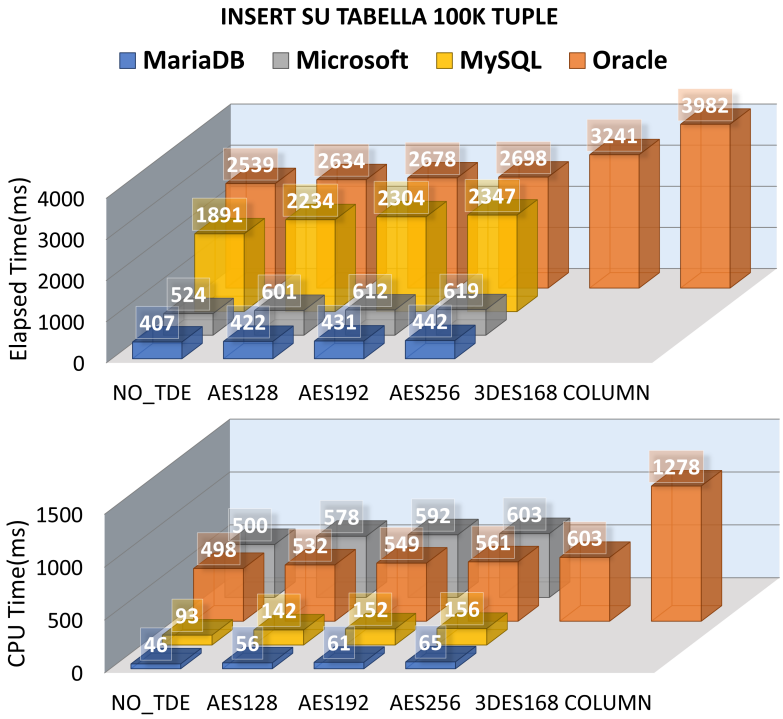


Figura 5.7: *TDE* - Confronto dell'Elapsed Time e CPU Time (INSERT) sulle tabelle da 100k tuple.

DELETE

Le Figure 5.8 e 5.9 mostrano l'Elapsed Time e CPU Time sull'operazione DELETE al variare dell'algoritmo, granularità e dimensione della tabella. Come mostrato nella Tabella 5.7, l'operazione di DELETE ha un impatto minimo o nullo per tutti i prodotti tranne Oracle dove l'overhead computazionale è accentuato. Ciò è giustificato dal fatto che durante l'operazione di DELETE non viene coinvolta nessuna operazione crittografica.

Le Tabelle D.13 D.14 D.15 e D.16 (Appendice D) mostrano lo stato del sistema prima, durante e dopo l'esecuzione dei test.

		MariaDB	MySQL	Microsoft	Oracle
AES	Elapsed Time	+3%	-4%	+0%	+22%
	CPU Time	+12%	-7%	+10%	+30%
COLUMN	Elapsed Time	-	-	-	+15%
	CPU Time	-	-	-	+20%
3DES	Elapsed Time	-	-	-	+27%
	CPU Time	-	-	-	+37%

Tabella 5.7

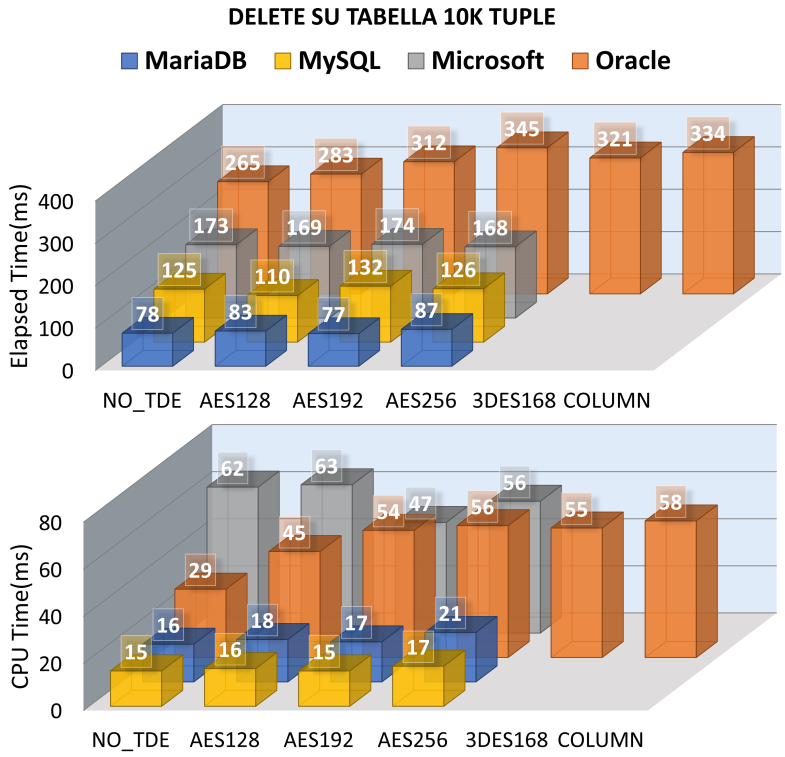


Figura 5.8: *TDE* - Confronto dell'Elapsed Time e CPU Time (DELETE) sulle tabelle da 10k tuple.

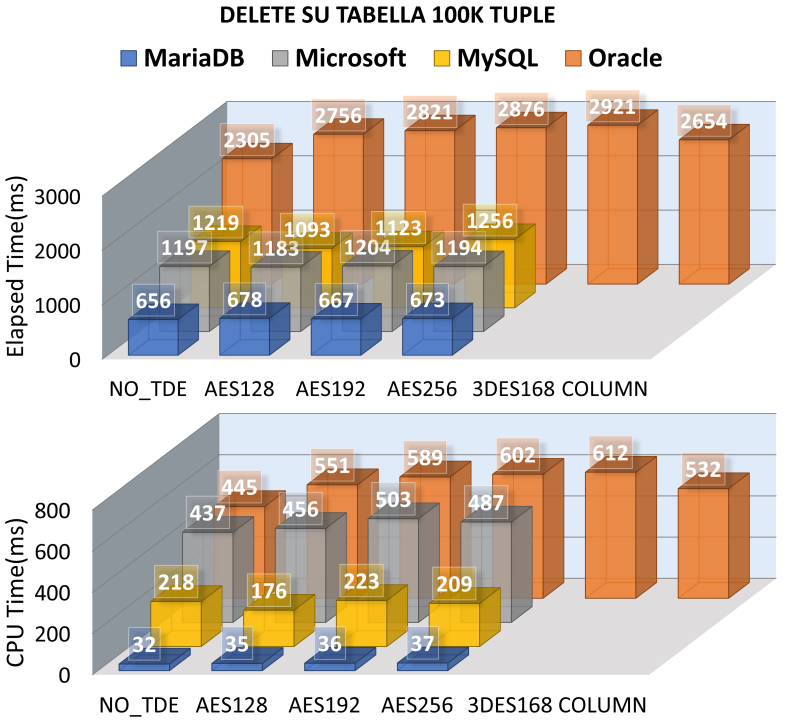


Figura 5.9: *TDE* - Confronto dell'Elapsed Time e CPU Time (DELETE) sulle tabelle da 100k tuple.

5.5 Risultati sperimentali SQL Interface

5.5.1 Scenario 1: Valutazione dell'impatto sullo storage

La Figura 5.10 mostra la dimensione dei database di prova al variare dell'algoritmo utilizzato e della granularità per ogni prodotto analizzato. Come si può notare l'impatto sullo storage di PostgreSQL è enorme e la grandezza dei database cifrati con AES è di circa 10 volte il database non cifrato. Invece Oracle non ha impatti sullo storage riguardo alla cifratura con AES mentre per 3DES c'è un aumento del 64%.

Rispetto alla Figura 5.1 che mostra l'impatto sullo storage della TDE si può notare come la soluzione SQL Interface sia assolutamente più inefficiente.

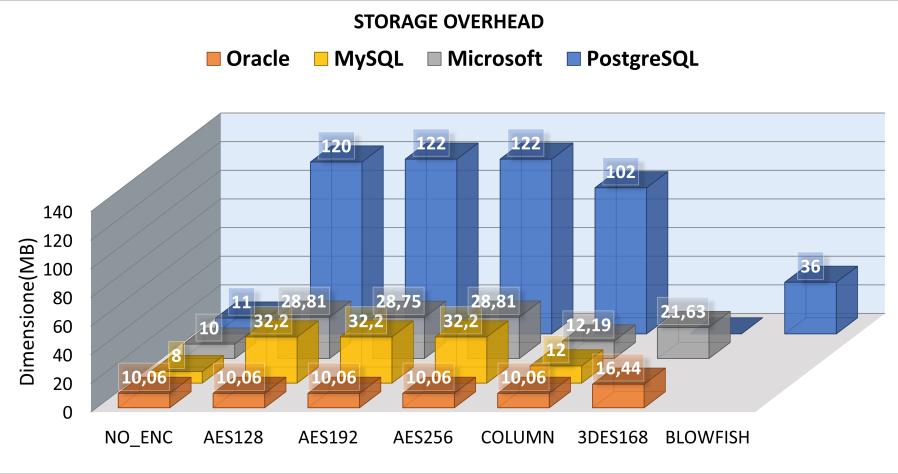


Figura 5.10: *SQL Interface* - Confronto Storage Overhead tra i diversi prodotti.

5.5.2 Scenario 2: Valutazione dell'Elapsed Time e CPU Time

INSERT

		PostgreSQL	MySQL	Microsoft	Oracle
AES	Elapsed Time	+212%	+95%	+171%	+124%
	CPU Time	+395%	+373%	+90%	+39%
COLUMN	Elapsed Time	+16%	-14%	+59%	-27%
	CPU Time	+306%	+283%	+30%	+9%
3DES	Elapsed Time	-	-	-	+202%
	CPU Time	-	-	-	+105%
Blowfish	Elapsed Time	167%	-	-	-
	CPU Time	+348%	-	-	-

Tabella 5.8

Le Figure 5.11 e 5.12 mostrano l'Elapsed Time e CPU Time riguardo l'operazione di INSERT. La Tabella 5.8 mostra l'incremento percentuale dell'Elapsed Time e CPU Time per ogni soluzione analizzata. Si può notare come PostgreSQL e MySQL abbiano le performance peggiori e l'utilizzo dell'algoritmo Blowfish è tendenzialmente meno pesante a livello computazionale rispetto ad AES [86]. La soluzione Microsoft è la più efficiente con il minor impatto sulle performance.

Si può notare che l'Elapsed Time per la cifratura a livello colonna in MySQL e Oracle è negativa. Questo apparente aumento di performance è pura casualità in quanto l'incremento corrispondente del CPU Time è positivo. Ciò è giustificato dal fatto che c'è un overhead computazionale

dovuto all'esecuzione di operazioni crittografiche (minori in numero rispetto alla cifratura per tutta la tabella).

Le Tabelle D.21 D.17 D.25 e D.29 (Appendice D) mostrano lo stato del sistema prima, durante e dopo l'esecuzione dei test.

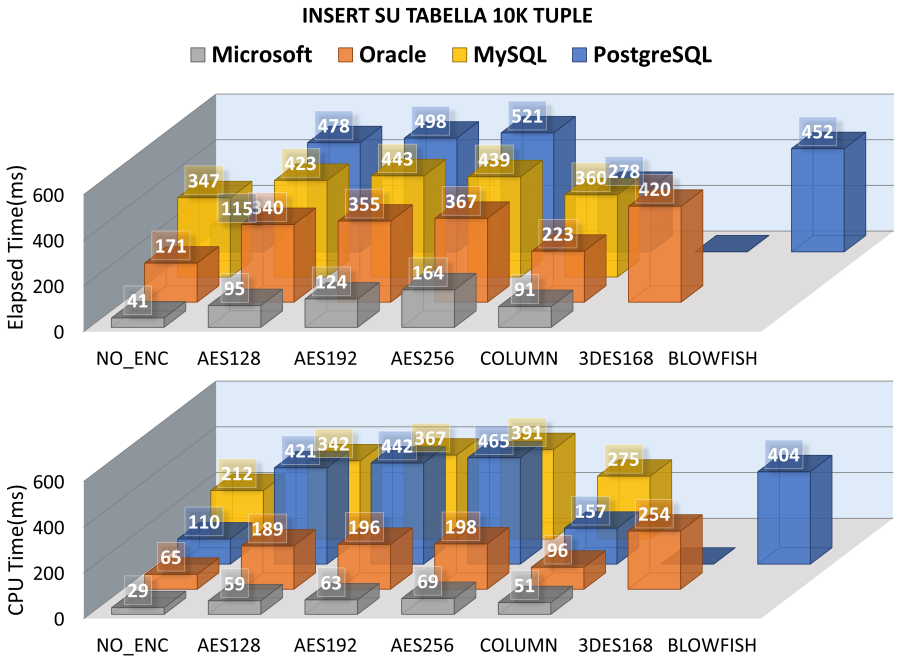


Figura 5.11: *SQL Interface* - Confronto dell'Elapsed Time e CPU Time (INSERT) sulle tabelle da 10k tuple.

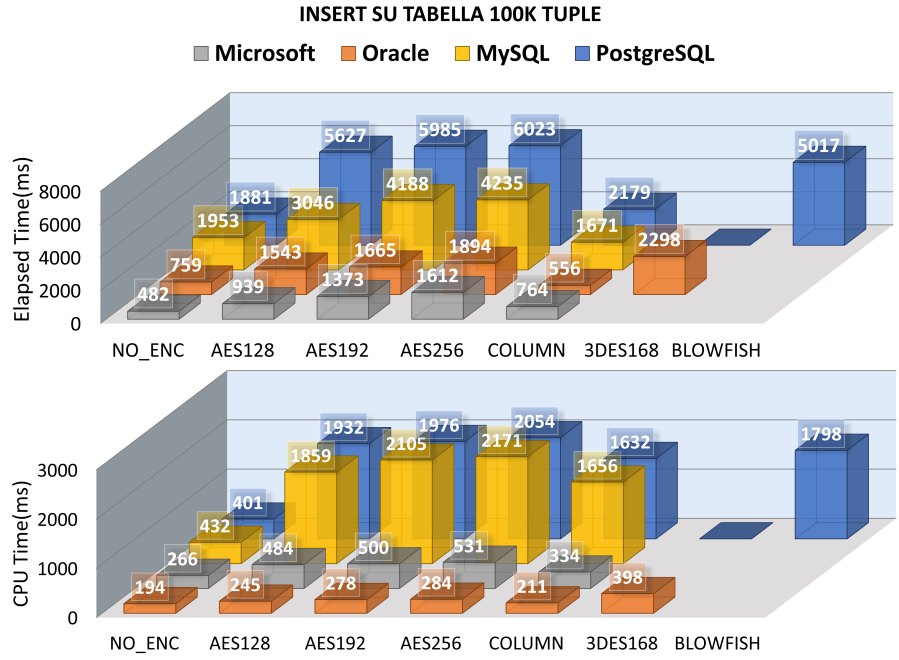


Figura 5.12: *SQL Interface* - Confronto dell'Elapsed Time e CPU Time (INSERT) sulle tabelle da 100k tuple.

SELECT

Le Figure 5.13 e 5.14 mostrano l'Elapsed Time e CPU Time riguardo l'operazione di SELECT. La Tabella 5.9 mostra l'incremento percentuale dell'Elapsed Time e CPU Time per ogni soluzione analizzata. Anche in questo caso PostgreSQL è altamente inefficiente mentre l'impatto sulle performance per quanto riguarda Microsoft è minimo rispetto alle altre soluzioni. Riguardo Oracle l'impatto è significativo e 3DES è tendenzialmente più lento rispetto ad AES. Anche in questo caso Blowfish ha performance migliori rispetto ad AES.

Da questi risultati sperimentali si evince che l'utilizzo delle funzioni crittografiche, utilizzate per decifrare i dati, messe a disposizioni dalle librerie SQL Interface siano altamente inefficienti rispetto alle funzioni di cifratura (utilizzate nella INSERT) e alla TDE.

Le Tabelle D.22 D.18 D.26 e D.30 (Appendice D) mostrano lo stato del sistema prima, durante e dopo l'esecuzione dei test.

		PostgreSQL	MySQL	Microsoft	Oracle
AES	Elapsed Time	+3187%	+384%	+8%	+650%
	CPU Time	+11970%	+4076%	+412%	+1195%
COLUMN	Elapsed Time	+262%	+190%	+3%	+292%
	CPU Time	+4487%	+1829%	+100%	+492%
3DES	Elapsed Time	-	-	-	+962%
	CPU Time	-	-	-	+1415%
Blowfish	Elapsed Time	+2666%	-	-	-
	CPU Time	+11037%	-	-	-

Tabella 5.9

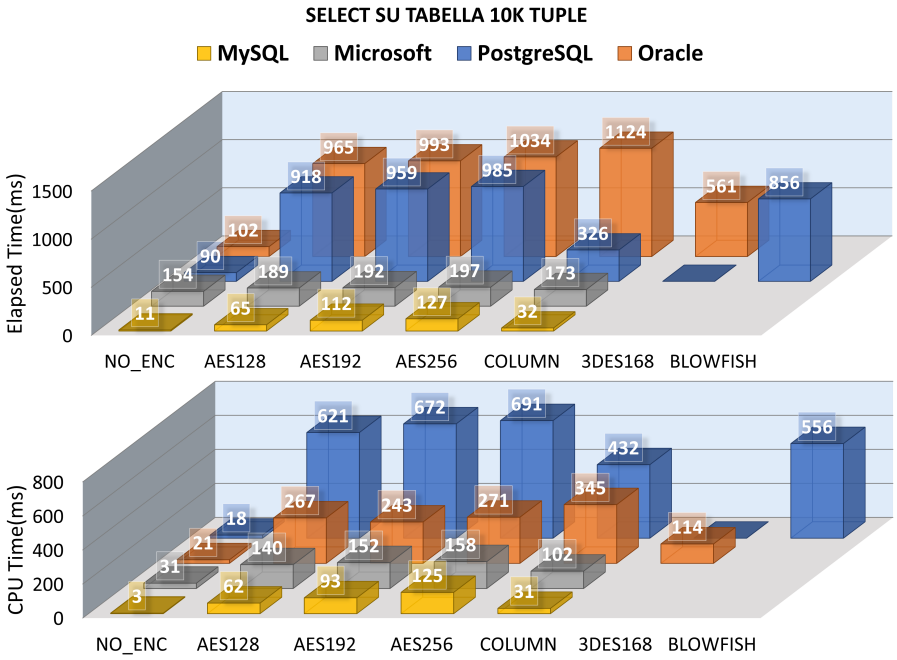


Figura 5.13: *SQL Interface* - Confronto dell'Elapsed Time e CPU Time (SELECT) sulle tabelle da 10k tuple.

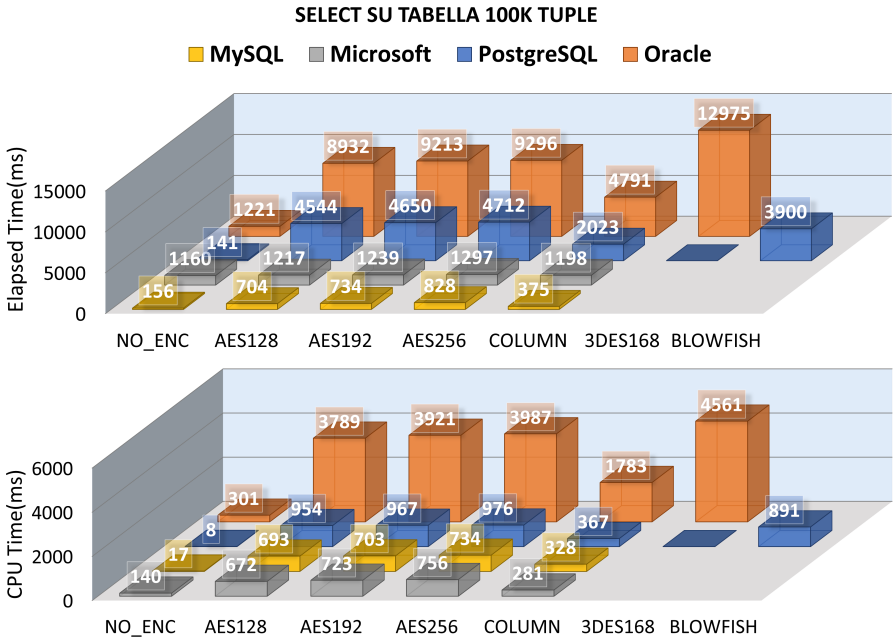


Figura 5.14: *SQL Interface* - Confronto dell'Elapsed Time e CPU Time (SELECT) sulle tabelle da 100k tuple.

UPDATE

Le Figure 5.15 e 5.16 mostrano l'Elapsed Time e il CPU Time riguardo l'operazione di UPDATE. La Tabella 5.10 mostra l'incremento percentuale dell'Elapsed Time e CPU Time per ogni soluzione analizzata. Anche in questo caso Blowfish appare più inefficiente rispetto ad AES. L'impatto sulle performance di Oracle è notevole con 3DES che aumenta le latenze rispetto ad AES. MySQL ha le migliori performance in questo caso. Microsoft e PostgreSQL hanno delle perdite contenute. Rispetto alla INSERT, in questo caso si cifra una singola colonna e la si aggiorna sul database per questa ragione le operazioni crittografiche sono minori. Ciò comporta un aumento dell'overhead computazionale ma comunque minore rispetto alla INSERT.

Le Tabelle D.23 D.19 D.27 e D.31 (Appendice D) mostrano lo stato del sistema prima, durante e dopo l'esecuzione dei test.

		PostgreSQL	MySQL	Microsoft	Oracle
AES	Elapsed Time	+62%	+9%	+58%	+196%
	CPU Time	+134%	+48%	+135%	+942%
COLUMN	Elapsed Time	+21%	+2%	+5%	+79%
	CPU Time	+34%	+21%	+43%	+662%
3DES	Elapsed Time	-	-	-	+273%
	CPU Time	-	-	-	+1060%
Blowfish	Elapsed Time	+37%	-	-	-
	CPU Time	+109%	-	-	-

Tabella 5.10

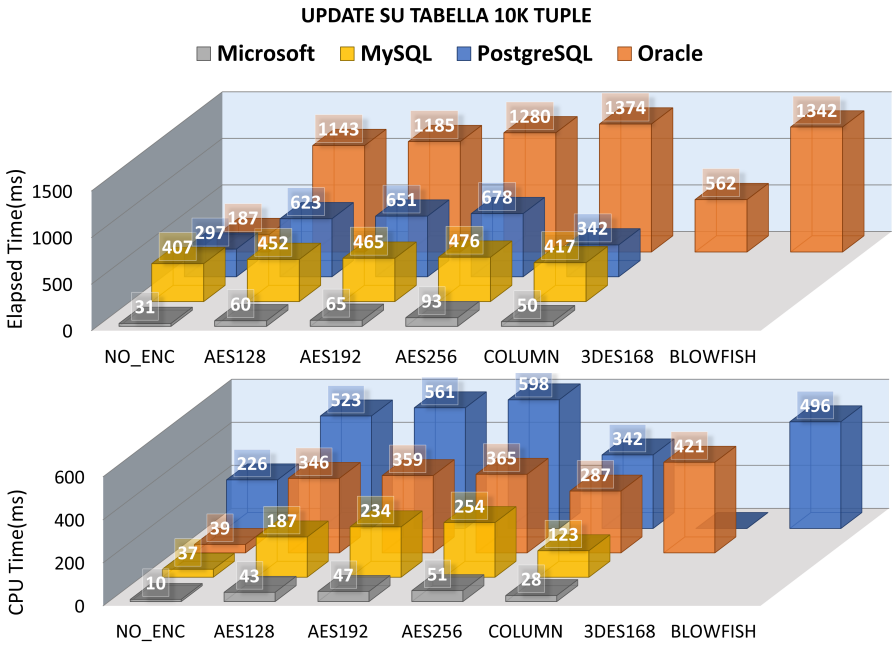


Figura 5.15: *SQL Interface* - Confronto dell'Elapsed Time e CPU Time (UPDATE) sulle tabelle da 10k tuple.

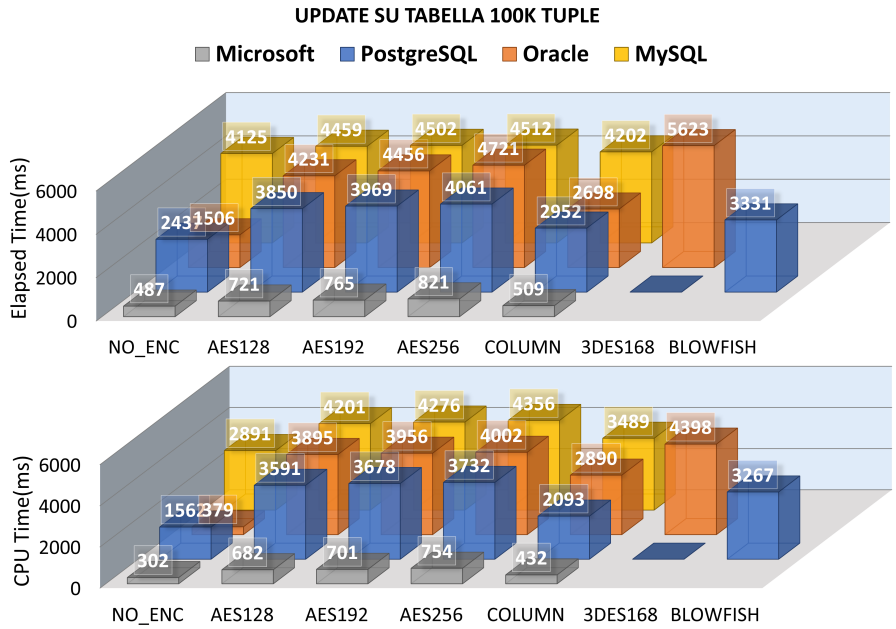


Figura 5.16: *SQL Interface* - Confronto dell'Elapsed Time e CPU Time (UPDATE) sulle tabelle da 100k tuple.

DELETE

Le Figure 5.17 e 5.18 mostrano l'Elapsed Time e il CPU Time riguardo l'operazione di DELETE. La Tabella 5.11 mostra l'incremento percentuale dell'Elapsed Time e CPU Time per ogni soluzione analizzata. Come si può notare dai grafici l'andamento del CPU Time non segue un criterio e l'incremento percentuale è circa nullo e costante. Ciò è dovuto al fatto che in questa operazioni non si effettuano operazioni crittografiche che non apportano overhead computazionale.

		PostgreSQL	MySQL	Microsoft	Oracle
AES	Elapsed Time	+74%	+122%	+113%	-3%
	CPU Time	-1%	0%	+1%	+2%
COLUMN	Elapsed Time	-33%	+10%	+77%	-41%
	CPU Time	+4%	0%	+7%	-28%
3DES	Elapsed Time	-	-	-	-9%
	CPU Time	-	-	-	-12%
Blowfish	Elapsed Time	+28%	-	-	-
	CPU Time	+1%	-	-	-

Tabella 5.11

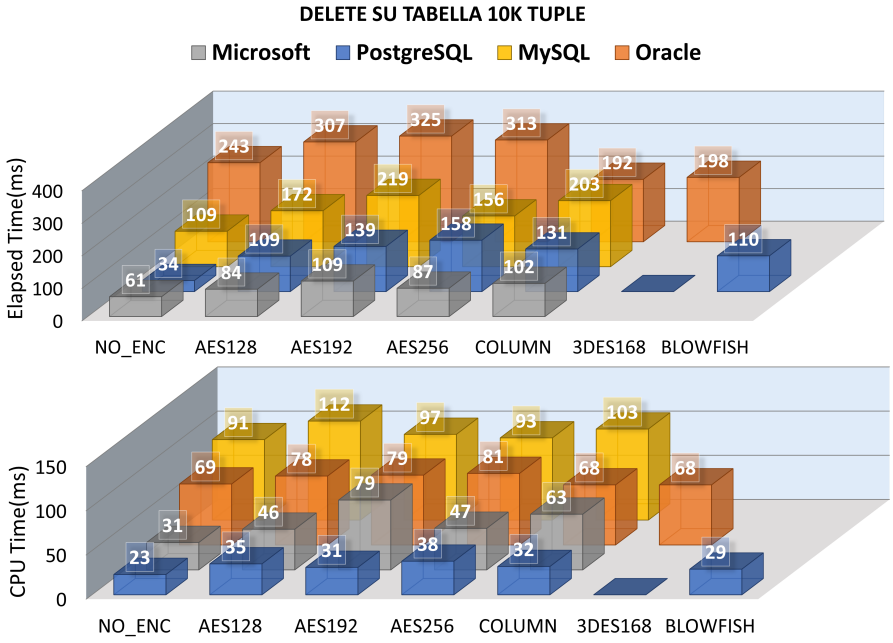


Figura 5.17: *SQL-Interface* - Confronto dell'Elapsed Time e CPU Time (DELETE) sulle tabelle da 10k tuple.

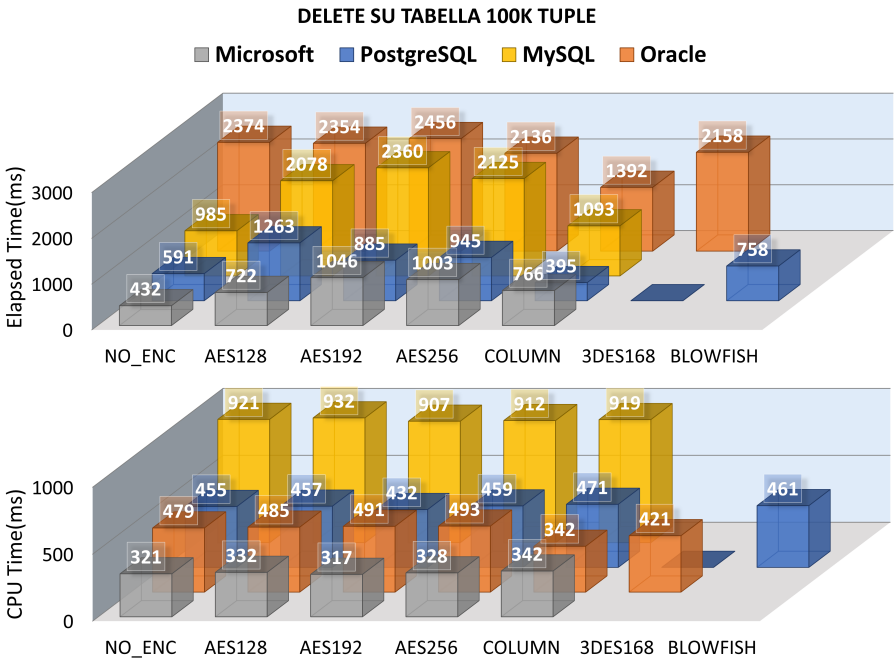


Figura 5.18: *SQL Interface* - Confronto dell'Elapsed Time e CPU Time (DELETE) sulle tabelle da 100k tuple.

Capitolo 6

Sviluppo soluzione Client-side

In questo capitolo si mostrerà l'architettura e la Key Management adottata durante lo sviluppo del driver software per l'abilitazione della client-side encryption. Il repository che contiene il codice sorgente si trova al seguente URL `https://github.com/LorenzoCeccarelli/Tesi`. È stata utilizzata la versione 11 di Java, tutte le classi utilizzate si riferiscono a questa versione.

L'appendice A contiene il *Manuale Utente*. Questo fornisce tutte le informazioni necessarie per installare e utilizzare il software sviluppato.

L'appendice B contiene il *Manuale Programmatore*. Questo, invece, fornisce la documentazione del codice e delle classi sviluppate. Inoltre sono descritte anche tutte le dipendenze necessarie per far funzionare il driver.

6.1 Architettura

La Figura 6.1 mostra lo scenario di utilizzo del software sviluppato per l'abilitazione della client-side encryption.

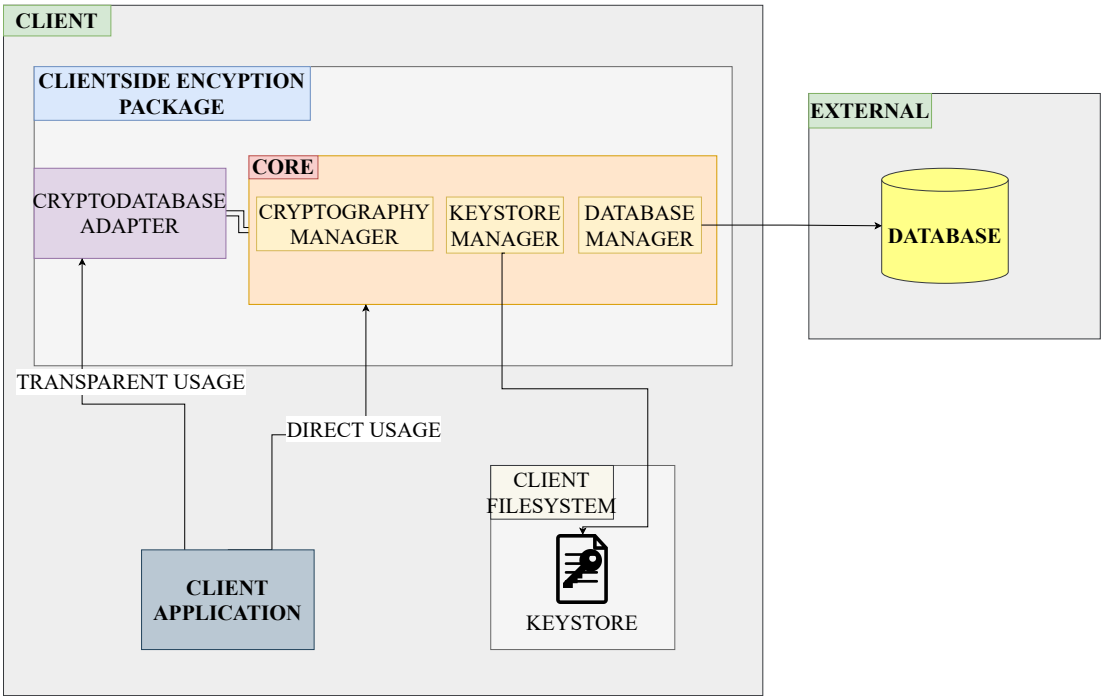


Figura 6.1: Scenario

Il software sviluppato, contenuto nel package *ClientsideEncryption*, si comporta come un driver (o adattatore) che si interfaccia con il database. Un'applicazione client può utilizzare le classi fornite dal package in due possibili modalità:

- **Trasparente**: consiste nell'utilizzare la classe *CryptoDatabaseAdapter* che permette di richiamare i metodi forniti dal package *Core* in modo trasparente all'applicazione client;
- **Diretto**: consiste nell'utilizzare direttamente le classi fornite dal package *Core* che rappresentano il nucleo di funzionamento del driver. In questo modo il programmatore ha maggiore flessibilità aumentando però la complessità del codice.

È stata fatta una semplificazione nell'architettura ovvero è stato assunto che l'applicazione client comunicasse direttamente con il DBMS senza terze parti nel mezzo (ad esempio un web server).

Il package *Core* è composto da:

- *CryptographyManager*: fornisce dei metodi per la gestione delle operazioni crittografiche;
- *KeystoreManager*: fornisce dei metodi per la gestione del keystore;
- *DatabaseManager*: fornisce dei metodi per gestire la connessione con il database e l'esecuzione delle query.

Riguardo al keystore, viene utilizzato un file PKCS12. Come discusso nel Capitolo 3, ciò rappresenta una buona soluzione in fase di sviluppo ma non fase di produzione in quanto il file .p12 non è l'alternativa più sicura a differenza dell'utilizzo di un HSM, che invece rappresenta la soluzione più efficace in termini di sicurezza. Dato che viene usata la classe *Keystore* del package *java.security* [87], la migrazione verso altre soluzioni (ad esempio HSM) in fase di produzione richiede un basso sforzo in quanto questa classe astrae rispetto alla tipologia di keystore utilizzato.

6.2 Key Management

Per quanto riguardo la gestione delle chiavi viene utilizzata un'architettura a due livelli. La chiave principale, chiamata Master Encryption Key (MEK), viene memorizzata nel keystore e serve a cifrare/decifrare le Encryption Key (EK) generate on-the-fly ogni volta in cui si necessita la cifratura di un dato. Le EK cifrate vengono inserite in un *Token*[1] che rappresenta, in formato stringa, il dato salvato nel database.

La Figura 6.2 mostra i passi eseguiti per cifrare i dati.

1. Il CryptographyManager genera una nuova Encryption Key (EK);
2. L'EK cifra i dati. I dati cifrati vengono inseriti nel Token;
3. Il KeystoreManager recupera dal keystore la MEK;
4. Il KeystoreManager utilizza la MEK per cifrare l'EK;
5. L'EK cifrata viene inserita nel Token.

La Figura 6.3 mostra i passi eseguiti per decifrare i dati.

1. Il DatabaseManager riceve dal database il Token;
2. Il TokenParser estrae la chiave cifrata e i dati dal token;

[1] Nel Manuale Programmatore Sezione B.2 è descritto il funzionamento e la motivazione dietro questa scelta.

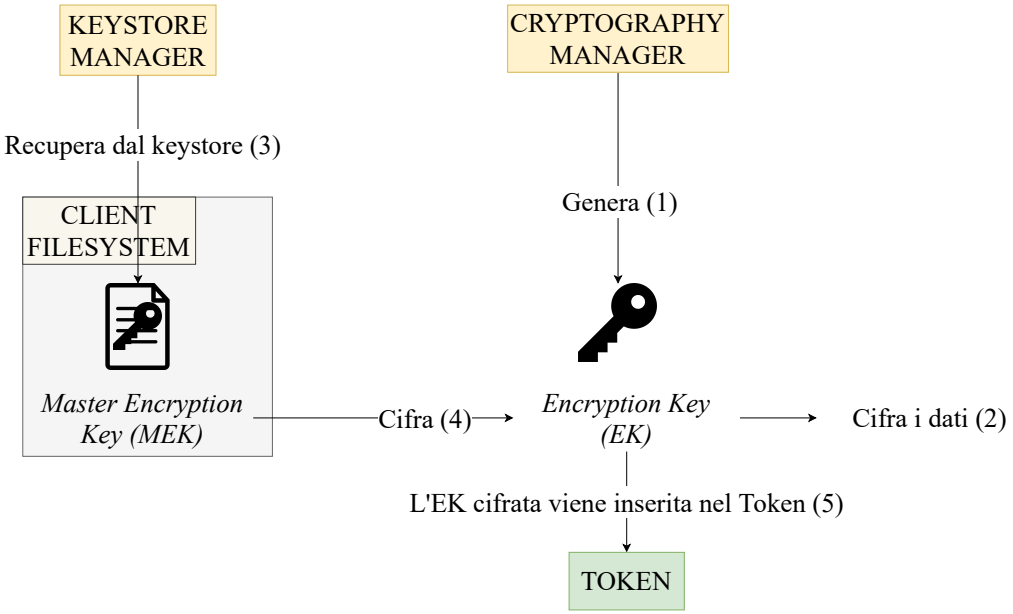


Figura 6.2: Key Management - Caso cifratura

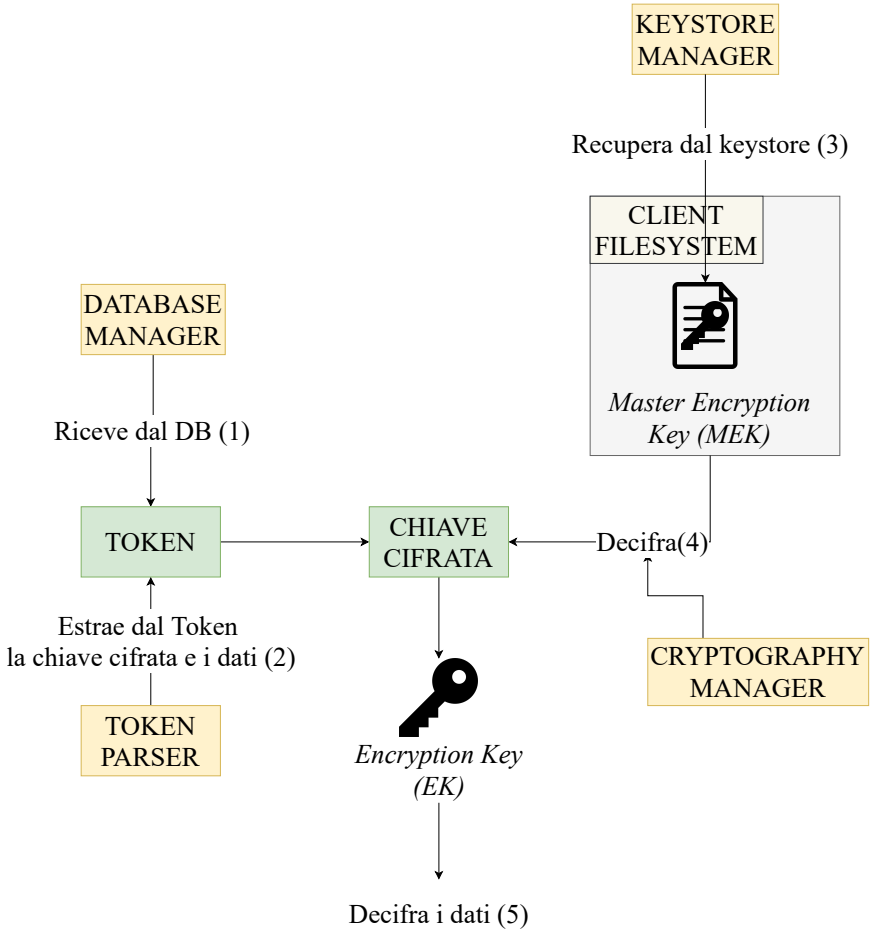


Figura 6.3: Key Management - Caso decifratura

3. Il KeystoreManager recupera dal keystore la MEK;
4. Il CryptographyManager utilizza la MEK per decifrare l'EK cifrata;

5. L'EK viene utilizzata per decifrare i dati.

Per tutte le informazioni aggiuntive consultare il Manuale Utente (Appendice A) e il Manuale Programmatore (Appendice B).

Capitolo 7

Analisi del sistema

In questo capitolo si mostrano i risultati sperimentali ottenuti testando il driver per la client-side encryption, sviluppato nel Capitolo 5, per quanto riguardo l'impatto sullo storage e sulle performance. Inoltre si effettua un'analisi critica del sistema sviluppato.

7.1 Client-side Benchmarking Framework

Nel Capitolo 5 è stato definito un processo di valutazione generale per la Database Encryption. È stato inoltre definito, per ciascuna soluzione (DBMS Engine e SQL Interface), un criterio di valutazione specifico.

Il Client-side Benchmarking Framework eredita il processo di valutazione generale (basato su metriche, algoritmo, dimensione della tabella, granularità dei dati e stato del sistema) e lo adatta al particolare caso della soluzione client-side sviluppata Capitolo 6.

Gli scenari possibili sono:

- Scenario 1: valutazione dell'impatto sullo storage per differenti: lunghezza delle chiavi, algoritmi di cifratura, dimensione della tabella e granularità;
- Scenario 2: valutazione delle performance per quanto riguarda l'Elapsed Time e CPU Time per differenti: lunghezza delle chiavi, algoritmi di cifratura, dimensione della tabella e granularità.

Per testare il sistema, sono stati scritti dei programmi Java che emulano delle possibili applicazioni client che utilizzano il driver sviluppato. Questi programmi sono consultabili al seguente URL `https://github.com/LorenzoCeccarelli/Tesi/tree/main/src/test/java/examples/performanceTest` [1].

Per effettuare le rilevazioni sul CPU Time ed Elapsed Time sono stati utilizzati i metodi *System.nanoTime()* [88] e *osMBean.getProcessCpuTime()* [89]. *nanoTime()* fornito dalla classe *System* [90] permette di ottenere il tempo attuale della Java Virtual Machine con precisione di nanosecondi ed è stato utilizzato per calcolare l'Elapsed Time. *getProcessCpuTime()* fornito dall'interfaccia *OperatingSystemMXBean* [91] permette di misurare il tempo per cui la CPU è stata dedicata per quel processo e perciò è stato utilizzato per calcolare il CPU Time.

[1] Nella Sezione 7.1.2, per ogni test viene inserito il riferimento al corrispondente programma che ha generato i risultati analizzati.

7.1.1 Storage overhead

Riguardo allo storage overhead occorre considerare il formato dei dati memorizzati sul database. Infatti, come mostrato nel Manuale Programmatore (Sezione B.2), il DB memorizza una stringa (un token) che è composta da due o tre campi separati da un '.' a seconda se è richiesta o meno la cifratura del dato originale. Nel caso in cui i dati originali richiedano cifratura, il token generato ha il seguente formato:

'CIPHERTEXT'.Base64('ciphertext').Base64('cipherkey')

Mentre per quanto i riguarda il caso in cui i dati originali *non* richiedano cifratura, il token generato ha il seguente formato:

'PLAINTEXT'.Base64('plaintext')

La Figura 7.1 mostra la dimensione dei database di prova al variare dell'algoritmo e della dimensione della tabella. Come si può notare, l'impatto sulla dimensione dello storage è significativa.

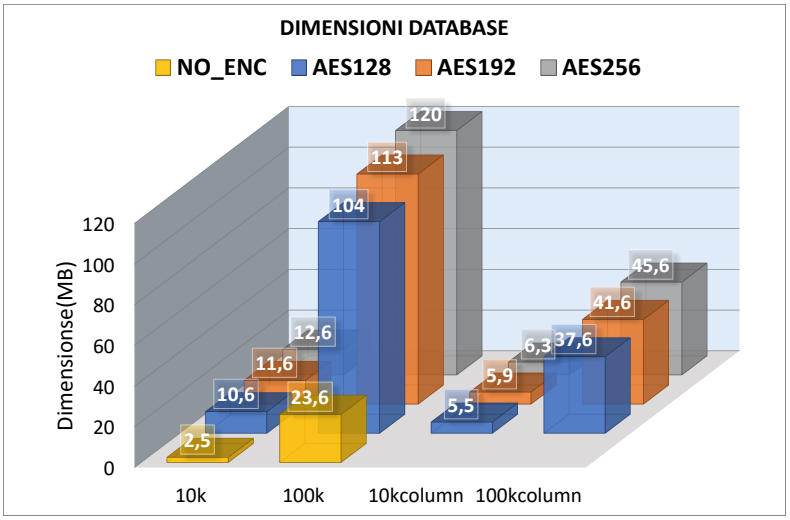


Figura 7.1: Dimensioni database

Dati cifrati

Nel caso dei dati cifrati, il token è composto da una parte fissa e da una parte variabile.

La parte fissa corrisponde a *'CIPHERTEXT'*, infatti è uguale in tutti gli EncryptedToken, ed ha una dimensione di 10 byte.

La parte variabile corrisponde a *'ciphertext'.Base64('cipherkey')* che dipende dalla lunghezza del dato originale e dalla lunghezza della chiave crittografica. La Tabella 7.1 mostra la dimensione della chiave cifrata codificata in Base64 al variare della lunghezza della chiave originale.

Algoritmo	Lunghezza chiave [byte]	Base64('cipherkey') [byte]
AES128	16	56
AES192	24	70
AES256	32	80

Tabella 7.1

La Tabella 7.2 mostra la dimensione dei dati cifrati in Base64 al variare della dimensione dei dati originali.

Tabella 7.2: Dimensione dati cifrati codificati in Base64.

Dimensione dati originali [byte]	Base64('ciphertext') [byte]
1	39
2	40
3	42
4	43
5	44
6	46
7	47
8	48
9	50
10	51
11	52
12	54
13	55
14	56
15	58
16	59
17	60
18	62
19	63
20	64
21	66
22	67
23	68
24	70
25	71
26	72
27	74
28	75
29	76
30	78
31	79
32	80
33	82
34	83
35	84
36	86
37	87
38	88
39	90
40	91
41	92
42	94
43	95
44	96
45	98
46	99
47	100
48	102
49	103
50	104
51	106
52	107

53	108
54	110
55	111
56	112
57	114
58	115
59	116
60	118
61	119
62	120
63	122
64	123
65	124
66	126
67	127
68	128
69	130
70	131
71	132
72	134
73	135
74	136
75	138
76	139
77	140
78	142
79	143
80	144
81	146
82	147
83	148
84	150
85	151
86	152
87	154
88	155
89	156
90	158
91	159
92	160
93	162
94	163
95	164
96	166
97	167
98	168
99	170
100	171
101	172
102	174
103	175
104	176
105	178
106	179
107	180

108	182
109	183
110	184
111	186
112	187
113	188
114	190
115	191
116	192
117	194
118	195
119	196
120	198
121	199
122	200
123	202
124	203
125	204
126	206
127	207
128	208
129	210
130	211
131	212
132	214
133	215
134	216
135	218
136	219
137	220
138	222
139	223
140	224
141	226
142	227
143	228
144	230
145	231
146	232
147	234
148	235
149	236
150	238
151	239
152	240
153	242
154	243
155	244
156	246
157	247
158	248
159	250
160	251
161	252
162	254

163	255
164	256
165	258
166	259
167	260
168	262
169	263
170	264
171	266
172	267
173	268
174	270
175	271
176	272
177	274
178	275
179	276
180	278
181	279
182	280
183	282
184	283
185	284
186	286
187	287
188	288
189	290
190	291
191	292
192	294
193	295
194	296
195	298
196	299
197	300
198	302
199	303
200	304
201	306
202	307
203	308
204	310
205	311
206	312
207	314
208	315
209	316
210	318
211	319
212	320
213	322
214	323
215	324
216	326
217	327

218	328
219	330
220	331
221	332
222	334
223	335
224	336
225	338
226	339
227	340
228	342
229	343
230	344
231	346
232	347
233	348
234	350
235	351
236	352
237	354
238	355
239	356
240	358
241	359
242	360
243	362
244	363
245	364
246	366
247	367
248	368
249	370
250	371
251	372
252	374
253	375
254	376
255	378

La dimensione totale del token è quindi data dalla somma delle tre parti più altri due byte che corrispondono ai due punti che separano i campi.

Lo storage overhead (in percentuale) è quindi dato dalla seguente formula:

$$\frac{dimensioneToken - dimensioneDatoOriginale}{dimensioneDatoOriginale} * 100 \tag{7.1}$$

La Tabella 7.3 mostra lo storage overhead al variare della lunghezza della chiave.

Tabella 7.3: Storage Overhead per i dati cifrati.

Dati originali[byte]	AES128 Overhead[%]	AES192 Overhead[%]	AES256 Overhead[%]
1	10900	12000	13000
2	5450	6000	6500
3	3667	4033	4367
4	2750	3025	3275

5	2200	2420	2620
6	1850	2033	2200
7	1586	1743	1886
8	1388	1525	1650
9	1244	1367	1478
10	1120	1230	1330
11	1018	1118	1209
12	942	1033	1117
13	869	954	1031
14	807	886	957
15	760	833	900
16	713	781	844
17	671	735	794
18	639	700	756
19	605	663	716
20	575	630	680
21	552	605	652
22	527	577	623
23	504	552	596
24	488	533	575
25	468	512	552
26	450	492	531
27	437	478	515
28	421	461	496
29	407	445	479
30	397	433	467
31	384	419	452
32	372	406	438
33	364	397	427
34	353	385	415
35	343	374	403
36	336	367	394
37	327	357	384
38	318	347	374
39	313	341	367
40	305	333	358
41	298	324	349
42	293	319	343
43	286	312	335
44	280	305	327
45	276	300	322
46	270	293	315
47	264	287	309
48	260	283	304
49	255	278	298
50	250	272	292
51	247	269	288
52	242	263	283
53	238	258	277
54	235	256	274
55	231	251	269
56	227	246	264
57	225	244	261
58	221	240	257
59	217	236	253

60	215	233	250
61	211	230	246
62	208	226	242
63	206	224	240
64	203	220	236
65	200	217	232
66	198	215	230
67	196	212	227
68	193	209	224
69	191	207	222
70	189	204	219
71	186	201	215
72	185	200	214
73	182	197	211
74	180	195	208
75	179	193	207
76	176	191	204
77	174	188	201
78	173	187	200
79	171	185	197
80	169	183	195
81	168	181	194
82	166	179	191
83	164	177	189
84	163	176	188
85	161	174	186
86	159	172	184
87	159	171	183
88	157	169	181
89	155	167	179
90	154	167	178
91	153	165	176
92	151	163	174
93	151	162	173
94	149	161	171
95	147	159	169
96	147	158	169
97	145	157	167
98	144	155	165
99	143	155	165
100	142	153	163
101	141	151	161
102	140	151	161
103	139	150	159
104	138	148	158
105	137	148	157
106	136	146	156
107	135	145	154
108	134	144	154
109	133	143	152
110	132	142	151
111	132	141	150
112	130	140	149
113	129	139	148
114	129	139	147

115	128	137	146
116	127	136	145
117	126	136	144
118	125	135	143
119	124	134	142
120	124	133	142
121	123	132	140
122	122	131	139
123	122	131	139
124	121	130	138
125	120	129	137
126	120	129	137
127	119	128	135
128	118	127	134
129	118	126	134
130	117	125	133
131	116	124	132
132	116	124	132
133	115	123	131
134	114	122	130
135	114	122	130
136	113	121	129
137	112	120	128
138	112	120	128
139	112	119	127
140	111	119	126
141	111	118	126
142	110	118	125
143	109	117	124
144	109	117	124
145	108	116	123
146	108	115	122
147	107	115	122
148	107	114	121
149	106	113	120
150	106	113	120
151	105	113	119
152	105	112	118
153	105	112	118
154	104	111	118
155	103	110	117
156	103	110	117
157	103	110	116
158	102	109	115
159	102	109	115
160	101	108	114
161	101	107	114
162	101	107	114
163	100	107	113
164	99	106	112
165	99	106	112
166	99	105	111
167	98	105	111
168	98	105	111
169	98	104	110

170	97	104	109
171	97	104	109
172	97	103	109
173	96	102	108
174	96	102	108
175	95	102	107
176	95	101	107
177	95	101	107
178	94	101	106
179	94	100	106
180	94	100	106
181	93	99	105
182	93	99	104
183	93	99	104
184	92	98	104
185	92	98	103
186	92	98	103
187	91	97	103
188	91	97	102
189	91	97	102
190	91	96	102
191	90	96	101
192	90	96	101
193	90	95	101
194	89	95	100
195	89	95	100
196	89	94	99
197	88	94	99
198	88	94	99
199	88	93	98
200	88	93	98
201	88	93	98
202	87	93	98
203	87	92	97
204	87	92	97
205	86	92	97
206	86	91	96
207	86	91	96
208	86	91	96
209	85	90	95
210	85	90	95
211	85	90	95
212	84	90	94
213	85	90	94
214	84	89	94
215	84	89	93
216	84	89	94
217	83	88	93
218	83	88	93
219	83	88	93
220	83	88	92
221	82	87	92
222	82	87	92
223	82	87	91
224	82	87	91

225	82	87	91
226	81	86	91
227	81	86	90
228	81	86	90
229	81	86	90
230	80	85	90
231	81	85	90
232	80	85	89
233	80	85	89
234	80	85	89
235	80	84	89
236	79	84	88
237	79	84	88
238	79	84	88
239	79	83	87
240	79	83	88
241	78	83	87
242	78	83	87
243	78	83	87
244	78	82	86
245	78	82	86
246	78	82	86
247	77	82	86
248	77	81	85
249	77	82	86
250	77	81	85
251	76	81	85
252	77	81	85
253	76	81	85
254	76	80	84
255	76	80	84

Si può notare come l'overhead diminuisca all'aumentare della dimensione dei dati originali. Ovviamente AES256 ha il maggior impatto in quanto la lunghezza della chiave è maggiore rispetto alle altre situazioni analizzate.

Dati in chiaro

Nel caso dei dati in chiaro, la dimensione del token dipende esclusivamente dalla lunghezza dei dati iniziali in quanto la dimensione di *'PLAINTEXT'* è fissa ed è pari a 9 byte.

La Tabella 7.4 mostra lo storage overhead al variare della lunghezza dei dati inziale.

Tabella 7.4: Storage Overhead per i dati in chiaro.

Dimensione dati	Dimensione dati (Base64)	Dimensione token	Overhead (%)
1	2	12	1100
2	3	13	550
3	4	14	367
4	6	16	300
5	7	17	240
6	8	18	200
7	10	20	186
8	11	21	163
9	12	22	144

10	14	24	140
11	15	25	127
12	16	26	117
13	18	28	115
14	19	29	107
15	20	30	100
16	22	32	100
17	23	33	94
18	24	34	89
19	26	36	89
20	27	37	85
21	28	38	81
22	30	40	82
23	31	41	78
24	32	42	75
25	34	44	76
26	35	45	73
27	36	46	70
28	38	48	71
29	39	49	69
30	40	50	67
31	42	52	68
32	43	53	66
33	44	54	64
34	46	56	65
35	47	57	63
36	48	58	61
37	50	60	62
38	51	61	61
39	52	62	59
40	54	64	60
41	55	65	59
42	56	66	57
43	58	68	58
44	59	69	57
45	60	70	56
46	62	72	57
47	63	73	55
48	64	74	54
49	66	76	55
50	67	77	54
51	68	78	53
52	70	80	54
53	71	81	53
54	72	82	52
55	74	84	53
56	75	85	52
57	76	86	51
58	78	88	52
59	79	89	51
60	80	90	50
61	82	92	51
62	83	93	50
63	84	94	49
64	86	96	50

65	87	97	49
66	88	98	48
67	90	100	49
68	91	101	49
69	92	102	48
70	94	104	49
71	95	105	48
72	96	106	47
73	98	108	48
74	99	109	47
75	100	110	47
76	102	112	47
77	103	113	47
78	104	114	46
79	106	116	47
80	107	117	46
81	108	118	46
82	110	120	46
83	111	121	46
84	112	122	45
85	114	124	46
86	115	125	45
87	116	126	45
88	118	128	45
89	119	129	45
90	120	130	44
91	122	132	45
92	123	133	45
93	124	134	44
94	126	136	45
95	127	137	44
96	128	138	44
97	130	140	44
98	131	141	44
99	132	142	43
100	134	144	44
101	135	145	44
102	136	146	43
103	138	148	44
104	139	149	43
105	140	150	43
106	142	152	43
107	143	153	43
108	144	154	43
109	146	156	43
110	147	157	43
111	148	158	42
112	150	160	43
113	151	161	42
114	152	162	42
115	154	164	43
116	155	165	42
117	156	166	42
118	158	168	42
119	159	169	42

120	160	170	42
121	162	172	42
122	163	173	42
123	164	174	41
124	166	176	42
125	167	177	42
126	168	178	41
127	170	180	42
128	171	181	41
129	172	182	41
130	174	184	42
131	175	185	41
132	176	186	41
133	178	188	41
134	179	189	41
135	180	190	41
136	182	192	41
137	183	193	41
138	184	194	41
139	186	196	41
140	187	197	41
141	188	198	40
142	190	200	41
143	191	201	41
144	192	202	40
145	194	204	41
146	195	205	40
147	196	206	40
148	198	208	41
149	199	209	40
150	200	210	40
151	202	212	40
152	203	213	40
153	204	214	40
154	206	216	40
155	207	217	40
156	208	218	40
157	210	220	40
158	211	221	40
159	212	222	40
160	214	224	40
161	215	225	40
162	216	226	40
163	218	228	40
164	219	229	40
165	220	230	39
166	222	232	40
167	223	233	40
168	224	234	39
169	226	236	40
170	227	237	39
171	228	238	39
172	230	240	40
173	231	241	39
174	232	242	39

175	234	244	39
176	235	245	39
177	236	246	39
178	238	248	39
179	239	249	39
180	240	250	39
181	242	252	39
182	243	253	39
183	244	254	39
184	246	256	39
185	247	257	39
186	248	258	39
187	250	260	39
188	251	261	39
189	252	262	39
190	254	264	39
191	255	265	39
192	256	266	39
193	258	268	39
194	259	269	39
195	260	270	38
196	262	272	39
197	263	273	39
198	264	274	38
199	266	276	39
200	267	277	39
201	268	278	38
202	270	280	39
203	271	281	38
204	272	282	38
205	274	284	39
206	275	285	38
207	276	286	38
208	278	288	38
209	279	289	38
210	280	290	38
211	282	292	38
212	283	293	38
213	284	294	38
214	286	296	38
215	287	297	38
216	288	298	38
217	290	300	38
218	291	301	38
219	292	302	38
220	294	304	38
221	295	305	38
222	296	306	38
223	298	308	38
224	299	309	38
225	300	310	38
226	302	312	38
227	303	313	38
228	304	314	38
229	306	316	38

230	307	317	38
231	308	318	38
232	310	320	38
233	311	321	38
234	312	322	38
235	314	324	38
236	315	325	38
237	316	326	38
238	318	328	38
239	319	329	38
240	320	330	38
241	322	332	38
242	323	333	38
243	324	334	37
244	326	336	38
245	327	337	38
246	328	338	37
247	330	340	38
248	331	341	38
249	332	342	37
250	334	344	38
251	335	345	37
252	336	346	37
253	338	348	38
254	339	349	37
255	340	350	37

7.1.2 Performance overhead

I seguenti test sono stati eseguiti su tabelle da 10 mila e 100 mila tuple. Le operazioni di INSERT e UPDATE sono state effettuate utilizzando i metodi *addToBatch()* e *executeBatch()* per aumentare le performance.

INSERT

La Figura 7.2 mostra l'Elapsed Time e il CPU Time riguardo l'operazione di INSERT. Come è possibile notare l'impatto è simile tra i vari algoritmi mentre c'è un overhead considerevole con il caso in cui la cifratura non è abilitata. La Tabella 7.5 riassume gli incrementi percentuali riguardo all'Elapsed Time mentre la Tabella 7.6 per quanto riguarda il CPU Time.

I risultati sono stati ottenuti utilizzando i programmi `https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/test/java/examples/performanceTest/InsertNoEncTableTest.java` (per il caso senza cifratura), `https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/test/java/examples/performanceTest/InsertTotalCipherTableTest.java` (per il caso della cifratura ad intera tabella) e `https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/test/java/examples/performanceTest/InsertSingleCipherColumnTest.java` (per il caso della cifratura a singola colonna). La Tabella D.33 (Appendice D) mostra lo stato del sistema prima, durante e dopo l'esecuzione dei test.

L'impatto è considerevole in quanto per ciascun dato sono coinvolte tre operazioni crittografiche quali: generazione della chiave di cifratura, cifratura dei dati e cifratura della chiave (con la Master Encryption Key). L'esecuzione di query multiple utilizzando i metodi *addToBatch()* e *executeBatch()* ha permesso di ridurre la latenza rispetto all'esecuzione singola per ciascuna query, come descritto nel Manuale Programmatore (Appendice B).

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	+58%	-2%	+75%	+10%
AES192	+42%	-2%	+70%	+4%
AES256	+50%	-1%	+73%	+9%

Tabella 7.5: Overhead INSERT (Elapsed Time)

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	+340%	+160%	+866%	+180%
AES192	+320%	+160%	+880%	+233%
AES256	+340%	+160%	+966%	+247%

Tabella 7.6: Overhead INSERT (CPU Time)

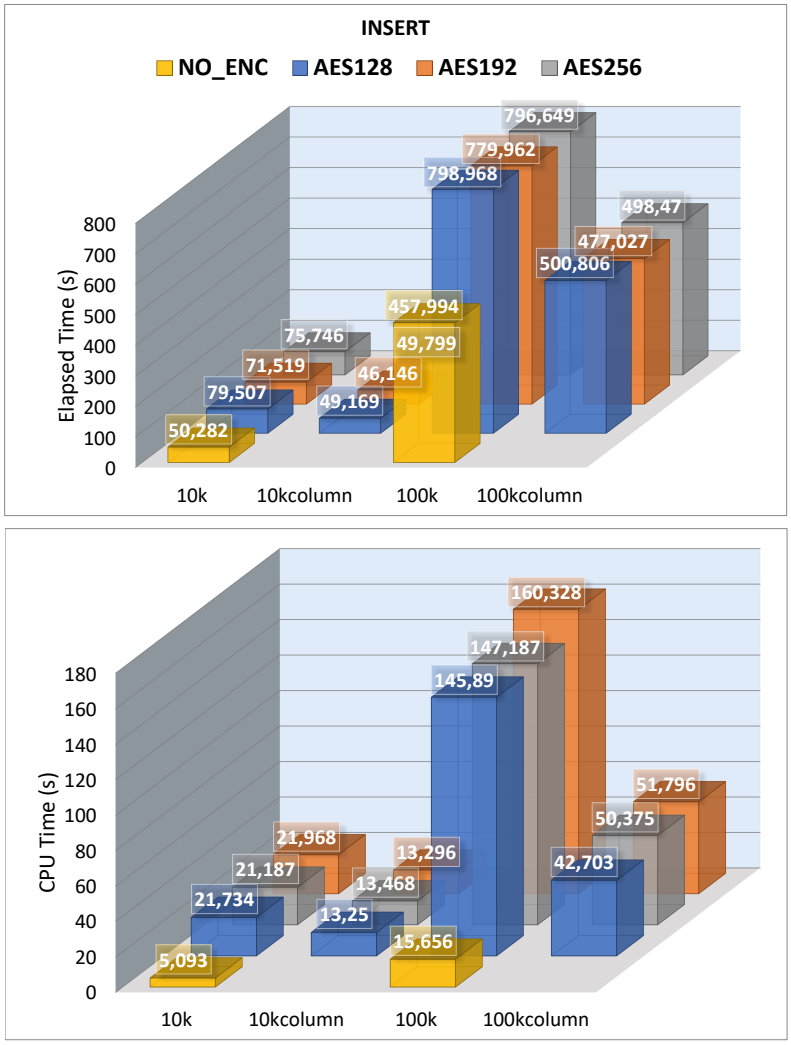


Figura 7.2: *Clientside Encryption* - Confronto dell'Elapsed Time e CPU Time(INSERT).

SELECT

La Figura 7.3 mostra l'Elapsed Time e il CPU Time riguardo l'operazione di SELECT. Le Tabelle 7.7 e 7.8 mostrano gli incrememnti percentuali rispetto alle operazioni sul database non cifrato, rispettivamente per l'Elapsed Time e per il CPU Time.

I risultati sono stati ottenuti utilizzando il programma al seguente URL https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/test/java/examples/performanceTest/SelectTest.java. La Tabella D.34 (Appendice D) mostra lo stato del sistema prima, durante e dopo l'esecuzione dei test.

L'operazione di SELECT coinvolge due operazioni crittografiche per ogni dato (token) ricevuto dal database, ovvero: decifratura della chiave di cifratura (con la Master Encryption Key) e la decifratura dei dati. Tali operazioni crittografiche causano un impatto negativo sulle performance rispetto al caso in cui la cifratura non è abilitata.

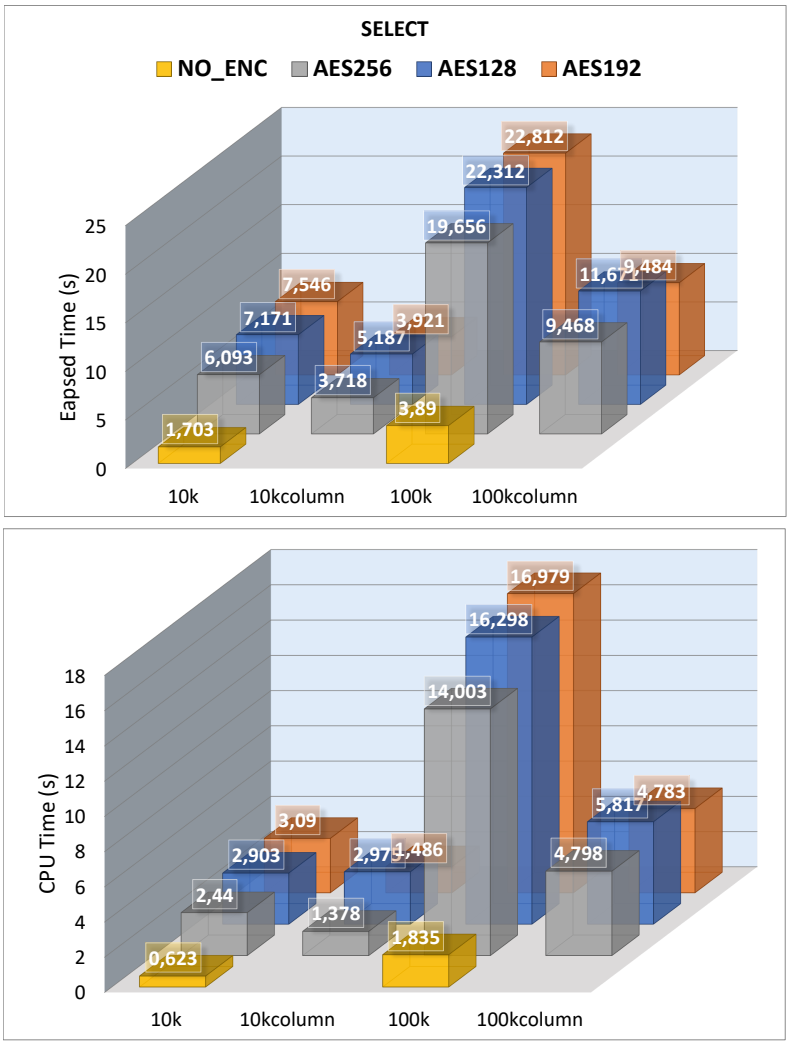


Figura 7.3: *Clientside Encryption* - Confronto dell'Elapsed Time e CPU Time (SELECT)

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	+321%	+204%	+473%	+200%
AES192	+343%	+130%	+486%	+143%
AES256	+257%	+118%	+405%	+143%

Tabella 7.7: Overhead SELECT (Elapsed Time)

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	+365%	+377%	+788%	+217%
AES192	+395%	+128%	+825%	+160%
AES256	+291%	+121%	+663%	+161%

Tabella 7.8: Overhead SELECT (CPU Time)

UPDATE

La Figura 7.4 mostra l'Elapsed Time e il CPU Time riguardo l'operazione di UPDATE. Coinvolge le stesse operazioni crittografiche della SELECT. Le Tabelle 7.9 e 7.10 mostrano l'incremento percentuale rispetto alle operazioni sul database non cifrato, rispettivamente per l'Elapsed Time e per il CPU Time.

I risultati sono stati ottenuti eseguendo il programma al seguente URL `https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/test/java/examples/performanceTest/UpdateCipherTableTest.java`. La Tabella D.35 (Appendice D) mostra lo stato del sistema prima, durante e dopo l'esecuzione dei test.

Come si può notare dai grafici, nonostante ci sia un notevole incremento dell'Elpased Time, il CPU Time rimane circa costante e il suo andamento non segue un criterio specifico. Ciò è dimostrato dal fatto che, ispezionando il codice del programma, viene effettuata una sola operazione crittografica e viene inviata una sola query al DBMS. Ciò comporta un aumento minimo del CPU Time (comunque non significativo). Essendo la chiamata ai driver JDBC sincrona (quindi il programma si mette in attesa della risposta del DBMS), le latenze sono alte, ciò giustifica l'elevato Elapsed Time. Detto in altri termini, il programma aspetta che il DBMS completi l'operazione di UPDATE per tutte le tuple coinvolte.

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	+71%	+15%	+132%	+14%
AES192	+93%	+1%	+146%	+34%
AES256	+115%	+11%	+120%	+26%

Tabella 7.9: Overhead UPDATE (Elapsed Time)

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	+17%	+21%	+61%	+61%
AES192	-32%	+28%	+38%	+77%
AES256	+0%	-35%	+11%	+105%

Tabella 7.10: Overhead UPDATE (CPU Time)

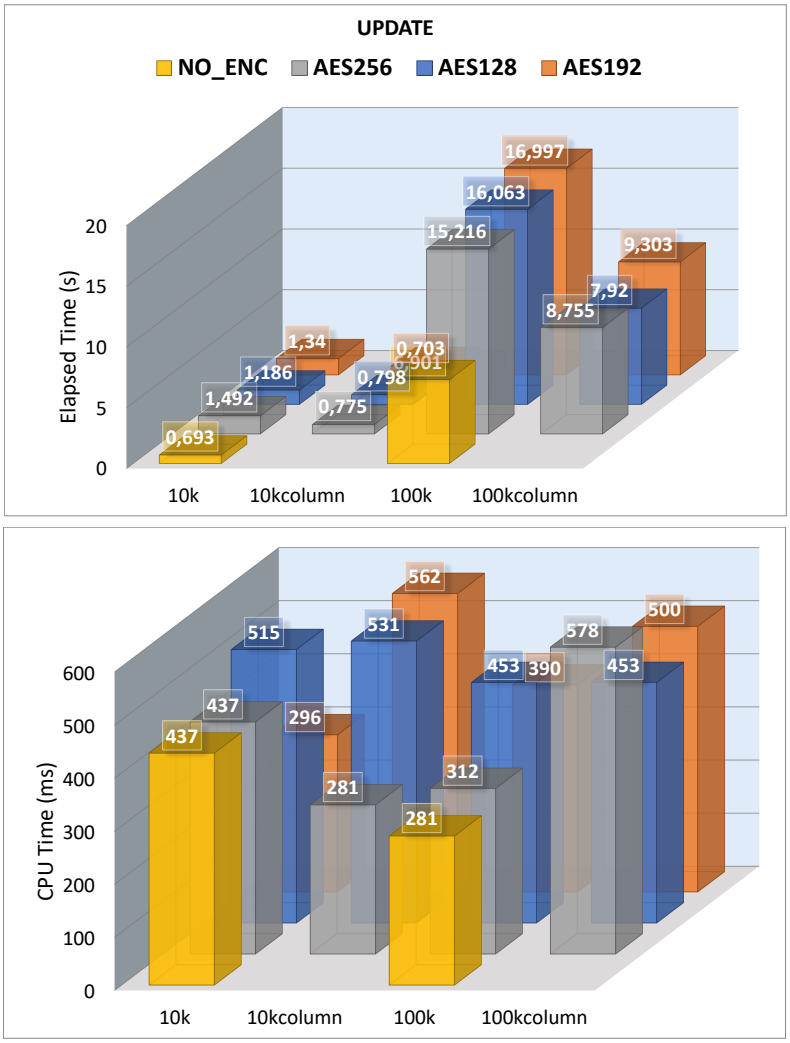


Figura 7.4: *Clientside Encryption* - Confronto dell'Elapsed Time e CPU Time (UPDATE)

DELETE

La Figura 7.5 mostra l'Elapsed Time e il CPU Time riguardo l'operazione di DELETE. Questa operazione non coinvolge nessuna operazione crittografica. Però, si può notare un impatto considerevole sulle performance, dovuto alla maggiore dimensione del database (rispetto a quello non cifrato) e quindi un maggior numero di dati da eliminare. Le Tabelle 7.11 e 7.12 mostrano l'incremento percentuale rispetto al database non cifrato, rispettivamente per l'Elapsed Time e CPU Time.

I risultati sono stati ottenuti eseguendo il programma al seguente URL `https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/test/java/examples/performanceTest/DeleteTest.java`. La Tabella D.36 (Appendice D) mostra lo stato del sistema prima, durante e dopo l'esecuzione dei test.

Come nel caso dell'UPDATE, anche in questo caso si nota un aumento significativo dell'Elapsed Time ma l'Elapsed Time non segue un andamento logico. Il motivo è che il programma invia una semplice DELETE al DBMS senza effettuare operazioni crittografiche perciò l'impatto sul CPU Time è nullo. L'elevata latenza è dovuta al fatto che la comunicazione con il database è sincrona, perciò il programma aspetta che il DBMS finisca l'operazione di DELETE.

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	+123%	-36%	+263%	+20%
AES192	+134%	-23%	+271%	+17%
AES256	+107%	-23%	+220%	+22%

Tabella 7.11: Overhead Delete (Elapsed Time)

Algoritmo	10k	10k colonna	100k	100k colonna
AES128	-41%	-41%	+136%	+204%
AES192	-20%	-41%	+171%	+69%
AES256	+60%	+19%	+102%	+171%

Tabella 7.12: Overhead Delete (CPU Time)

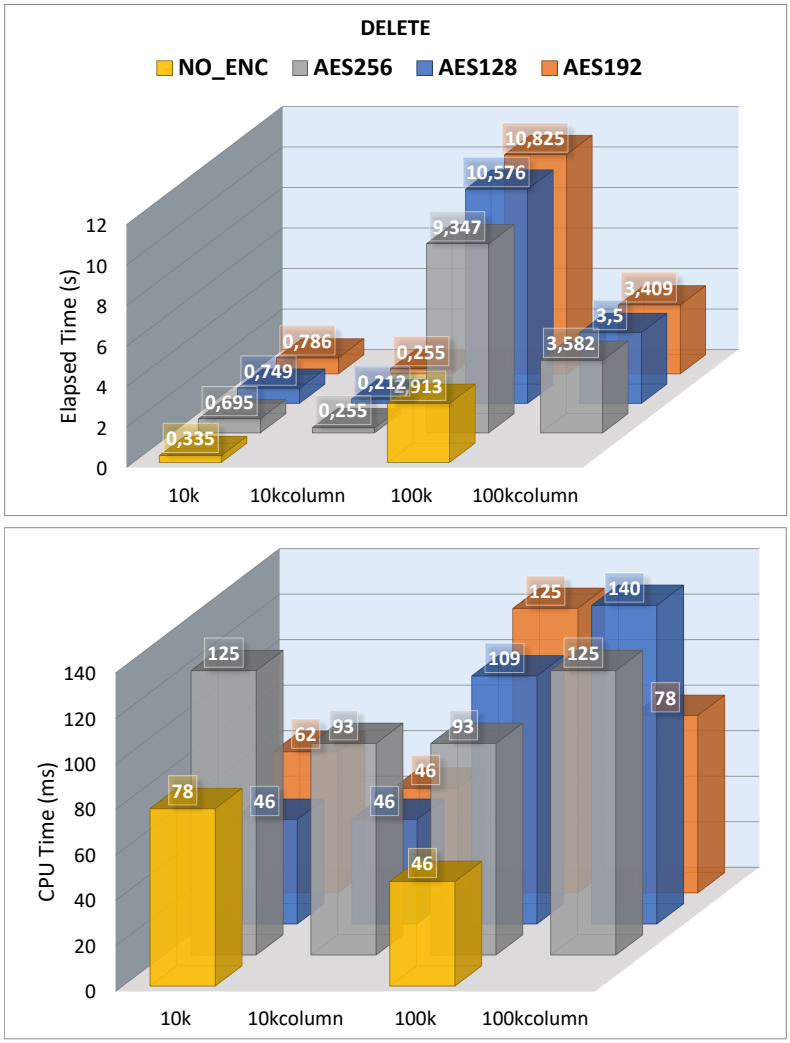


Figura 7.5: *Clientside Encryption* - Confronto dell'Elapsed Time e CPU Time (DELETE)

7.2 Analisi del sistema

Problemi e possibili estensioni

Dal punto di vista della sicurezza, il software sviluppato garantisce la confidenzialità dei dati nel caso in cui venga richiesta la cifratura. Infatti i campi dell'*EncryptedToken* (dati e chiave) sono cifrati utilizzando un algoritmo sicuro, AES in modalità GCM, che garantisce anche la proprietà di integrità. Nel *ClearToken*, invece, i dati vengono solamente codificati in Base64 e quando vengono ritornati dal database non si effettua nessun integrity check. Di fatto, ciò non rappresenta un prerequisito fondamentale ma è un aspetto da tenere in considerazione. Rappresenta una problematica comune di quando si memorizzano i dati in un database remoto e il più delle volte viene gestito internamente dal DBMS. Una eventuale estensione del software potrebbe essere quella di permettere l'impostazione, opzionale, di un meccanismo di controllo dell'integrità basato, ad esempio, su MAC o firma digitale, quest'ultimo garantendo anche la proprietà di *non ripudio*.

Il driver sviluppato accetta come parametri solo stringhe. Si può pensare di aggiungere il supporto anche ad altri tipi di dato sia in fase di cifratura e decifratura.

Il Capitolo 8.1 contiene il confronto tra tutti i dati sperimentali ottenuti nel corso della trattazione.

Scenari di utilizzo

Il software sviluppato può essere integrato in tutti i processi che necessitino di un livello di sicurezza aggiuntivo sfruttando i vantaggi della clientside encryption a discapito di performance e dimensione dello storage. Ad esempio un'applicazione che deve gestire dati che richiedano elevata segretezza (es. password conto corrente) può richiamare le classi offerte dal *ClientsideEncryption package* per abilitare la cifratura dei dati in modo che nessuno, al di fuori dell'applicazione stessa, possa ottenere i dati originali. Infatti un possibile attaccante deve prima riuscire ad ottenere l'accesso al database e successivamente l'accesso alla macchina che ospita l'applicazione client. Nel caso di utilizzo di un HSM deve, inoltre, violare l'HSM stesso per ottenere le chiavi di cifratura.

Capitolo 8

Considerazioni finali

In questo ultimo capitolo si effettua una visione d'insieme sui risultati sperimentali ottenuti nei precedenti capitoli. Si procede poi con uno studio sull'evoluzione della Database Encryption per poi concludere la trattazione.

8.1 Confronto dei dati sperimentali

Il Capitolo 5 riporta i risultati sperimentali ottenuti testando le soluzioni di cifratura a livello DBMS Engine e SQL Interface mentre il Capitolo 7 riporta i risultati sperimentali ottenuti testando il driver sviluppato per l'abilitazione della cifratura lato client.

Riguardo allo **storage overhead**, la miglior soluzione è la *Transparent Data Encryption*. Infatti l'impatto sullo storage è nullo tranne che per il database in cui la cifratura è abilitata a livello colonna (+60%). Infatti i vendor dei prodotti analizzati, sia proprietari che open-source, hanno sviluppato la propria TDE per limitare l'overhead sullo storage. La cifratura a livello SQL Interface rappresenta un buon compromesso dove l'impatto è maggiore della Transparent Data Encryption ma comunque contenuto, tranne che per PostgreSQL dove l'aumento della dimensione del database è addirittura maggiore di quello della soluzione clientside (circa il 10x). Richiamando le funzioni offerte a linguaggio SQL si perdono tutte le ottimizzazioni della TDE. Infine, la clientside encryption rappresenta la soluzione peggiore. Memorizzando sul database un oggetto di tipo Token che è composto da campi aggiuntivi oltre al dato originale e considerando inoltre l'overhead aggiuntivo della codifica Base64 (+30-40 %) si raggiunge un impatto di circa il 500%.

Anche nel caso della perdita di **performance** la migliore soluzione è la *Transparent Data Encryption*. Infatti, i vari prodotti hanno un overhead che va da pochi punti percentuali fino al 30-40%, percentuale che è accettabile nel caso della cifratura. Ciò è giustificato dal fatto che i processi crittografici vengono eseguiti nel DBMS Engine prima di memorizzare la pagina sul disco permettendo una forte ottimizzazione. Nel caso della clientside encryption l'impatto è considerevole soprattutto nel caso della INSERT. Il driver, per ogni dato, deve effettuare due operazioni crittografiche più la codifica in Base64 e poi inviare il token (che ha dimensioni maggiori rispetto al dato orginale, quindi il numero di byte inviati che devono essere gestiti dal db è maggiore). Le soluzioni a livello SQL Interface hanno un impatto enorme (in alcuni casi fino a 100x !) e per questa ragione rappresentano la tecnica peggiore.

Dalle Tabelle dell'Appendice D si può effettuare un'ulteriore osservazione. Durante l'esecuzione dei test, le soluzioni TDE e SQL Interface non hanno influito sull'utilizzo della RAM [1] ma solo sull'utilizzo del disco. Infatti i dati venivano direttamente scritti sul disco senza essere memorizzati temporaneamente in RAM. Invece, per la soluzione di cifratura lato client sviluppata, durante l'esecuzione dei test si è notato un aumento della RAM utilizzata. Ciò è giustificato dal

[1] L'occupazione della RAM è rimasta costante per tutta la durata del test.

fatto che durante le operazioni crittografiche, i dati venivano temporaneamente memorizzati in memoria centrale per poi essere inviati in batch al database. Inoltre si è osservato che il processo non impattava sull'utilizzo del disco (0% in tutti i casi), poiché durante la clientside encryption non veniva scritto nulla sul disco (a differenza delle altre soluzioni TDE e SQL Interface). Solo il processo del DBMS Engine utilizzava il disco.

Da questi risultati, in aggiunta alle considerazioni teoriche fatte nel Capitolo 2, si deduce che le soluzioni di cifratura a livello SQL Interface sono le peggiori in quanto non scalano e non offrono un livello di sicurezza adeguato.

La clientside encryption rappresenta un buon compromesso. Infatti è la soluzione che offre il maggior livello di sicurezza a fronte di una perdita considerevole, ma accettabile, di performance. Richiede inoltre maggiore sforzo in quanto occorre modificare le applicazioni in modo da supportare la cifratura a livello client. Se si possiedono le risorse necessarie per effettuare questa migrazione e se l'applicazione può tollerare elevate latenze nelle operazioni di INSERT, la clientside encryption rappresenta la miglior soluzione.

La Transparent Data Encryption permette di raggiungere un buon livello di sicurezza e protezione a bassissimo costo. L'impatto sulle performance minimo, dimensione del database invariata e completa trasparenza nel suo utilizzo rendono la TDE la soluzione migliore in assoluto. Inoltre richiede un minimo sforzo nella configurazione che non è in ogni caso paragonabile con lo sviluppo di una soluzione clientside.

8.2 Conclusioni

Questo lavoro di tesi ha presentato uno studio sullo stato dell'arte della Database Encryption mostrando a quale livello e granularità possa essere effettuata. Per ciascuno livello è stata fornita una descrizione teorica del funzionamento interno mettendo in risalto vantaggi e svantaggi nell'utilizzo di quella particolare soluzione. La problematica della gestione delle chiavi è apparsa, fin dall'inizio della trattazione, come un punto cruciale nel corretto utilizzo della crittografia.

Per questo motivo è stato approfondito tutto il processo della *Key Management* partendo da una descrizione generale sul funzionamento e sul ciclo di vita delle chiavi. Si è sottolineato inoltre come il fattore umano sia fondamentale nella corretta gestione di un sistema crittografico. Infatti se tutta la conoscenza fosse nelle mani di una singola entità si avrebbe un'enorme falla nella sicurezza e a tale ragione si sono elencate delle buone pratiche da seguire per evitare questo tipo di vulnerabilità. Infine si sono descritte le più diffuse piattaforme per la memorizzazione delle chiavi e i più diffusi protocolli per la gestione delle chiavi crittografiche. Lo studio ha confermato come l'HSM sia il modulo che offre il maggior livello di sicurezza.

Dopo aver analizzato dal punto di vista teorico le varie soluzioni si è passati ad una descrizione delle funzionalità dei prodotti esistenti, forniti dai principali vendor (proprietari e open-source), che implementano tecniche di cifratura. Successivamente, ciascun prodotto è stato sottoposto a testing in modo da comprendere la reale perdita di performance e aumento della dimensione del database qualora la cifratura fosse abilitata e con quale granularità. A questo scopo si è costruito uno schema di valutazione, il *Benchmarking Framework*, che potesse essere più critico e omogeneo possibile. I risultati di questi test hanno portato la Transparent Data Encryption ad essere la tecnica più performante anche in comparazione con il livello di sicurezza offerto.

Come obiettivo di tesi si è voluto aggiungere un risvolto pratico alla fase di studio teorico della Database Encryption andando a sviluppare una proof-of-concept di una soluzione di clientside encryption. Perciò è stata proposta una possibile implementazione di un software in grado di essere facilmente configurabile da un'applicazione client per l'abilitazione della cifratura lato client. Durante la fase di implementazione si ha avuto la possibilità di approfondire i framework crittografici offerti da Java oltre ai driver JDBC per la comunicazione con il database. Successivamente il sistema è stato testato con i criteri definiti nello schema di valutazione in modo da permettere un confronto con le soluzioni di cifratura a livello DBMS Engine e SQL Interface. Ci si può ritenere soddisfatti del risultato ottenuto in quanto il sistema sviluppato offre il più alto livello di sicurezza con una perdita accettabile di performance, seppur maggiori della Transparent Data Encryption.

8.3 Possibili sviluppi futuri

Una delle problematiche della Database Encryption, ampiamente discussa in questa trattazione, è l'impatto sulle performance. L'utilizzo degli indici, che diminuiscono le latenze, è fortemente limitato in questo ambito. Uno studio su come permettere l'abilitazione degli indici è sicuramente interessante.

La *Crittografia omomorfa*, accennata nel Capitolo 2, può permettere di aumentare la velocità delle operazioni su database cifrati aumentandone anche la sicurezza. Permette infatti di eseguire operazioni su dati cifrati senza che questi vengano decifrati. Ad oggi, però, questo tipo di crittografia non è ancora utilizzata in quanto le operazioni sono estremamente lente. In futuro, con l'aumento ipotetico delle prestazioni dei processori, potrebbe trovare maggior campo di applicazione.

Una problematica riscontrata nello sviluppo del driver per l'abilitazione della cifratura lato client è stata quella del formato dei dati da memorizzare sul database. Il sistema, che supporta solo stringhe, deve prima trasformare la stringa in formato binario, effettuare la cifratura ed infine riconvertire in stringa usando la codifica Base64 che aggiunge un overhead considerevole alla dimensione del dato. La *Format Preserving Encryption* permette di cifrare un dato senza alterare il proprio formato [92].

Ulteriori studi su come applicare e sviluppare questi due concetti riguardo alla Database Encryption possono sicuramente apportare benefici alla comunità dei database.

Appendice A

Manuale Utente

Questo capitolo contiene il *Manuale Utente* che fornisce tutte le informazioni necessarie per installare ed utilizzare il driver software sviluppato per l'abilitazione della cifratura lato client.

A.1 Preparazione Ambiente

Per poter utilizzare il driver occorre scaricare la versione **11** del *Java Development Kit (JDK)* al seguente URL `https://www.oracle.com/it/java/technologies/javase-jdk11-downloads.html`. JDK è un ambiente di sviluppo che permette di costruire applicazioni e librerie per il linguaggio di programmazione Java. Il JDK contiene anche il *Java Runtime Environment (JRE)* e il compilatore (*javac*). Quindi è in grado di creare e compilare programmi. Il JRE, invece, è un pacchetto che contiene solo il necessario per eseguire un programma Java compilato utilizzando la Java Virtual Machine (JVM). Perciò, non può essere utilizzato per creare nuovi programmi. Se si vuole solamente testare il software sviluppato ci si può limitare a scaricare solo il JRE versione 11. Se invece si vuole modificare o aggiungere nuove funzionalità, occorre scaricare il JDK versione 11.

A.2 Importazione del software

Dopo aver preparato l'ambiente occorre importare il software sviluppato. Dato che il codice non è situato in nessun repository online (es. Maven Central) ci sono solo due opzioni per l'importazione del driver:

- Scaricando il file JAR [93] dal seguente URL `https://github.com/LorenzoCeccarelli/Tesi/blob/main/artifacts/CryptoDatabaseAdapter.jar`. Questo approccio è l'ideale se si vuole esclusivamente utilizzare il sistema, e quindi aggiungerlo come dipendenza al proprio progetto[1], in quanto il file JAR contiene già tutti i file compilati;
- Utilizzando *git* [94] e clonare tutto il repository sulla propria macchina. Il comando per fare ciò è `git clone https://github.com/LorenzoCeccarelli/Tesi.git`. Questo approccio, invece, è utile nel caso in cui si vuole andare a modificare il codice. Ovviamente occorre esplicitamente compilare i file.

Riguardo al primo caso, è possibile importare il file JAR scaricato nel progetto se si utilizza un ambiente di sviluppo integrato (IDE) in modo molto semplice. In questa sezione si mostra il

[1] Per progetto si intende l'applicazione client che necessita delle funzionalità offerte dal driver sviluppato.

processo di importazione in Intellij Idea (versione 2021.1.3) che è uno degli IDE professionali più utilizzati per quanto riguarda le applicazioni Java[2].

1. Aprire il progetto Intellij in cui si desidera utilizzare il driver per l'abilitazione della cifratura lato client;
2. Dal menù principale selezionare **File** → **Project Structure** → **Project Settings** → **Modules**;
3. Selezionare il modulo per il quale si vuole aggiungere il driver come dipendenza;
4. Cliccare su **Dependencies**;
5. Cliccare sul bottone + e poi selezionare **JARs or directories...**;
6. Infine cliccare sul file JAR corrispondente al driver.

La Figura A.1 mostra come inserire il driver all'interno del modulo Prova. Come si può notare il file JAR si trova al percorso *C:\Users\loren\Downloads*.

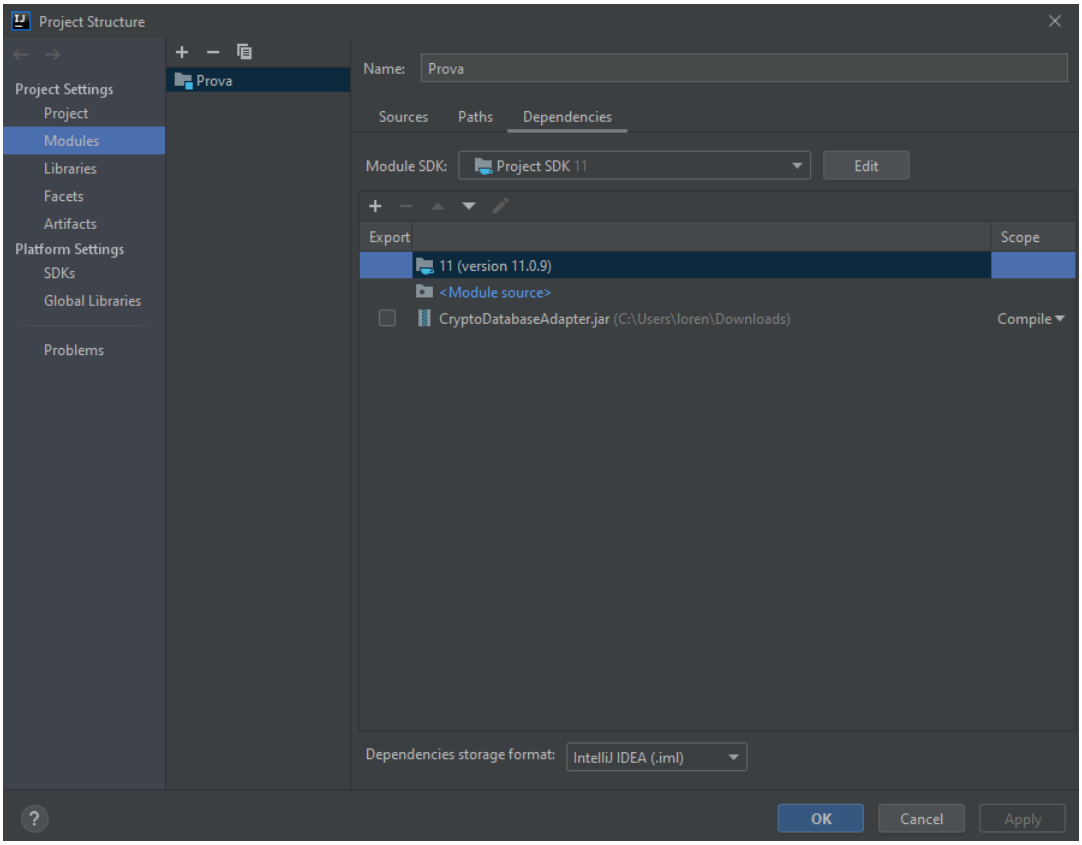


Figura A.1: Aggiunta delle dipendenze in Intellij.

A.3 Database supportati

Per quanto riguarda il database vengono usati i driver JDBC perciò è possibile connettere qualsiasi DB che supporta tale standard. In particolare, nella fase di sviluppo, è stato utilizzato MySQL. Per utilizzare un DBMS diverso occorre inserire la dipendenza nel progetto. Ad esempio, se si utilizza

[2]Di fatto la procedura è molto simile in tutti gli altri IDE (es. Eclipse).

Maven, occorre aggiungere nel file pom.xml la dipendenza corrispondente all'implementazione dei driver per quel determinato DBMS. Maven importerà la libreria specificata direttamente nel progetto. Ad esempio per i driver JDBC Microsoft occorre inserire nel file pom.xml all'interno del tag *dependencies*:

```
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <version>9.2.1.jre11</version>
</dependency>
```

Questo permette di scaricare la dipendenza da Maven Central [95].

Inoltre, i DBMS vendor forniscono nei loro siti web direttamente il file JAR che può essere importato all'interno del progetto come descritto nella Sezione A.2.

A.4 Utilizzo

Come discusso nel Capitolo 6 ci sono due possibili modalità di utilizzo del driver: **Transparente** e **Diretto**.

Il seguente programma mostra un possibile utilizzo del sistema in modalità Trasparente.

```
public class SimpleUsage {
public static void main(String[] args) {
    try {
        CryptoDatabaseAdapter cda = new CryptoDatabaseAdapter.Builder()
        .buildByFile("src/main/resources/config.properties");
        cda.init();
        boolean q = cda.newQueryBuilder("insert into crypto(id) values(?)")
                .setCipherParameter(1,"VerySecret",
                    CryptoUtils.Algorithm.AES128)
                .run();
        cda.newQueryBuilder("insert into crypto(id) values(?)")
                .setParameter(1,"NotSecret")
                .run();
        Set<Tuple> rs = cda.newQueryBuilder("select * from crypto")
                        .runSelect();
        rs.forEach(System.out::println);
    } catch (ClientsideEncryptionError error) {
        error.printStackTrace();}}}
```

Come si può notare occorre creare un oggetto di tipo CryptoDatabaseAdapter[3]. La configurazione di tale classe avviene all'interno della fase di creazione dell'oggetto. Successivamente occorre abilitare il driver e da questo momento è possibile interagire con esso.

La classe CryptoDatabaseAdapter mette a disposizione dei metodi per creare delle query da inviare al database dove i parametri sono, eventualmente, cifrati.

I risultati di una query di tipo SELECT possono essere immagazzinati in un Set di Tuple che può essere gestito dai metodi messi a disposizione da questo container standard.

Riguardo alla modalità Diretta, il seguente programma mostra un possibile utilizzo.

[3]Tutte le classi sviluppate vengono documentate all'interno del Manuale Programmatore (Appendice B).

```
public class ClientsideEncryptionExample {
public static void main(String[] args) {
    final String url = "jdbc:mysql://localhost:6789/tesi";
        final String username = "YourUsername";
        final String password = "YourPassword";

        DatabaseManager dbManager = new DatabaseManager(url, username,
            password);
        try {
                //Connect to database
                dbManager.connect();
                //Create or load a keystore (see KeystoreUsage in this folder)
                KeyStoreInfo ksi = KeystoreUtils.createKeystore("password");
                KeystoreUtils.saveKeystore(ksi,"prova.p12");

                //Create or load the master encryption key
                SecretKey sk =
                    CryptoUtils.createSymKey(CryptoUtils.Algorithm.AES256);
        //In case of creation, add the MEK to keystore and save it into the
            filesystem
                KeystoreUtils.insertKey(ksi,sk,"MEK");
                KeystoreUtils.saveKeystore(ksi,"prova.p12");
        //Create on-the-fly the encryption key
                SecretKey ek =
                    CryptoUtils.createSymKey(CryptoUtils.Algorithm.AES128);
        //Create the new query
                Query q = new Query("insert into crypto(id) values(?)"); //The
                    DB must contain the table called "crypto" otherwise an
                    error occurs
        //Encrypt the data with the encryption key
                String data = "VerySecret";
                System.out.println("Original data (before to send it to db) :
                    " + data);
                byte[] ciphertext = CryptoUtils.encryptDataWithPrefixIV(ek,
                    data.getBytes());
        //Encrypt the encryption key with the MEK
                byte[] cipherKey = CryptoUtils.encryptDataWithPrefixIV(sk,
                    ek.getEncoded());
        //Create the token (the db saves this token)
                EncryptedToken c = new EncryptedToken(cipherKey, ciphertext);
        //Set the token into the query
                q.setParameter(1, c.generateToken());
        //Run the query (INSERT modifies the DB)
                dbManager.runMutableQuery(q);
        //Retrieve tokens from the db
                Query q2 = new Query("select * from crypto");
                ResultSet rs = dbManager.runImmutableQuery(q2);
        //Retrieve the original data from the token
                while (rs.next()) {
            String a = rs.getString("id"); //The table crypto must have a
                column called "id"
            //Parse token
                        Token tk = TokenParser.parseToken(a);
                        //If token is an EncryptedToken
                        if(tk instanceof EncryptedToken) {
                                //Decrypt the key with MEK in order to retrieve
                                    the original key
```

```
                        byte[] key =
                            CryptoUtils.decryptDataWithPrefixIV(sk,
                            ((EncryptedToken) tk).getEncryptedKey());
                        //Reconstruct the original key from byte[] to
                            SecretKey object
                        SecretKey originalKey = new SecretKeySpec(key,
                            0, key.length, "AES");
                //Decrypt the original data and print it
                        System.out.println("Original data retrieved by
                            the db: " + new
                            String(CryptoUtils.decryptDataWithPrefixIV(
                            originalKey, ((EncryptedToken)
                            tk).getCiphertext())));
                    }
                    //If token is a ClearToken print the data
                    if(tk instanceof ClearToken){
                        System.out.println(((ClearToken) tk).getData());
                }
                }
            }catch (EncryptionError | DecryptionError | KeystoreOperationError |
                SQLException | ConnectionParameterNotValid |
                NoSuchAlgorithmException encryptionError) {
                        encryptionError.printStackTrace();
                }
        }
    }
```

Come si può notare, questo approccio comporta la scrittura di un maggior numero di linee di codice permettendo, però, una maggiore flessibilità.

Occorre innanzitutto creare una connessione con il database utilizzando la classe Database Manager.

Dopodiché bisogna creare o caricare un keystore utilizzando la classe KeystoreUtils.

Per iniziare il processo di cifratura occorre creare la Master Encryption Key (MEK) e l'Encryption Key (EK) utilizzando la classe CryptoUtils.

Inoltre, è possibile memorizzare la MEK all'interno del keystore.

Successivamente occorre creare una nuova query da inviare al database utilizzando la classe Query e impostare come parametro un oggetto di tipo Token.

Infine utilizzando il DatabaseManager si esegue la query.

L'ultimo WHILE permette di ottenere i dati originali decomponendo il token ricevuto dal database.

Per eseguire un programma Java si può utilizzare l'interfaccia grafica messa a disposizione da un IDE oppure utilizzando i comandi da terminale JAVAC, per compilare, e JAVA, per iniziare l'esecuzione del programma.

Appendice B

Manuale Programmatore

Questo capitolo contiene il *Manuale Programmatore* che fornisce la documentazione delle classi sviluppate e delle dipendenze necessarie al sistema per funzionare.

B.1 Dipendenze

Nel software sviluppato la maggior parte delle classi utilizzate sono classi standard di Java. In aggiunta è stata solo utilizzata la libreria **Commons lang 3** (versione 3.11) fornita da Apache Commons [96]. In particolare si è utilizzata la classe *ImmutablePair<L,R>* [97] in quanto permette di modellare in modo sofisticato un oggetto wrapper che incapsula una coppia di oggetti. È utile nel momento in cui un metodo deve ritornare due oggetti.

L'altra dipendenza riguarda l'implementazione dei driver JDBC di MySQL. Infatti, come spiegato nel Manuale Utente (A.3), la libreria sviluppata supporta solo i driver JDBC di MySQL in quanto è importata solo questa implementazione di questo DBMS vendor. Inoltre, sempre nel Manuale Utente, è spiegato come utilizzare il software con altri DBMS.

Il sistema è in grado di adattarsi molto facilmente al tipo di DBMS grazie al metodo *java.sql.DriverManager.getConnection(url,username,password)* [98] che in automatico utilizza la corretta implementazione dei driver JDBC analizzando l'URL del database passato come parametro. Come descritto nella documentazione (Sezione B.2), questa classe viene utilizzata nel package *Database* e nello specifico nella classe *DatabaseManager*.

B.2 Documentazione

B.2.1 Package *Core*

Il package ***Core*** rappresenta il nucleo del sistema ed è composto da diversi sotto-package per la gestione del database, keystore, crittografia e logging.

Package *Database*

Gestisce la comunicazione con il database. La Figura B.1 mostra il diagramma UML del package.

- **DatabaseManager class**. Si tratta di una classe dove il costruttore richiede tre stringhe: l'url del database, username e password per accedere al database. Utilizza l'interfaccia *java.sql.Connection* [99] per modellare una connessione con il database e l'interfaccia *java.sql.PreparedConnection* [100] per modellare uno statement SQL parametrico. Offre i seguenti metodi:

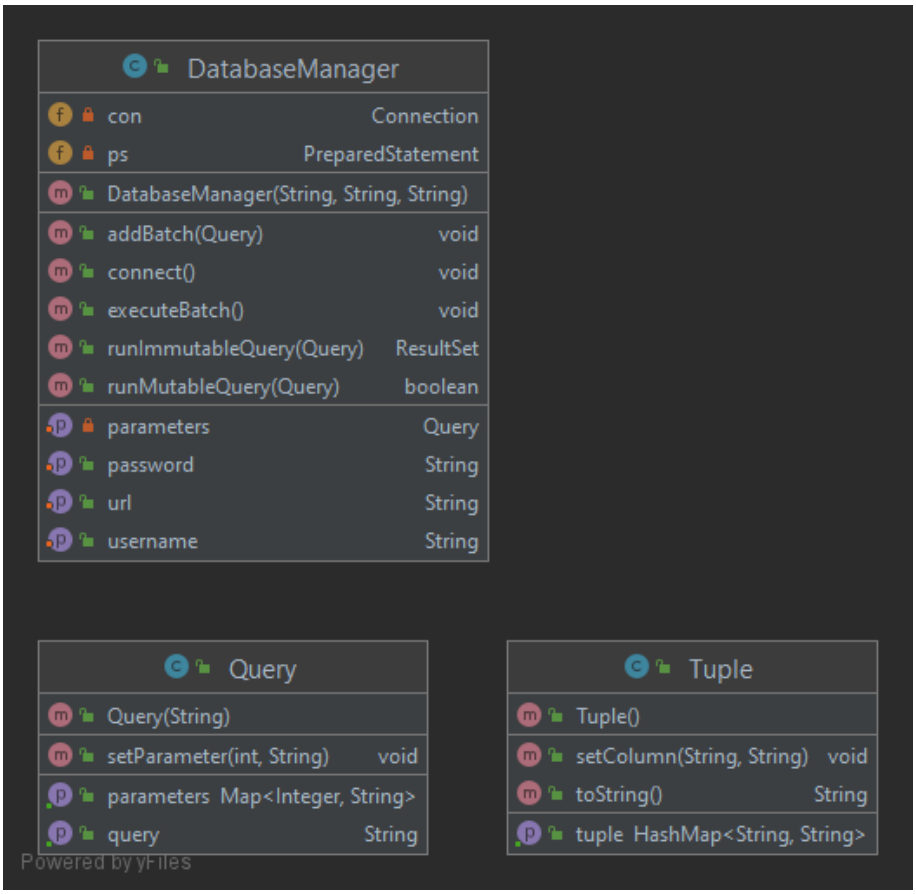


Figura B.1: Package Database - Diagramma UML

- *connect()*: permette la connessione al DB con i parametri specificati nel costruttore. Rilancia l'eccezione *ConnectionParameterNotValid*[1] se non riesce a stabilire una connessione con il database. Viene utilizzato il metodo *java.sql.DriverManager .getConnection(url,username,password)*;
- *runImmutableQuery(Query)*: esegue una query di tipo SELECT. Rilancia l'eccezione *SQLException* se la query non è di tipo SELECT o se è avvenuto un errore SQL generico. Infatti, viene utilizzato il metodo *java.sql.PreparedStatement .executeQuery* [100] che supporta solo le query che non modificano il database;
- *runMutableQuery(Query)*: esegue una query non di tipo SELECT ma che modifica il database (ad esempio UPDATE). Rilancia l'eccezione *SQLException* se la query è di tipo SELECT o se è avvenuto un errore SQL generico. Viene utilizzato il metodo *java.sql.PreparedStatement.execute* che supporta solo le query che modificano il database;
- *addBatch(Query)*: inserisce una query nel batch[2] (per un'esecuzione più veloce, utile nelle INSERT e UPDATE). Il batch viene gestito utilizzando il metodo *java.sql.PreparedStatement.addBatch()* [101];
- *executeBatch()*: esegue le query contenute nel batch. Rilancia le eccezioni *ConnectionParameterNotValid*, se la connessione o il batch non sono inizializzati, e *SQL Exception* se avviene un errore SQL generico durante l'esecuzione del batch.

- **Query class**. Modella una query da inviare al database. Il costruttore richiede una stringa che corrisponde ad una query parametrizzata dove il parametro è segnato con un

[1] Le eccezioni vengono documentate successivamente nel Manuale Programmatore.

[2] Si tratta di un insieme di istruzioni che vengono eseguite insieme.

'?' (ad esempio SELECT * FROM TABELLA WHERE nome=?). Questo approccio offre una maggior flessibilità nel definire i parametri della query. Offre il seguente metodo:

  - *setParameter(int,String)*: permette di settare una stringa come parametro alla posizione specificata. Sono supportate solo le stringhe come parametro delle query. Nella Sezione 7.2 si discute come il supporto ad altri tipi possa rappresentare una possibile estensione del sistema. Un esempio di utilizzo è *new Query("SELECT * FROM TABELLA WHERE nome=?").setParameter(1,"Lorenzo")*

- **Tuple class**. Modella una tupla(riga) ritornata dal database. Viene usata per ritornare i risultati di una query SELECT al client. Offre i seguenti metodi:
  - *setColumn(String, String)*: permette di inserire il mapping tra il nome della colonna e l'attributo (ad esempio ID: 1);
  - *toString()*: ritorna l'oggetto Tuple in formato stringa che corrisponde ad una mappa in formato stringa.

Package *Crypto*

Gestisce le operazioni crittografiche. La Figura B.2 mostra il diagramma UML del package.

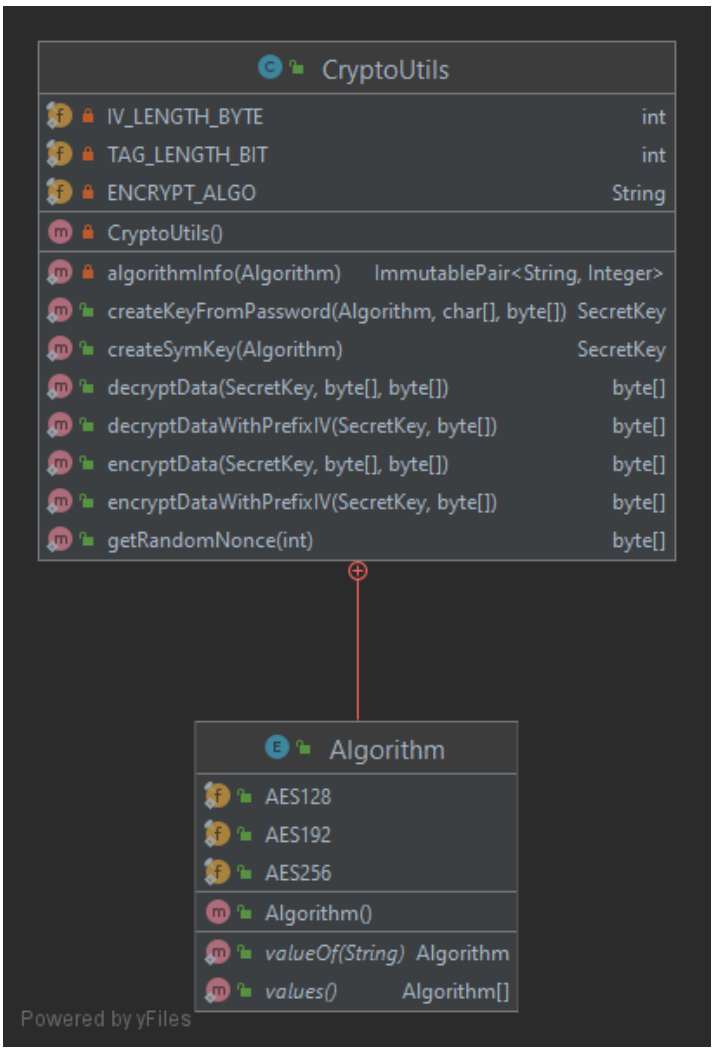


Figura B.2: Package Crypto - Diagramma UML

È composto da una classe static che offre il supporto alle operazioni crittografiche. Gli algoritmi supportati sono quelli dell'enum Algorithm (definito nel file CryptoUtils.java) e sono AES128, AES192 e AES256 tutti in modalità GCM, preferito per i vantaggi dell'*Authenticated*

Encryption With Associated Data (garantisce non solo confidenzialità ma anche verifica dell'integrità e autenticazione con performance migliori) [102]. Il Security Provider[3] utilizzato è SunJCE versione 11 che è un framework crittografico nativo del JRE[4]. Siccome l'algoritmo AES-GCM è standard non è stato necessario installare altri Security Provider che fornissero implementazioni custom o più avanzate. Il programma *CryptoProviderList.java* (`https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/test/java/examples/CryptoProviderList.java`) permette di visualizzare tutti i Security Provider installati sul JRE.

La classe *CryptoUtils* offre i seguenti metodi:

- *createKeyFromPassword(Algorithm, char[], byte[])*: permette di creare una chiave segreta partendo da una password ed applicando una *Key Derivation Function* [103] (in particolare PBKDF2) che prende come input il sale, la password e il numero di iterazioni (fisso a 65536 per una migliore sicurezza perdendo però in performance);
- *createSymKey(Algorithm)*: permette di creare una chiave simmetrica per l'algoritmo specificato;
- *decryptData(SecretKey,byte[],byte[])*: permette di decrifrare dei dati utilizzando la chiave simmetrica e l'iv[5] passati come parametri. Rilancia l'eccezione *DecryptionError* se avviene un errore durante il processo di decifratura;
- *decryptDataWithPrefixIV(SecretKey, byte[])*: permette di estrarre l'iv dai dati cifrati ed infine decifrarli. Rilancia l'eccezione *DecryptionError* se avviene un errore durante il processo di decifratura;
- *encryptData(SecretKey, byte[], byte[])*: permette di cifrare dei dati utilizzando la chiave simmetrica e l'iv passati come parametri. Rilancia l'eccezione *EncryptionError* se avviene un errore durante il processo di cifratura;
- *encryptDataWithPrefixIV(SecretKey, byte[])*: permette di cifrare i dati ed inserire l'iv in essi (concatenandoli uno di seguito all'altro, prima i dati e poi l'iv). Rilancia l'eccezione *EncryptionError* se avviene un errore durante il processo di cifratura;
- *getRandomNonce(int)*: ritorna un nonce [104] della dimensione specificata.

Package *Keystore*

Gesctisce le operazioni sul keystore. La Figura B.3 mostra il diagramma UML del package.

- **KeystoreInfo class**. Si tratta di una classe wrapper che modella le informazioni del keystore ovvero il keystore stesso (un file pkcs12, la Sezione 6.1 contiene la motivazione di questa scelta) e la password che lo protegge. Il costruttore riceve come parametri un oggetto di tipo Keystore [87] e una stringa che rappresenta la password.
- **KeystoreUtils class**. Si tratta una classe static che offre il supporto alla gestione del keystore. Il tipo di keystore supportato è un file *.p12*. Offre i seguenti metodi:
  - *createKeystore(String)*: permette di creare un oggetto di tipo Keystore e proteggerlo con la password passata come parametro. Rilancia *KeystoreOperationError* se avviene un errore durante il processo di creazione del keystore;
  - *deleteKeystore(String)*: permette di eliminare il keystore salvato al path del filesystem specificato come parametro. Rilancia *KeystoreOperationError* se avviene un errore durante il processo di cancellazione del keystore dal filesystem;
  - *deleteKey(KeystoreInfo, String)*: permette di eliminare la chiave con il nome specificato dal keystore passati come parametri. Rilancia *KeystoreOperationError* se avviene un errore durante il processo di cancellazione della chiave;

[3] Si tratta dell'infrastruttura che fornisce l'implementazione degli algoritmi crittografici.

[4] Java Runtime Environment.

[5] Initialization Vector.

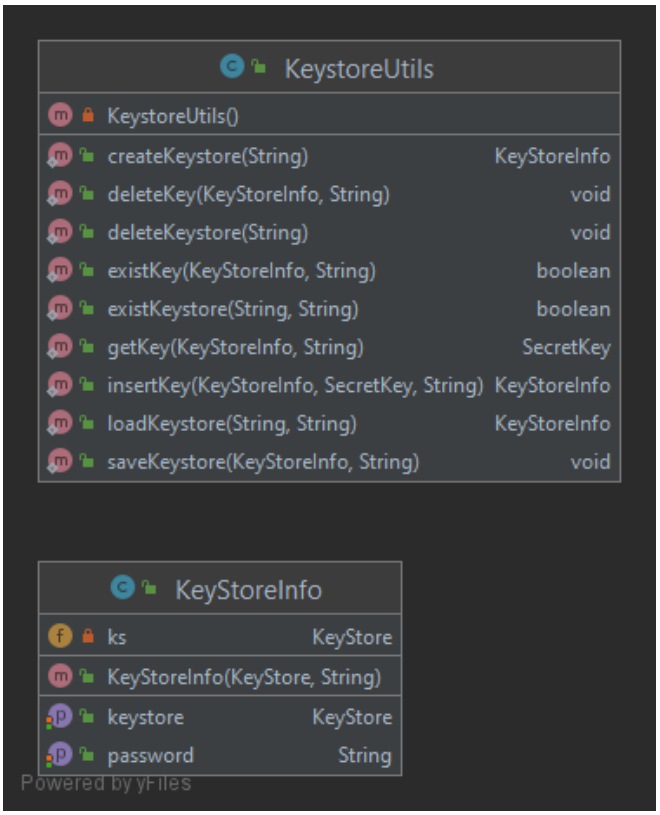


Figura B.3: Package Keystore - Diagramma UML

- *existKey(KeystoreInfo, String)*: permette di verificare se uno specifico keystore contiene una chiave il cui nome è passato come parametro. Ritorna **true** se la chiave esiste, **false** altrimenti;
- *existKeystore(String, String)*: permette di verificare se uno specifico keystore esiste nel filesystem. Ritorna **true** se il keystore esiste, **false** altrimenti;
- *getKey(KeystoreInfo, String)*: ritorna la chiave specificata dal keystore. Rilancia le eccezioni *KeystoreOperationError*, se avviene un errore durante il processo di ottenimento della chiave dal keystore, e *KeyDoesNotExistException* se la chiave cercata non esiste;
- *insertKey(KeystoreInfo, SecretKey, String)*: permette di inserire una chiave nel keystore protetto dalla password specificata, Rilancia *KeystoreOperationError* se avviene un errore durante il processo di inserimento della chiave nel keystore;
- *loadKeystore(String, String)*: permette di caricare il keystore in memoria dal filesystem. Rilancia le eccezioni *FileNotFoundException*, se il keystore non esiste, e *KeystoreOperationError* se avviene un errore durante il processo di caricamento del keystore in memoria centrale;
- *saveKeystore(KeystoreInfo, String)*: permette di salvare il keystore nel filesystem. Rilancia *KeystoreOperationError* se avviene un errore durante il processo di salvataggio del keystore nel filesystem.

Package *Token*

La Figura B.4 mostra il diagramma UML del package.

- **Token interface**. Rappresenta l'astrazione di un token che corrisponde al dato salvato nel database;
- **Encrypted Token**. Si tratta di un classe che implementa l'interfaccia Token e modella i dati cifrati nel DB. Ridefinisce il metodo generateToken() che genera il token nel seguente modo: *'CIPHERTEXT'.Base64(ciphertext).Base64(cipherkey)* dove *ciphertext*

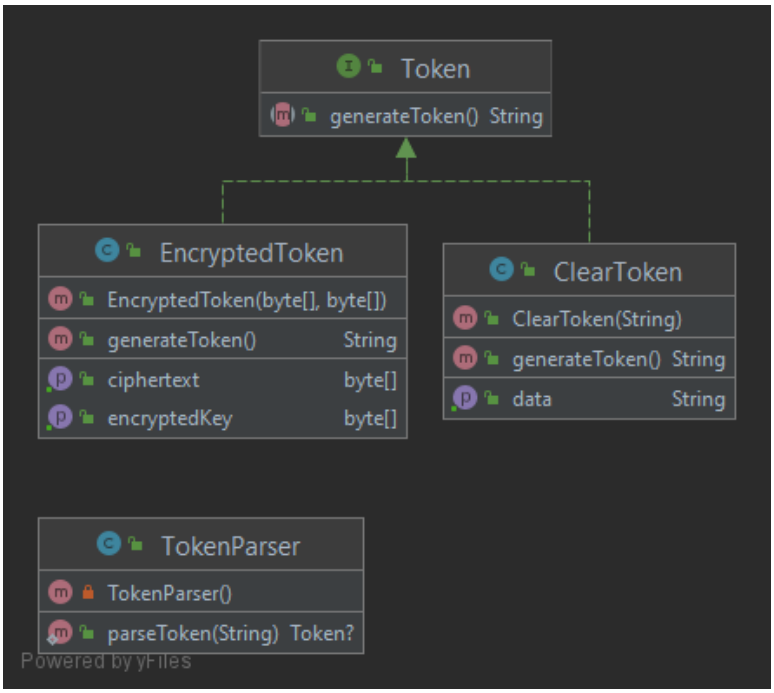


Figura B.4: Package Token - Diagramma UML

sono i dati cifrati con la chiave di cifratura EK generata on-the-fly mentre *cipherkey* è la chiave EK cifrata con la Master Encryption Key memorizzata nel keystore;

- **ClearToken**. Si tratta di una classe che implementa l'interfaccia Token e modella i dati in chiaro nel DB. Ridefinisce il metodo *generateToken()* che genera il token nel seguente modo: *'PLAINTEXT'.Base64(plaintext)* dove *plaintext* corrisponde ai dati in chiaro.
- **Token Parser**. Si tratta di una classe static che permette di estrarre le informazioni contenute nel token esaminando l'header e ritorna un oggetto di tipo Token. L'header nel token è stato aggiunto per permettere una differenziazione tra i due tipi di token in fase di estrapolazione dei dati contenuti in esso.

Il salvataggio dei dati in questo formato aumenta la dimensione dello storage, come mostrato nella Sezione 7.1.1. Però, permette una maggior flessibilità e semplicità nella gestione dei dati cifrati. Tutte le informazioni vengono incapsulate in un unico oggetto. Infatti, la soluzione alternativa, adottata anche da Microsoft Always Encrypted (discusso nella Sezione 4.5), che consiste nel memorizzare tutte le informazioni dei dati nei metadati della tabella, possiede lo svantaggio di essere dispersiva nell'andare a gestire diverse parti della tabella. Ciò introduce una maggior complessità nell'effettuare le operazioni in modo corretto oltre all'aumento della dimensione della tabella (seppur minore rispetto al Token) che viene "arricchita" di numerosi metadati (che contengono ad esempio le chiavi di cifrature cifrate dei dati). Quindi, riassumendo, la seconda soluzione consiste nello spostare i vari campi del Token nei metadati della tabella andando, però, ad aumentare la complessità.

Package *Logger*. Contiene le classi per implementare un sistema di logging basato su file. Il formato dei log definito dalla classe *CustomFormatter* è del seguente tipo: **[TIMESTAMP] [LOG_TYPE] Message** dove TIMESTAMP rappresenta la data e l'ora in formato stringa, LOG_TYPE si riferisce alla criticità del messaggio, ovvero differisce i messaggi tra informazioni, avvisi ed errori, mentre Message corrisponde al messaggio del log.

Package *Exception*. Contiene le eccezioni utilizzate dal sistema per la gestione degli errori riguardo l'IO, Database, Keystore e operazioni crittografiche. Le eccezioni definite sono:

- *ClientsideEncryptionError*. È l'eccezione più generica, viene rilanciata dalla classe CryptoDatabaseAdapter per segnalare un errore generico durante il processo della clientside encryption.
- *ConfigurationFileError*. Usata per segnalare un errore sul file di configurazione.
- *ConnectionParameterNotValid*. Usata per segnalare un errore sui parametri della connessione con il DB.
- *DecryptionError*. Usata per segnalare un errore generico durante il processo di decifratura.
- *EncryptionError*. Usata per segnalare un errore generico durante il processo di cifratura.
- *InitializationError*. Usata per segnalare un errore nel metodo *init()* della classe CryptoDatabaseAdapter.
- *InvalidQueryException*. Usata per segnalare un errore nella definizione di una query.
- *KeyDoesNotExistException*. Usata per segnalare che una chiave cercata non esiste nel keystore.
- *KeystoreOperationError*. Usata per segnalare un errore generico durante i processi di gestione del keystore.
- *QueryExecutionError*. Usata per segnalare un errore generico durante l'esecuzione di una query.

Configuration class

La Figura B.5 mostra il diagramma UML.

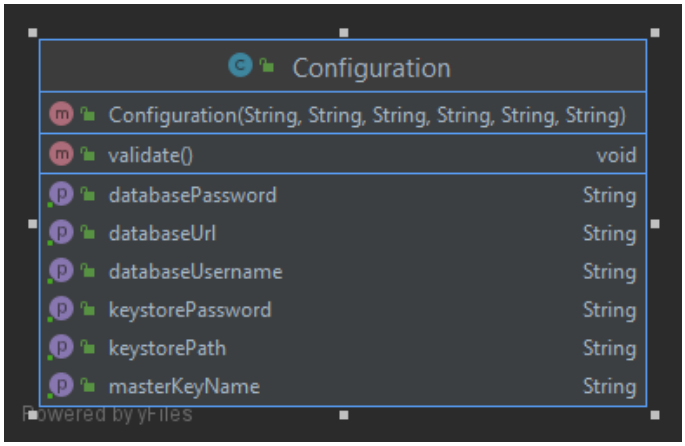


Figura B.5: Configuration class - Diagramma UML

Il costruttore riceve sette stringhe come parametri ovvero l'url del database, username e password per il database, il percorso del filesystem dove è presente il keystore, la password che protegge il keystore, il nome della Master Encryption Key e il path del file di log. Quest'ultimo può essere settato a *null* (in questo caso il sistema di logging viene disabilitato).

Il metodo *validate()* controlla se tutti i campi (tranne logFilePath) sono settati e diversi da null.

B.2.2 *CryptoDatabaseAdapter* class

Si tratta di una classe per l'utilizzo trasparente del sistema (ovvero senza utilizzare direttamente le classi fornite dal package Core). Un'applicazione client può istanziare un oggetto di tipo CryptoDatabaseManager, configurarlo e infine interagire con esso per eseguire query al database. Un possibile utilizzo è mostrato nella Sezione A.4 del Manuale Utente.

La Figura B.6 mostra il diagramma UML.

La classe CryptoDatabaseAdapter possiede due inner class, Builder e QueryBuilder.

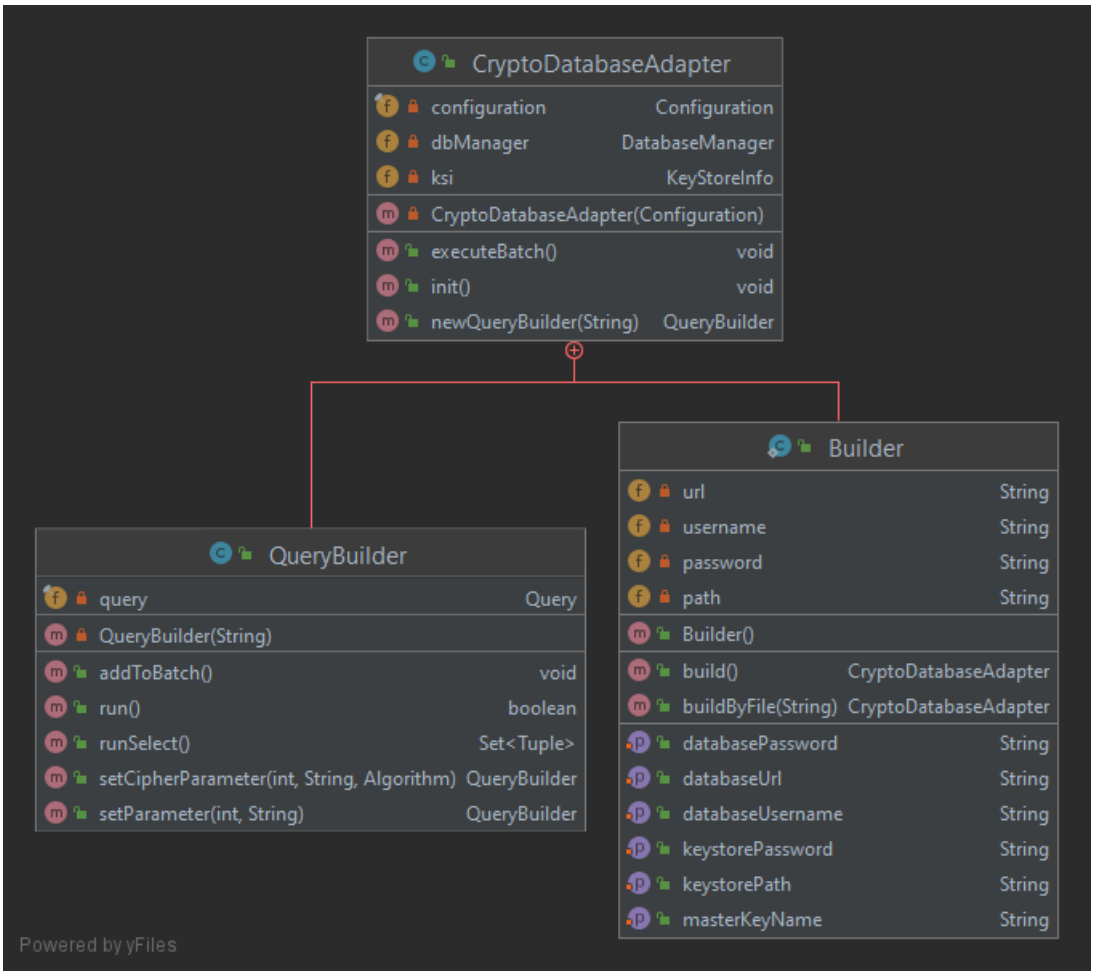


Figura B.6: CryptoDatabaseAdapter class - Diagramma UML

- **Builder**. Si tratta di una classe statica che abilita il Builder Pattern [105], ovvero viene utilizzata per creare e configurare un oggetto di tipo CryptoDatabaseAdapter. Sono possibili due diversi metodi di configurazione:
  - Attraverso un file di configurazione (vedi `https://github.com/LorenzoCeccarelli/Tesi/blob/main/src/main/resources/config.properties`) e richiamando il metodo *buildByFile(path)* dove *path* è il percorso del filesystem dove è situato il file di configurazione. La Figura B.7 mostra un esempio di file di configurazione.
  - Richiamando i metodi set*Property* che vanno a configurare un singolo parametro alla volta.

  Il metodo *build()* valida la configurazione e in caso positivo crea e ritorna un oggetto di tipo CryptoDatabaseAdapter.
- **QueryBuilder**. Si tratta di una classe che permette la creazione e configurazione dei parametri di una query. Il costruttore riceve come parametro una stringa in formato parametrico (con '?'). Offre i seguenti metodi:
  - *setCipherParameter(int, String, Algorithm)*: permette di impostare un parametro e cifrarlo con l'algoritmo specificato nei parametri;
  - *setParameter(int, String)*: permette di impostare un parametro in chiaro nella query;
  - *run()*: esegue una query che modifica il DB (ad esempio UPDATE);
  - *runSelect()*: esegue una query di tipo SELECT, estrae ed eventualmente decifra i dati cifrati e li ritorna all'applicazione client in modo trasparente;

```
#Database settings
DatabaseUrl = jdbc:mysql://localhost:6789/tesi
DatabaseUsername = groot
DatabasePassword = passw0rd

#Keystore settings
#A keystore is created if KeystorePath does not refer an existing
    keystore
KeystorePath = ./src/main/resources/keystore.p12
KeystorePassword = fort

#Encryption settings
MasterKeyName = MEK

#Logging settings
#Logging is disable if LogFilePath is not specified
LogFilePath = ./log.txt
```

Figura B.7: Esempio di file di configurazione.

  - *addToBatch()*: permette di inserire la query costruita in un batch.

- **CryptoDatabaseAdapter**. Si tratta della classe che si intefaccia con l'applicazione client. Il costruttore è privato e richiamabile solo dal Builder. Offre i seguenti metodi:
  - *init()*: inizializza il sistema (da richiamare sempre);
  - *newQueryBuilder(String)*: permette di costruire un oggetto di tipo QueryBuilder per costruire, configurare ed eseguire una query;
  - *executeBatch()*: esegue le istruizioni contenute nel batch.

B.3 Test

La cartella "Test/java" (https://github.com/LorenzoCeccarelli/Tesi/tree/main/src/test/java) contiene due sottocartelle: examples e junit.

All'interno di **examples** sono contenuti diversi programmi *main* che mostrano l'utilizzo del sistema nelle diverse modalità (trasparente e diretto) andando a simulare una possibile applicazione client. Inoltre sono contenuti i programmi utilizzati per la valutazione delle performance e dello storage overhead.

junit invece contiene due suite di test junit per il testing delle funzioni crittografiche e per l'intero processo della clientside encryption.

Appendice C

Comandi

Passo	Descrizione	Comando
0	Prerequisiti	Oracle Database Express 18c
1	Impostare la locazione del keystore nel file SQLNET.ORA	1. cd $ORACLE_BASE/ 2. mkdir wallets 3. Aggiungere nel file sqlnet.ora WALLET_ROOT=wallets
2	Creazione del Password-based Keystore	1. sqlplus sys/password as sysdba 2. ADMINISTER KEY MANAGEMENT CREATE KEYSTORE PosizioneWallet IDENTIFIED BY Password;
3	Apertura del keystore	ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY Password;
4	Impostare la Master Encryption Key	ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY Password WITH BACKUP USING 'tde_key';

Tabella C.1: Istruzioni attivazione Oracle Transparent Data Encryption

Generazione tablespace senza TDE

`CREATE TABLESPACE NOTDE`
`DATAFILE LOCAZIONE_DATAFILE`
`SIZE 1M AUTOEXTEND ON NEXT 1M;`

Tabella C.2: Oracle - Generazione Tablespace senza TDE

Generazione tablespace con TDE a granularità tablespace con AES256

```
CREATE TABLESPACE TDE_AES256
DATAFILE LOCAZIONE_DATAFILE SIZE 1M AUTOEXTEND ON NEXT 1M ENCRYPTION
USING 'AES256' ENCRYPT;
```

Tabella C.3: Oracle - Generazione tablespace con TDE cifrato con AES256

Generazione tablespace con TDE a granularità tablespace con AES192

```
CREATE TABLESPACE TDE_AES192
DATAFILE LOCAZIONE_DATAFILE SIZE 1M AUTOEXTEND ON NEXT 1M ENCRYPTION
USING 'AES192' ENCRYPT;
```

Tabella C.4: Oracle - Generazione tablespace con TDE cifrato con AES192

Generazione tablespace con TDE a granularità tablespace con AES128

```
CREATE TABLESPACE TDE_AES128
DATAFILE LOCAZIONE_DATAFILE SIZE 1M AUTOEXTEND ON NEXT 1M ENCRYPTION
USING 'AES128' ENCRYPT;
```

Tabella C.5: Oracle - Generazione tablespace con TDE cifrato con AES128

Generazione tablespace con TDE a granularità colonna

```
CREATE TABLESPACE TDE_COLUMN
DATAFILE LOCAZIONE_TABLESPACE SIZE 1M AUTOEXTEND ON NEXT 1M;
```

Tabella C.6: Oracle - Generazione tablespace con TDE a livello colonna

Generazione tabella di test nei tablespace senza e con TDE

```
CREATE TABLE NOTDE_TEST(
    ID INTEGER PRIMARY KEY,
    NOME VARCHAR(50),
    COGNOME VARCHAR(50),
    NUMEROCARTACREDITO VARCHAR(50),
    CITTA VARCHAR(50)
)
TABLESPACE NOTDE
```

Tabella C.7: Oracle - Generazione tabella nei tablespace senza e con TDE

Generazione tabella di test nel tablespace con TDE a granularità colonna

```
CREATE TABLE TDE_COLUMN_TEST(
    ID INTEGER PRIMARY KEY,
    NOME VARCHAR(50),
    COGNOME VARCHAR(50),
    NUMEROCARTACREDITO VARCHAR(50) ENCRYPT,
    CITTA VARCHAR(50)
)
TABLESPACE TDE_COLUMN
```

Tabella C.8: Oracle - Generazione tabella nel tablespace con TDE a granularità colonna

SELECT

```
set autot on explain stat;
set timing on;
SELECT *
FROM TABELLA
```

Tabella C.9: Oracle TDE - SELECT

INSERT

```
set autot on explain stat;
set timing on;
INSERT INTO TABLE (ID, NOME, COGNOME, NUMEROCARTACREDITO, CITTA)
SELECT *
FROM random.Generator
```

Tabella C.10: Oracle TDE - INSERT

UPDATE

```
set autot on explain stat;
set timing on;
UPDATE TABLE Tabella
SET Citta='Torino'
```

Tabella C.11: Oracle TDE - UPDATE

DELETE

```
DELETE FROM Tabella
```

Tabella C.12: Oracle TDE - DELETE

SELECT

```
set autot on explain stat;
set timing on;
select
UTL_I18N.RAW_TO_CHAR(dbms_crypto.decrypt(id,6+256+4096,
utl_i18n.string_to_raw('passwordsicuriss','AL32UTF8')),'AL32UTF8'),
UTL_I18N.RAW_TO_CHAR(dbms_crypto.decrypt(nome,6+256+4096,
utl_i18n.string_to_raw('passwordsicuriss','AL32UTF8')),'AL32UTF8'),
UTL_I18N.RAW_TO_CHAR(dbms_crypto.decrypt(cognome,6+256+4096,
utl_i18n.string_to_raw('passwordsicuriss','AL32UTF8')),'AL32UTF8'),
UTL_I18N.RAW_TO_CHAR(dbms_crypto.decrypt(numerocartacredito,
6+256+4096,utl_i18n.string_to_raw('passwordsicuriss','AL32UTF8')),'AL32UTF8'),
UTL_I18N.RAW_TO_CHAR(dbms_crypto.decrypt(citta,6+256+4096,
utl_i18n.string_to_raw('passwordsicuriss','AL32UTF8')),'AL32UTF8')
from aes12810k
```

Tabella C.13: Oracle DBMS_CRYPTO - SELECT

UPDATE

```
set autot on explain stat;
set timing on;
update aes12810k
set numerocartacredito=dbms_crypto.encrypt(
utl_i18n.string_to_raw(1,'AL32UTF8'),
6+256+4096,
utl_i18n.string_to_raw('passwordsicuriss','AL32UTF8'))
```

Tabella C.14: Oracle DBMS_CRYPTO - UPDATE

Configurazione della Transparent Data Encryption

```
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Passwordsicurissima';
go
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'TDECertificate';
go
USE Tesi;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO
ALTER DATABASE Tesi
SET ENCRYPTION ON;
GO

-- Questo in particolare per AES128, per gli altri
    algoritmi usare AES_192 o AES_256 come ALGORITHM
```

Tabella C.15: Istruzioni utilizzate per configurare la Microsoft SQL Server TDE.

Creazione tabella

```
create table TABELLA
(
    ID int primary key,
    NOME nvarchar(50) not null,
    COGNOME nvarchar(50) not null,
    NUMEROCARTACREDITO nvarchar(50) not null,
    CITTA nvarchar(50) not null
)
```

Tabella C.16: Microsoft TDE - Creazione tabella

SELECT

```
SET STATISTICS TIME ON;
SELECT *
FROM TABELLA
SET STATISTICS TIME OFF;
```

Tabella C.17: Microsoft TDE - SELECT

INSERT

```
SET STATISTICS TIME ON;
INSERT INTO TABELLA
SELECT * FROM TABELLA_GENARATOR
SET STATISTICS TIME OFF;
```

Tabella C.18: Microsoft TDE - INSERT

UPDATE

```
SET STATISTICS TIME ON;
UPDATE TABELLA
SET Citta='Torino';
SET STATISTICS TIME OFF;
```

Tabella C.19: Microsoft TDE - UPDATE

DELETE

```
SET STATISTICS TIME ON;
DELETE
FROM TABELLA
SET STATISTICS TIME OFF;
```

Tabella C.20: Microsoft TDE - DELETE

CREAZIONE CHIAVE SIMMETRICA

```
CREATE SYMMETRIC KEY AES256
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE AES256;
GO

-- Questo in particolare per AES1256, per gli altri
algoritmi usare AES_128 o AES_192 come ALGORITHM
```

Tabella C.21: Microsoft Transact SQL - Creazione chiave simmetrica

APERTURA CHIAVE SIMMETRICA

```
OPEN SYMMETRIC KEY AES256
DECRYPTION BY CERTIFICATE AES256 WITH PASSWORD='AES256'

-- Questo in particolare per AES128, per gli altri
algoritmi aprire le corrispondenti chiavi
```

Tabella C.22: Microsoft Transact SQL - Apertura chiave simmetrica

SELECT

```
OPEN SYMMETRIC KEY AES128
DECRYPTION BY CERTIFICATE AES128 WITH PASSWORD='AES128';
GO

SET STATISTICS TIME ON;
SELECT
CONVERT(varchar,DECRYPTBYKEY(id)),
CONVERT(varchar,DECRYPTBYKEY(nome)),
CONVERT(varchar,DECRYPTBYKEY(cognome)),
CONVERT(varchar,DECRYPTBYKEY(numerocartacredito)),
CONVERT(varchar,DECRYPTBYKEY(citta))
FROM AES128.dbo.aes128_10k
SET STATISTICS TIME OFF;

-- Questo in particolare per AES128, per gli altri
algoritmi aprire le corrispondenti chiavi
```

Tabella C.23: Microsoft Transact SQL - SELECT

UPDATE

```
OPEN SYMMETRIC KEY AES128
DECRYPTION BY CERTIFICATE AES128 WITH PASSWORD='AES128';
GO

SET STATISTICS TIME ON;
UPDATE AES128.dbo.aes128_10k
SET NUMEROCARTACREDITO=ENCRYPTBYKEY(KEY_GUID('AES128'),'Torino')

SET STATISTICS TIME OFF;

-- Questo in particolare per AES128, per gli altri
algoritmi aprire le corrispondenti chiavi
```

Tabella C.24: Microsoft Transact SQL - UPDATE

Passo	Descrizione	Comando
1	Abilitare il File Key Management Plugin	1. Inserire nel file my.ini la riga plugin_load_add = file_key_management. 2. Creazione del file contenente le chiavi nella forma id:chiave_hex utilizzando per esempio OpenSSL. 3. Configurare il path al file inserendo in my.ini loose_file_key_management_filename = PATH_KEYFILE
2	Abilitare Aria Encryption	SET GLOBAL aria_encrypt_tables=ON;
3	Modificare la tabella da cifrare	ALTER TABLE nomeTabella ENGINE=Aria ROW_FORMAT=PAGE;

Tabella C.25: Istruzioni attivazione MariaDB/Aria Encryption

Creazione tabella

```
create table TABELLA
(
    ID int primary key,
    NOME nvarchar(50) not null,
    COGNOME nvarchar(50) not null,
    NUMEROCARTACREDITO nvarchar(50) not null,
    CITTA nvarchar(50) not null
)
```

Tabella C.26: MariaDB - Creazione tabella

SELECT

```
SET profiling=1;
SELECT *
FROM TABELLA
```

Tabella C.27: MariaDB - SELECT

INSERT
SET profiling=1; INSERT INTO TABELLA SELECT * FROM TABELLA_GENARATOR

Tabella C.28: MariaDB - Insert

UPDATE
SET profiling=1; UPDATE TABELLA SET Citta='Torino';

Tabella C.29: MariaDB - UPDATE

DELETE
SET profiling=1; DELETE FROM TABELLA

Tabella C.30: MariaDB - DELETE

Passo	Descrizione	Comando
1	Abilitare il Keyring File Plugin	Inserire nel file my.ini early-plugin-load=keyring_file.dll
2	Abilitare InnoDB Encryption	SET GLOBAL default_table_encryption=ON;
3	Impostare l'algoritmo	Nelle query settare oppurtunamente la variabile di sistema "block_encryption_mode"

Tabella C.31: Istruzioni InnoDB Encryption

Creazione tabella
create table TABELLA (ID int primary key, NOME nvarchar(50) not null, COGNOME nvarchar(50) not null, NUMEROCARTACREDITO nvarchar(50) not null, CITTA nvarchar(50) not null)

Tabella C.32: MySQL TDE - Creazione Tabella

SELECT
SET profiling=1; SELECT * FROM TABELLA

Tabella C.33: MySQL TDE - SELECT

INSERT
SET profiling=1; INSERT INTO TABELLA SELECT * FROM TABELLA_GENARATOR

Tabella C.34: MySQL TDE - INSERT

UPDATE

```
SET profiling=1;
UPDATE TABELLA
SET Citta='Torino';
```

Tabella C.35: MySQL TDE - UPDATE

DELETE

```
SET profiling=1;
DELETE
FROM TABELLA
```

Tabella C.36: MySQL TDE - DELETE

SELECT

```
SET profiling=1;

SET block_encryption_mode = 'aes-128-cbc';
SET @key_str = SHA2('My secret passphrase',512);
SET @init_vector = 'aaaaaaaaaaaaaaaa';
SET @crypt_nome = AES_ENCRYPT('lorenzo',@key_str,@init_vector);
SET @crypt_cognome = AES_ENCRYPT('ceccarelli',@key_str,@init_vector);
SET @crypt_cartacredito = AES_ENCRYPT('2434133',@key_str,@init_vector);
SET @crypt_citta = AES_ENCRYPT('Terni',@key_str,@init_vector);
select
id,
aes_decrypt(@crypt_nome,@key_str,@init_vector),
aes_decrypt(@crypt_cognome,@key_str,@init_vector),
aes_decrypt(@crypt_cartacredito,@key_str,@init_vector),
aes_decrypt(@crypt_citta,@key_str,@init_vector)
from 10k

-- Per gli algoritmi inserire aes-128-cbc, aes-192cbc,
aes-256-cbc in block_encryption_mode
```

Tabella C.37: MySQL Security Functions - SELECT

UPDATE

```
SET profiling=1;

SET block_encryption_mode = 'aes-128-cbc';
SET @key_str = SHA2('My secret passphrase',512);
SET @init_vector = 'aaaaaaaaaaaaaaaa';
SET @crypt_cartacredito = AES_ENCRYPT('12345678',@key_str,@init_vector);

UPDATE 10k
SET numerocartacredito = @crypt_cartacredito

-- Per gli algoritmi inserire aes-128-cbc, aes-192cbc,
aes-256-cbc in block_encryption_mode
```

Tabella C.38: MySQL Security Functions - UPDATE

ABILITAZIONE PGCRYPTO

```
select * from pg_available_extensions;
create extension if not exist pgcrypto
```

Tabella C.39: PostgreSQL PGCRYPTO - Abilitazione package PGCRYPTO

CREAZIONE TABELLA

```
create table random10k
(
    ID varchar(256) primary key,
    NOME varchar(256) not null,
    COGNOME varchar(256) not null,
    NUMEROCARTACREDITO varchar(256) not null,
    CITTA varchar(256) not null
)
```

Tabella C.40: PostgreSQL PGCRYPTO - Creazione tabella

SELECT

```
create table random10k
select
pgp_sym_decrypt(id::bytea,'key','cipher-algo=aes128'),
pgp_sym_decrypt(nome::bytea,'key','cipher-algo=aes128'),
pgp_sym_decrypt(cognome::bytea,'key','cipher-algo=aes128'),
pgp_sym_decrypt(numerocartacredito::bytea,'key','cipher-algo=aes128'),
pgp_sym_decrypt(citta::bytea,'key','cipher-algo=aes128')
from test10kaes128
-- Per gli algoritmi inserire aes128, aes192, aes256 o blowfish in
    cipher-algo
```

Tabella C.41: PostgreSQL PGCRYPTO - SELECT

UPDATE

```
update test10kaes128
set numerocartacredito=
    pgp_sym_encrypt('1234567','key','cipher-algo=aes128')
-- Per gli algoritmi inserire aes128, aes192, aes256 o blowfish in
    cipher-algo
```

Tabella C.42: PostgreSQL PGCRYPTO - UPDATE

Appendice D

Tabelle Stato Sistema

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	205	4	0.78	58	0
	AES128	208	5	0.63	4.5	0
	AES192	207	5	0.68	4.4	1
	AES256	206	5	0.63	4.5	0
Durante	NO_TDE	250	6	0.89	58	2
	AES128	208	8	0.91	4.5	4
	AES192	207	9	0.89	4.4	5
	AES256	206	10	1.03	4.5	5
Dopo	NO_TDE	250	4	0.73	58	0
	AES128	208	6	0.68	4.5	0
	AES192	207	6	0.71	4.4	0
	AES256	206	7	0.68	4.5	0
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.70	58	0
	AES128	208	5	0.68	4.5	1
	AES192	207	5	0.63	4.4	2
	AES256	206	6	0.72	4.5	0
Durante	NO_TDE	205	8	0.91	58	3
	AES128	208	11	1.23	4.5	6
	AES192	207	12	1.12	4.4	7
	AES256	206	13	1.12	4.5	8
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	208	6	0.72	4.5	0
	AES192	207	6	0.72	4.4	0
	AES256	206	6	0.63	4.5	1

Tabella D.1: TDE - Stato sistema (MariaDB) per la SELECT.

Tabella D.2: TDE - Stato sistema (MySQL) per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	207	5	0.68	60	0
	AES128	209	5	0.73	62	0
	AES192	204	6	0.72	56	0
	AES256	205	4	0.51	56	1
Durante	NO_TDE	207	6	0.69	60	1

	AES128	209	7	0.84	62	1
	AES192	204	8	0.89	56	2
	AES256	205	7	0.88	56	6
Dopo	NO_TDE	207	4	0.63	60	0
	AES128	209	6	0.69	62	0
	AES192	204	5	0.68	56	0
	AES256	205	5	0.67	56	0
Tabella 100k tuple						
Prima	NO_TDE	207	4	0.63	60	1
	AES128	209	6	0.68	62	0
	AES192	204	6	0.68	56	1
	AES256	205	6	0.86	56	0
Durante	NO_TDE	207	7	0.78	60	2
	AES128	209	12	0.91	62	5
	AES192	204	11	0.97	56	6
	AES256	205	12	1.12	56	8
Dopo	NO_TDE	207	5	0.61	60	0
	AES128	209	7	0.73	62	1
	AES192	204	6	0.71	56	0
	AES256	205	5	0.82	56	1

Tabella D.3: TDE - Stato sistema (Microsoft) per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	212	9	0.75	57	0
	AES128	216	6	0.75	57	1
	AES192	219	7	0.67	62	1
	AES256	216	6	0.74	62	0
Durante	NO_TDE	212	10	0.84	57	1
	AES128	216	14	1.12	57	3
	AES192	219	10	0.84	62	2
	AES256	216	8	0.87	62	1
Dopo	NO_TDE	212	7	0.74	57	0
	AES128	216	7	0.7	57	0
	AES192	219	6	0.72	62	0
	AES256	216	6	0.72	62	0
Tabella 100k tuple						
Prima	NO_TDE	212	8	0.88	57	0
	AES128	216	6	0.78	57	1
	AES192	219	6	0.86	62	0
	AES256	216	7	0.7	62	1
Durante	NO_TDE	212	11	1.02	57	3
	AES128	216	15	1.17	57	3
	AES192	219	12	0.97	62	1
	AES256	216	12	1.03	62	3
Dopo	NO_TDE	212	8	0.83	57	0
	AES128	216	9	0.72	57	0
	AES192	219	7	0.78	62	0
	AES256	216	6	0.78	62	0

Tabella D.4: TDE - Stato sistema (Oracle) per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						

Prima	NO_TDE	205	4	0.78	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.75	59	0
	AES256	205	4	0.68	58	1
	COLUMN	205	5	0.72	61	1
	3DES	206	5	0.78	60	1
Durante	NO_TDE	205	6	0.89	58	2
	AES128	202	8	0.95	57	2
	AES192	204	7	0.97	59	1
	AES256	205	9	0.95	58	4
	COLUMN	205	7	0.91	61	2
	3DES	206	11	1.02	60	3
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.78	59	0
	AES256	205	4	0.72	58	0
	COLUMN	205	5	0.69	61	0
	3DES	206	4	0.72	60	0
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.70	58	0
	AES128	202	5	0.72	57	0
	AES192	204	5	0.73	59	1
	AES256	205	4	0.75	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	5	0.68	60	0
Durante	NO_TDE	205	8	0.91	58	3
	AES128	202	11	1.21	57	4
	AES192	204	12	1.22	59	3
	AES256	205	11	1.19	58	5
	COLUMN	205	9	1.12	61	2
	3DES	206	14	1.28	60	6
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.74	57	0
	AES192	204	5	0.72	59	1
	AES256	205	5	0.68	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	4	0.71	60	1

Tabella D.5: TDE - Stato sistema (MariaDB) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	207	5	0.79	58	0
	AES128	208	6	0.72	58	0
	AES192	207	6	0.72	56	1
	AES256	206	6	0.72	56	1
Durante	NO_TDE	207	6	0.88	58	2
	AES128	208	8	0.89	58	3
	AES192	207	9	0.91	56	3
	AES256	206	10	1.03	56	3
Dopo	NO_TDE	207	4	0.68	58	0
	AES128	208	5	0.68	58	1
	AES192	207	5	0.68	56	0
	AES256	206	5	0.63	56	0
Tabella 100k tuple						

Prima	NO_TDE	207	5	0.73	58	0
	AES128	208	5	0.68	58	1
	AES192	207	5	0.68	56	0
	AES256	206	7	0.68	56	0
Durante	NO_TDE	207	19	1.23	58	13
	AES128	208	30	1.56	58	15
	AES192	207	31	1.61	56	16
	AES256	206	32	1.45	56	15
Dopo	NO_TDE	207	6	0.70	58	2
	AES128	208	6	0.72	58	0
	AES192	207	6	0.68	56	0
	AES256	206	6	0.63	56	0

Tabella D.6: TDE - Stato sistema (MySQL) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	207	4	0.78	81	0
	AES128	205	5	0.66	59	0
	AES192	205	6	0.73	56	1
	AES256	202	6	0.76	56	1
Durante	NO_TDE	207	6	0.89	81	3
	AES128	205	8	0.91	59	4
	AES192	205	13	1.03	56	10
	AES256	202	9	0.98	56	20
Dopo	NO_TDE	207	4	0.73	81	1
	AES128	205	5	0.73	59	0
	AES192	205	7	0.75	56	1
	AES256	202	5	0.68	56	1
Tabella 100k tuple						
Prima	NO_TDE	207	4	0.70	81	0
	AES128	205	7	0.73	59	0
	AES192	205	7	0.68	56	0
	AES256	202	7	0.73	56	0
Durante	NO_TDE	207	8	0.91	81	12
	AES128	205	35	1.56	59	26
	AES192	205	32	1.45	56	37
	AES256	202	32	1.67	56	27
Dopo	NO_TDE	207	4	0.73	81	1
	AES128	205	6	0.68	59	1
	AES192	205	6	0.65	56	2
	AES256	202	6	0.78	56	0

Tabella D.7: TDE - Stato sistema (Microsoft) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	216	7	0.67	56	0
	AES128	217	6	0.75	58	0
	AES192	223	6	0.92	62	0
	AES256	215	6	0.82	62	0
Durante	NO_TDE	216	9	0.84	56	1
	AES128	217	26	1.8	58	20
	AES192	223	9	1.23	62	4

	AES256	215	10	0.98	62	3
Dopo	NO_TDE	216	8	0.76	56	0
	AES128	217	6	0.72	58	0
	AES192	223	6	0.94	62	0
	AES256	215	6	0.80	62	1
Tabella 100k tuple						
Prima	NO_TDE	216	8	0.80	56	0
	AES128	217	6	0.82	58	0
	AES192	223	6	0.83	62	0
	AES256	215	6	0.80	62	0
Durante	NO_TDE	216	20	1.4	56	7
	AES128	217	28	0.82	58	30
	AES192	223	28	1.8	62	24
	AES256	215	30	1.92	62	11
Dopo	NO_TDE	216	6	0.53	56	0
	AES128	217	6	0.72	58	1
	AES192	223	6	0.89	62	1
	AES256	215	6	0.72	62	1

Tabella D.8: TDE - Stato sistema (Oracle) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	205	4	0.78	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.75	59	0
	AES256	205	4	0.68	58	1
	COLUMN	205	5	0.72	61	1
	3DES	206	5	0.78	60	1
Durante	NO_TDE	205	6	0.89	58	2
	AES128	202	13	0.95	57	6
	AES192	204	14	0.97	59	4
	AES256	205	13	0.95	58	7
	COLUMN	205	9	0.91	61	3
	3DES	206	16	1.02	60	8
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.78	59	0
	AES256	205	4	0.72	58	0
	COLUMN	205	5	0.69	61	0
	3DES	206	4	0.72	60	0
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.70	58	0
	AES128	202	5	0.72	57	0
	AES192	204	5	0.73	59	1
	AES256	205	4	0.75	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	5	0.68	60	0
Durante	NO_TDE	205	12	1.02	58	12
	AES128	202	34	1.45	57	45
	AES192	204	37	1.51	59	42
	AES256	205	38	1.52	58	43
	COLUMN	205	22	1.21	61	31
	3DES	206	41	1.72	60	48
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.74	57	0

	AES192	204	5	0.72	59	1
	AES256	205	5	0.68	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	4	0.71	60	1

Tabella D.9: TDE - Stato sistema (MariaDB) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	206	4	0.72	58	0
	AES128	208	7	0.72	58	1
	AES192	207	6	0.68	56	1
	AES256	206	6	0.63	56	1
Durante	NO_TDE	206	8	0.89	58	2
	AES128	208	10	1.03	58	3
	AES192	207	10	0.98	56	3
	AES256	206	11	1.11	56	5
Dopo	NO_TDE	206	5	0.60	58	0
	AES128	208	5	0.58	58	0
	AES192	207	6	0.72	56	0
	AES256	206	6	0.72	56	0
Tabella 100k tuple						
Prima	NO_TDE	206	5	0.61	58	1
	AES128	208	6	0.68	58	0
	AES192	207	6	0.72	56	0
	AES256	206	7	0.68	56	0
Durante	NO_TDE	206	11	1.01	58	3
	AES128	208	15	1.42	58	5
	AES192	207	16	1.34	56	16
	AES256	206	17	1.23	56	6
Dopo	NO_TDE	206	4	0.75	58	0
	AES128	208	6	0.65	58	0
	AES192	207	5	0.68	56	0
	AES256	206	5	0.63	56	0

Tabella D.10: TDE - Stato sistema (MySQL) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	205	4	0.78	58	0
	AES128	204	5	0.70	62	0
	AES192	204	6	0.72	56	0
	AES256	205	5	0.71	56	1
Durante	NO_TDE	205	6	0.89	58	2
	AES128	204	7	0.77	62	1
	AES192	204	12	1.04	56	11
	AES256	205	15	1.23	56	17
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	204	6	0.68	62	0
	AES192	204	7	0.73	56	0
	AES256	205	8	0.72	56	1
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.70	58	0
	AES128	204	6	0.73	62	0

	AES192	204	5	0.68	56	0
	AES256	205	6	0.74	56	1
Durante	NO_TDE	205	8	0.91	58	3
	AES128	204	35	1.56	62	24
	AES192	204	35	1.45	56	18
	AES256	205	36	1.54	56	25
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	204	5	0.69	62	1
	AES192	204	4	0.53	56	1
	AES256	205	4	0.61	56	0

Tabella D.11: TDE - Stato sistema (Microsoft) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	218	6	0.76	57	0
	AES128	215	7	0.81	58	1
	AES192	217	6	0.82	62	0
	AES256	216	6	0.78	59	1
Durante	NO_TDE	218	9	0.90	57	3
	AES128	215	10	0.97	58	3
	AES192	217	9	0.93	62	2
	AES256	216	9	0.92	59	3
Dopo	NO_TDE	218	7	0.75	57	0
	AES128	215	6	0.74	58	0
	AES192	217	5	0.82	62	1
	AES256	216	6	0.72	59	1
Tabella 100k tuple						
Prima	NO_TDE	218	6	0.73	57	1
	AES128	215	6	0.74	58	0
	AES192	217	6	0.80	62	0
	AES256	216	6	0.82	59	1
Durante	NO_TDE	218	27	1.6	57	6
	AES128	215	11	1.15	58	3
	AES192	217	19	1.34	62	6
	AES256	216	23	1.43	59	6
Dopo	NO_TDE	218	7	1.78	57	0
	AES128	215	6	0.86	58	1
	AES192	217	6	0.76	62	0
	AES256	216	6	0.82	59	1

Tabella D.12: TDE - Stato sistema (Oracle) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	205	4	0.78	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.75	59	0
	AES256	205	4	0.68	58	1
	COLUMN	205	5	0.72	61	1
	3DES	206	5	0.78	60	1
Durante	NO_TDE	205	6	0.89	58	2
	AES128	202	8	0.95	57	3
	AES192	204	9	0.97	59	3
	AES256	205	8	0.95	58	4

	COLUMN	205	7	0.91	61	2
	3DES	206	10	1.02	60	4
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.78	59	0
	AES256	205	4	0.72	58	0
	COLUMN	205	5	0.69	61	0
	3DES	206	4	0.72	60	0
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.70	58	0
	AES128	202	5	0.72	57	0
	AES192	204	5	0.73	59	1
	AES256	205	4	0.75	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	5	0.68	60	0
Durante	NO_TDE	205	12	1.02	58	4
	AES128	202	34	1.45	57	7
	AES192	204	37	1.51	59	8
	AES256	205	38	1.52	58	6
	COLUMN	205	22	1.21	61	5
	3DES	206	41	1.72	60	5
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.74	57	0
	AES192	204	5	0.72	59	1
	AES256	205	5	0.68	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	4	0.71	60	1

Tabella D.13: TDE - Stato sistema (MariaDB) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	205	5	0.78	58	0
	AES128	208	7	0.72	58	1
	AES192	207	6	0.72	56	0
	AES256	206	7	0.68	56	0
Durante	NO_TDE	205	6	0.85	58	1
	AES128	208	10	1.03	58	3
	AES192	207	8	0.85	56	2
	AES256	206	9	0.93	56	2
Dopo	NO_TDE	205	4	0.65	58	0
	AES128	208	5	0.58	58	0
	AES192	207	5	0.68	56	0
	AES256	206	6	0.63	56	0
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.58	58	0
	AES128	208	6	0.68	58	0
	AES192	207	6	0.63	56	0
	AES256	206	5	0.63	56	0
Durante	NO_TDE	205	6	0.78	58	1
	AES128	208	15	1.42	58	5
	AES192	207	10	0.96	56	2
	AES256	206	10	1.06	56	3
Dopo	NO_TDE	205	5	0.63	58	0
	AES128	208	6	0.65	58	0

	AES192	207	6	0.68	56	0
	AES256	206	6	0.72	56	0

Tabella D.14: TDE - Stato sistema (MySQL) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	205	4	0.78	58	0
	AES128	207	4	0.77	59	1
	AES192	204	7	0.79	56	0
	AES256	208	7	0.73	56	1
Durante	NO_TDE	205	6	0.89	58	1
	AES128	207	6	0.88	59	2
	AES192	204	16	1.34	56	9
	AES256	208	22	1.23	56	7
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	207	5	0.73	59	0
	AES192	204	6	0.72	56	1
	AES256	208	6	0.68	56	0
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.70	58	0
	AES128	207	4	0.70	59	0
	AES192	204	6	0.72	56	0
	AES256	208	6	0.68	56	0
Durante	NO_TDE	205	8	0.91	58	3
	AES128	207	39	1.43	59	61
	AES192	204	38	1.67	56	42
	AES256	208	35	1.78	56	33
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	207	7	0.70	59	1
	AES192	204	7	0.78	56	1
	AES256	208	7	0.78	56	1

Tabella D.15: TDE - Stato sistema (Microsoft) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	216	7	0.68	56	0
	AES128	220	6	0.76	58	0
	AES192	223	6	0.87	62	0
	AES256	216	6	0.83	62	0
Durante	NO_TDE	216	8	0.82	56	1
	AES128	220	9	0.91	58	4
	AES192	223	8	1.06	62	3
	AES256	216	9	0.91	62	2
Dopo	NO_TDE	216	7	0.67	56	0
	AES128	220	6	0.72	58	0
	AES192	223	6	0.83	62	0
	AES256	216	5	0.80	62	0
Tabella 100k tuple						
Prima	NO_TDE	216	8	0.8	56	0
	AES128	220	6	0.71	58	1
	AES192	223	6	0.87	62	0
	AES256	216	6	0.80	62	0
Durante	NO_TDE	216	16	1.2	56	10

	AES128	220	17	1.3	58	57
	AES192	223	18	1.42	62	4
	AES256	216	21	1.56	62	27
Dopo	NO_TDE	216	6	0.53	56	0
	AES128	220	6	0.77	58	0
	AES192	223	7	0.73	62	0
	AES256	216	6	0.71	62	2

Tabella D.16: TDE - Stato sistema (Oracle) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_TDE	205	4	0.78	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.75	59	0
	AES256	205	4	0.68	58	1
	COLUMN	205	5	0.72	61	1
	3DES	206	5	0.78	60	1
Durante	NO_TDE	205	6	0.89	58	2
	AES128	202	8	0.95	57	3
	AES192	204	9	0.97	59	3
	AES256	205	8	0.95	58	4
	COLUMN	205	7	0.91	61	2
	3DES	206	10	1.02	60	4
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.73	57	0
	AES192	204	4	0.78	59	0
	AES256	205	4	0.72	58	0
	COLUMN	205	5	0.69	61	0
	3DES	206	4	0.72	60	0
Tabella 100k tuple						
Prima	NO_TDE	205	4	0.70	58	0
	AES128	202	5	0.72	57	0
	AES192	204	5	0.73	59	1
	AES256	205	4	0.75	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	5	0.68	60	0
Durante	NO_TDE	205	12	1.02	58	12
	AES128	202	15	1.12	57	23
	AES192	204	17	1.08	59	24
	AES256	205	13	1.03	58	23
	COLUMN	205	12	1.12	61	21
	3DES	206	14	1.11	60	20
Dopo	NO_TDE	205	4	0.73	58	0
	AES128	202	5	0.74	57	0
	AES192	204	5	0.72	59	1
	AES256	205	5	0.68	58	1
	COLUMN	205	4	0.68	61	0
	3DES	206	4	0.71	60	1

Tabella D.17: SQL Interface - Stato sistema (PostgreSQL) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						

Prima	NO_ENC	211	9	0.89	70	0
	AES128	197	8	0.98	61	0
	AES192	205	11	1.16	61	0
	AES256	210	10	1.19	57	0
	COLUMN	207	11	1.24	54	0
	Blowfish	206	10	1.09	57	0
Durante	NO_ENC	211	10	0.98	70	2
	AES128	197	11	1.23	61	22
	AES192	205	14	1.32	61	5
	AES256	210	16	1.45	57	6
	COLUMN	207	13	1.34	54	5
	Blowfish	206	12	1.21	57	3
Dopo	NO_ENC	211	8	0.83	70	0
	AES128	197	9	0.92	61	1
	AES192	205	10	1.12	61	0
	AES256	210	11	1.23	57	1
	COLUMN	207	11	1.16	54	0
	Blowfish	206	9	1.08	57	0
Tabella 100k tuple						
Prima	NO_ENC	211	9	0.78	70	0
	AES128	197	8	0.82	61	0
	AES192	205	11	1.17	61	0
	AES256	210	12	1.23	57	0
	COLUMN	207	12	1.27	54	0
	Blowfish	206	11	1.18	57	0
Durante	NO_ENC	211	14	1.12	70	11
	AES128	197	29	1.96	61	26
	AES192	205	30	1.96	61	26
	AES256	210	33	1.89	57	17
	COLUMN	207	34	1.89	54	11
	Blowfish	206	27	1.67	57	11
Dopo	NO_ENC	211	9	0.83	70	0
	AES128	197	9	1.03	61	1
	AES192	205	11	1.17	61	0
	AES256	210	10	1.24	57	1
	COLUMN	207	11	1.19	54	1
	Blowfish	206	10	1.12	57	0

Tabella D.18: SQL Interface - Stato sistema (PostgreSQL) per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	209	12	1.36	65	0
	AES128	206	11	1.21	66	1
	AES192	207	11	1.18	57	0
	AES256	207	10	1.07	57	0
	COLUMN	208	11	1.16	54	0
	Blowfish	207	11	1.12	57	0
Durante	NO_ENC	209	14	1.51	65	3
	AES128	206	13	1.34	66	3
	AES192	207	13	1.34	57	2
	AES256	207	12	1.23	57	3
	COLUMN	208	13	1.32	54	2
	Blowfish	207	12	1.45	57	2
Dopo	NO_ENC	209	11	1.29	65	0

	AES128	206	10	1.07	66	0
	AES192	207	11	0.97	57	0
	AES256	207	10	1.12	57	0
	COLUMN	208	10	1.08	54	0
	Blowfish	207	10	1.09	57	0
Tabella 100k tuple						
Prima	NO_ENC	209	13	1.28	65	0
	AES128	206	13	0.93	66	0
	AES192	207	10	1.10	57	0
	AES256	207	12	1.21	57	1
	COLUMN	208	12	1.19	54	1
	Blowfish	207	12	1.19	57	1
Durante	NO_ENC	209	15	1.57	65	4
	AES128	206	15	1.34	66	4
	AES192	207	15	1.54	57	3
	AES256	207	16	1.56	57	4
	COLUMN	208	15	1.54	54	3
	Blowfish	207	15	1.54	57	3
Dopo	NO_ENC	209	12	1.23	65	1
	AES128	206	12	1.03	66	0
	AES192	207	9	1.09	57	0
	AES256	207	9	1.12	57	0
	COLUMN	208	11	1.17	54	0
	Blowfish	207	13	1.21	57	0

Tabella D.19: SQL Interface - Stato sistema (PostgreSQL) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	206	10	1.17	62	0
	AES128	208	11	1.21	61	1
	AES192	206	11	1.22	57	0
	AES256	207	10	1.12	54	0
	COLUMN	208	11	1.12	54	0
	Blowfish	207	10	1.09	57	
Durante	NO_ENC	206	13	1.32	62	2
	AES128	208	13	1.45	61	3
	AES192	206	14	1.54	57	3
	AES256	207	14	1.34	54	5
	COLUMN	208	12	1.23	54	1
	Blowfish	207	12	1.23	57	1
Dopo	NO_ENC	206	10	1.01	62	0
	AES128	208	10	1.02	61	0
	AES192	206	10	1.17	57	0
	AES256	207	11	1.13	54	0
	COLUMN	208	10	1.09	54	0
	Blowfish	207	9	0.98	57	0
Tabella 100k tuple						
Prima	NO_ENC	206	10	1.16	62	0
	AES128	208	13	0.93	61	0
	AES192	206	11	1.12	57	0
	AES256	207	11	1.17	54	0
	COLUMN	208	12	1.19	54	0
	Blowfish	207	11	1.12	57	1
Durante	NO_ENC	206	14	1.45	62	8
	AES128	208	15	1.23	61	4

	AES192	206	15	1.68	57	4
	AES256	207	16	1.54	54	6
	COLUMN	208	14	1.38	54	2
	Blowfish	207	13	1.34	57	2
Dopo	NO_ENC	206	10	1.12	62	0
	AES128	208	11	0.94	61	1
	AES192	206	10	1.03	57	0
	AES256	207	10	1.07	54	0
	COLUMN	208	12	1.23	54	0
	Blowfish	207	10	1.09	57	0

Tabella D.20: SQL Interface - Stato sistema (PostgreSQL) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	211	9	0.89	70	0
	AES128	209	8	1.01	61	0
	AES192	206	10	1.01	61	0
	AES256	210	10	1.19	57	1
	COLUMN	207	10	1.19	54	0
	Blowfish	206	11	1.12	57	0
Durante	NO_ENC	211	10	0.95	70	1
	AES128	209	10	1.23	61	3
	AES192	206	12	1.13	61	8
	AES256	210	12	1.31	57	7
	COLUMN	207	12	1.36	54	2
	Blowfish	206	13	1.21	57	1
Dopo	NO_ENC	211	6	0.85	70	0
	AES128	209	7	0.96	61	1
	AES192	206	9	1.16	61	0
	AES256	210	9	1.09	57	0
	COLUMN	207	10	1.11	54	0
	Blowfish	206	10	1.08	57	0
Tabella 100k tuple						
Prima	NO_ENC	211	8	0.91	70	0
	AES128	209	7	0.93	61	0
	AES192	206	10	0.97	61	0
	AES256	210	11	1.24	57	1
	COLUMN	207	11	1.23	54	0
	Blowfish	206	12	1.21	57	0
Durante	NO_ENC	211	12	1.34	70	8
	AES128	209	9	1.12	61	41
	AES192	206	15	1.45	61	40
	AES256	210	14	1.45	57	47
	COLUMN	207	13	1.56	54	4
	Blowfish	206	14	1.45	57	74
Dopo	NO_ENC	211	7	0.88	70	1
	AES128	209	8	0.99	61	1
	AES192	206	10	0.92	61	1
	AES256	210	11	1.15	57	1
	COLUMN	207	10	1.16	54	0
	Blowfish	206	11	1.18	57	0

Tabella D.21: SQL Interface - Stato sistema (Microsoft) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	216	7	0.67	56	0
	AES128	217	6	0.73	73	0
	AES192	216	7	0.78	57	0
	AES256	212	6	0.70	66	0
	COLUMN	208	5	0.63	68	0
Durante	NO_ENC	216	9	0.84	56	1
	AES128	217	9	0.92	73	2
	AES192	216	34	1.78	57	15
	AES256	212	36	1.56	66	16
	COLUMN	208	31	1.98	68	26
Dopo	NO_ENC	216	8	0.76	56	0
	AES128	217	6	0.72	73	0
	AES192	216	6	0.74	57	1
	AES256	212	8	0.86	66	1
	COLUMN	208	5	0.94	68	1
Tabella 100k tuple						
Prima	NO_ENC	216	8	0.80	56	0
	AES128	217	6	0.68	73	0
	AES192	216	7	0.74	57	0
	AES256	212	5	0.82	66	1
	COLUMN	208	5	0.70	68	0
Durante	NO_ENC	216	20	1.4	56	7
	AES128	217	25	1.45	73	37
	AES192	216	54	1.93	57	48
	AES256	212	58	2.1	66	52
	COLUMN	208	43	1.68	68	54
Dopo	NO_ENC	216	6	0.53	56	0
	AES128	217	6	0.74	73	1
	AES192	216	7	0.95	57	1
	AES256	212	7	0.76	66	2
	COLUMN	208	6	0.82	68	1

Tabella D.22: SQL Interface - Stato sistema (Microsoft) per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	217	5	0.69	76	1
	AES128	218	6	0.89	73	0
	AES192	218	6	0.60	65	0
	AES256	217	5	0.75	73	0
	COLUMN	210	5	0.80	66	0
Durante	NO_ENC	217	7	0.83	76	2
	AES128	218	10	1.21	73	4
	AES192	218	10	0.95	65	3
	AES256	217	8	0.89	73	6
	COLUMN	210	7	0.92	66	2
Dopo	NO_ENC	217	6	0.72	76	1
	AES128	218	6	0.74	73	1
	AES192	218	6	0.72	65	0
	AES256	217	5	0.80	73	1
	COLUMN	210	6	0.76	66	0
Tabella 100k tuple						

Prima	NO_ENC	217	6	0.79	76	0
	AES128	218	6	0.86	73	1
	AES192	218	6	0.78	65	0
	AES256	217	5	0.79	73	1
	COLUMN	210	5	0.87	66	0
Durante	NO_ENC	217	9	0.98	76	9
	AES128	218	17	1.45	73	4
	AES192	218	18	1.43	65	13
	AES256	217	14	1.45	73	14
	COLUMN	210	12	1.23	66	8
Dopo	NO_ENC	217	5	0.73	76	0
	AES128	218	7	0.78	73	0
	AES192	218	7	0.71	65	0
	AES256	217	5	0.54	73	1
	COLUMN	210	5	0.82	66	0

Tabella D.23: SQL Interface - Stato sistema (Microsoft) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	217	6	0.89	76	1
	AES128	218	6	0.73	78	0
	AES192	218	6	0.77	63	0
	AES256	212	5	0.85	66	0
	COLUMN	209	6	0.75	62	0
Durante	NO_ENC	217	10	0.98	76	3
	AES128	218	9	0.98	78	5
	AES192	218	11	1.02	63	4
	AES256	212	8	1.12	66	6
	COLUMN	209	8	0.92	62	6
Dopo	NO_ENC	217	6	0.81	76	1
	AES128	218	6	0.78	78	0
	AES192	218	6	0.81	63	1
	AES256	212	4	0.89	66	0
	COLUMN	209	6	0.71	62	0
Tabella 100k tuple						
Prima	NO_ENC	217	6	0.75	76	0
	AES128	218	6	0.84	78	1
	AES192	218	6	0.82	63	0
	AES256	212	4	0.8	66	1
	COLUMN	209	4	0.78	62	0
Durante	NO_ENC	217	18	1.12	76	5
	AES128	218	20	1.56	78	10
	AES192	218	21	1.67	63	11
	AES256	212	18	1.67	66	11
	COLUMN	209	12	1.23	62	7
Dopo	NO_ENC	217	6	0.67	76	1
	AES128	218	7	0.75	78	1
	AES192	218	7	0.74	63	1
	AES256	212	5	0.60	66	1
	COLUMN	209	4	0.72	62	1

Tabella D.24: SQL Interface - Stato sistema (Microsoft) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	223	7	0.82	61	0
	AES128	219	7	0.62	73	0
	AES192	218	7	0.77	57	0
	AES256	210	6	0.82	66	0
	COLUMN	211	6	0.82	68	1
Durante	NO_ENC	223	9	0.91	61	1
	AES128	219	14	1.09	73	27
	AES192	218	10	0.97	57	8
	AES256	210	10	1.12	66	9
	COLUMN	211	8	0.97	68	3
Dopo	NO_ENC	223	6	0.82	61	0
	AES128	219	6	0.72	73	1
	AES192	218	6	0.81	57	0
	AES256	210	7	0.72	66	1
	COLUMN	211	5	0.76	68	0
Tabella 100k tuple						
Prima	NO_ENC	223	7	0.78	61	0
	AES128	219	7	0.77	73	0
	AES192	218	7	0.80	57	0
	AES256	210	6	0.67	66	0
	COLUMN	211	8	0.83	68	0
Durante	NO_ENC	223	18	1.12	61	22
	AES128	219	22	1.23	73	34
	AES192	218	28	1.67	57	57
	AES256	210	27	1.71	66	49
	COLUMN	211	27	1.34	68	26
Dopo	NO_ENC	223	7	0.74	61	0
	AES128	219	6	0.72	73	2
	AES192	218	7	0.80	57	0
	AES256	210	7	0.71	66	2
	COLUMN	211	7	0.76	68	1

Tabella D.25: SQL Interface - Stato sistema (MySQL) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	206	4	0.78	56	1
	AES128	205	4	0.69	56	0
	AES192	205	4	0.78	56	0
	AES256	203	5	0.72	54	1
	COLUMN	203	6	0.86	56	1
Durante	NO_ENC	206	13	1.12	56	6
	AES128	205	13	0.92	56	13
	AES192	205	16	1.04	56	31
	AES256	203	17	1.23	54	30
	COLUMN	203	16	1.12	56	8
Dopo	NO_ENC	206	5	0.79	56	0
	AES128	205	5	0.83	56	1
	AES192	205	4	0.69	56	1
	AES256	203	5	0.55	54	1
	COLUMN	203	5	0.59	56	1
Tabella 100k tuple						
Prima	NO_ENC	206	4	0.81	56	0

	AES128	205	4	0.78	56	0
	AES192	205	5	0.78	56	0
	AES256	203	6	0.78	54	0
	COLUMN	203	5	0.72	56	1
Durante	NO_ENC	206	30	1.43	56	32
	AES128	205	44	1.78	56	73
	AES192	205	51	1.78	56	84
	AES256	203	55	1.87	54	83
	COLUMN	203	32	1.34	56	35
Dopo	NO_ENC	206	5	0.79	56	0
	AES128	205	4	0.83	56	1
	AES192	205	5	0.81	56	0
	AES256	203	5	0.79	54	1
	COLUMN	203	5	0.71	56	1

Tabella D.26: SQL Interface - Stato sistema (MySQL) per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	208	5	0.79	54	0
	AES128	204	5	0.73	56	1
	AES192	204	5	0.71	56	0
	AES256	206	5	077	54	0
	COLUMN	208	6	0.64	56	1
Durante	NO_ENC	208	8	0.91	54	2
	AES128	204	22	1.45	56	5
	AES192	204	11	1.03	56	4
	AES256	206	40	1.78	54	15
	COLUMN	208	8	0.87	56	3
Dopo	NO_ENC	208	4	0.78	54	0
	AES128	204	6	0.72	56	1
	AES192	204	6	0.74	56	0
	AES256	206	5	0.64	54	1
	COLUMN	208	5	0.79	56	0
Tabella 100k tuple						
Prima	NO_ENC	208	5	0.66	54	1
	AES128	204	5	0.78	56	0
	AES192	204	5	0.78	56	1
	AES256	206	5	0.69	54	0
	COLUMN	208	7	0.55	56	1
Durante	NO_ENC	208	11	1.02	54	3
	AES128	204	45	1.98	56	23
	AES192	204	30	1.56	56	7
	AES256	206	51	1.95	54	18
	COLUMN	208	15	1.12	56	5
Dopo	NO_ENC	208	5	0.78	54	0
	AES128	204	7	0.68	56	1
	AES192	204	5	0.68	56	1
	AES256	206	5	0.78	54	0
	COLUMN	208	5	0.76	56	1

Tabella D.27: SQL Interface - Stato sistema (MySQL) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)

Tabella 10k tuple						
Prima	NO_ENC	207	4	0.72	54	1
	AES128	205	5	0.56	56	0
	AES192	205	6	0.79	54	1
	AES256	205	5	0.74	56	0
	COLUMN	208	6	0.65	56	0
Durante	NO_ENC	207	8	0.99	54	4
	AES128	205	11	0.93	56	6
	AES192	205	15	1.12	54	18
	AES256	205	52	1.82	56	72
	COLUMN	208	16	1.02	56	13
Dopo	NO_ENC	207	5	0.81	54	0
	AES128	205	6	0.67	56	0
	AES192	205	5	0.87	54	0
	AES256	205	5	0.75	56	1
	COLUMN	208	6	0.77	56	0
Tabella 100k tuple						
Prima	NO_ENC	207	6	0.77	54	0
	AES128	205	6	0.70	56	0
	AES192	205	5	0.73	54	0
	AES256	205	5	0.61	56	0
	COLUMN	208	7	0.72	56	0
Durante	NO_ENC	207	37	1.56	54	26
	AES128	205	51	1.92	56	32
	AES192	205	50	1.78	54	59
	AES256	205	58	2.01	56	74
	COLUMN	208	50	1.64	56	33
Dopo	NO_ENC	207	5	0.65	54	1
	AES128	205	5	0.77	56	1
	AES192	205	5	0.78	54	0
	AES256	205	6	0.78	56	0
	COLUMN	208	5	0.78	56	1

Tabella D.28: SQL Interface - Stato sistema (MySQL) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	204	5	0.85	56	0
	AES128	207	5	0.69	56	1
	AES192	204	6	0.76	56	0
	AES256	209	6	0.76	54	0
	COLUMN	206	5	0.78	56	1
Durante	NO_ENC	204	13	1.23	56	6
	AES128	207	11	1.04	56	13
	AES192	204	13	1.05	56	14
	AES256	209	26	1.34	54	31
	COLUMN	206	18	1.23	56	14
Dopo	NO_ENC	204	5	0.78	56	1
	AES128	207	4	0.72	56	1
	AES192	204	5	0.62	56	1
	AES256	209	5	0.68	54	1
	COLUMN	206	6	0.71	56	1
Tabella 100k tuple						
Prima	NO_ENC	204	4	0.79	56	1
	AES128	207	5	0.58	56	0
	AES192	204	5	0.72	56	1

	AES256	209	5	0.82	54	1
	COLUMN	206	6	0.76	56	1
Durante	NO_ENC	204	32	1.54	56	26
	AES128	207	53	1.89	56	64
	AES192	204	57	1.91	56	65
	AES256	209	49	1.79	54	71
	COLUMN	206	41	1.62	56	45
Dopo	NO_ENC	204	5	0.83	56	1
	AES128	207	5	0.95	56	3
	AES192	204	6	0.77	56	0
	AES256	209	5	0.53	54	1
	COLUMN	206	6	0.76	56	0

Tabella D.29: SQL Interface - Stato sistema (Oracle) per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.73	57	0
	AES192	203	5	0.82	56	0
	AES256	206	4	0.68	58	0
	COLUMN	205	5	0.78	55	0
	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	13	1.02	58	5
	AES128	202	15	1.12	57	12
	AES192	203	15	1.09	56	14
	AES256	206	16	1.11	58	14
	COLUMN	205	12	1.04	55	7
	3DES	202	16	1.12	57	15
Dopo	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.68	57	0
	AES192	203	4	0.73	56	0
	AES256	206	5	0.78	58	1
	COLUMN	205	5	0.82	55	1
	3DES	202	4	0.73	57	0
Tabella 100k tuple						
Prima	NO_ENC	206	5	0.82	58	1
	AES128	202	4	0.68	57	1
	AES192	203	5	0.82	56	1
	AES256	206	4	0.78	58	0
	COLUMN	205	5	0.68	55	0
	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	29	1.34	58	26
	AES128	202	37	1.54	57	34
	AES192	203	38	1.56	56	35
	AES256	206	37	1.48	58	34
	COLUMN	205	32	1.32	55	31
	3DES	202	41	1.67	57	37
Dopo	NO_ENC	206	4	0.73	58	0
	AES128	202	4	0.68	57	0
	AES192	203	5	0.78	56	0
	AES256	206	5	0.82	58	0
	COLUMN	205	4	0.73	55	0
	3DES	202	5	0.78	57	0

Tabella D.30: SQL Interface - Stato sistema (Oracle) per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.73	57	0
	AES192	203	5	0.82	56	0
	AES256	206	4	0.68	58	0
	COLUMN	205	5	0.78	55	0
	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	13	1.02	58	5
	AES128	202	15	1.12	57	7
	AES192	203	15	1.09	56	8
	AES256	206	16	1.11	58	7
	COLUMN	205	12	1.04	55	8
	3DES	202	16	1.12	57	6
Dopo	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.68	57	0
	AES192	203	4	0.73	56	0
	AES256	206	5	0.78	58	1
	COLUMN	205	5	0.82	55	1
	3DES	202	4	0.73	57	0
Tabella 100k tuple						
Prima	NO_ENC	206	5	0.82	58	1
	AES128	202	4	0.68	57	1
	AES192	203	5	0.82	56	1
	AES256	206	4	0.78	58	0
	COLUMN	205	5	0.68	55	0
	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	29	1.34	58	12
	AES128	202	37	1.54	57	16
	AES192	203	38	1.56	56	17
	AES256	206	37	1.48	58	15
	COLUMN	205	32	1.32	55	14
	3DES	202	41	1.67	57	17
Dopo	NO_ENC	206	4	0.73	58	0
	AES128	202	4	0.68	57	0
	AES192	203	5	0.78	56	0
	AES256	206	5	0.82	58	0
	COLUMN	205	4	0.73	55	0
	3DES	202	5	0.78	57	0

Tabella D.31: SQL Interface - Stato sistema (Oracle) per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.73	57	0
	AES192	203	5	0.82	56	0
	AES256	206	4	0.68	58	0
	COLUMN	205	5	0.78	55	0
	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	13	1.02	58	5
	AES128	202	15	1.12	57	13
	AES192	203	15	1.09	56	16
	AES256	206	16	1.11	58	17

	COLUMN	205	12	1.04	55	14
	3DES	202	16	1.12	57	21
Dopo	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.68	57	0
	AES192	203	4	0.73	56	0
	AES256	206	5	0.78	58	1
	COLUMN	205	5	0.82	55	1
	3DES	202	4	0.73	57	0
Tabella 100k tuple						
Prima	NO_ENC	206	5	0.82	58	1
	AES128	202	4	0.68	57	1
	AES192	203	5	0.82	56	1
	AES256	206	4	0.78	58	0
	COLUMN	205	5	0.68	55	0
	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	29	1.34	58	21
	AES128	202	56	1.54	57	34
	AES192	203	58	1.56	56	38
	AES256	206	55	1.48	58	41
	COLUMN	205	49	1.32	55	29
	3DES	202	61	1.67	57	42
Dopo	NO_ENC	206	4	0.73	58	0
	AES128	202	4	0.68	57	0
	AES192	203	5	0.78	56	0
	AES256	206	5	0.82	58	0
	COLUMN	205	4	0.73	55	0
	3DES	202	5	0.78	57	0

Tabella D.32: SQL Interface - Stato sistema (Oracle) per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.73	57	0
	AES192	203	5	0.82	56	0
	AES256	206	4	0.68	58	0
	COLUMN	205	5	0.78	55	0
	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	13	1.02	58	5
	AES128	202	15	1.12	57	13
	AES192	203	15	1.09	56	16
	AES256	206	16	1.11	58	17
	COLUMN	205	12	1.04	55	14
	3DES	202	16	1.12	57	21
Dopo	NO_ENC	206	5	0.78	58	0
	AES128	202	4	0.68	57	0
	AES192	203	4	0.73	56	0
	AES256	206	5	0.78	58	1
	COLUMN	205	5	0.82	55	1
	3DES	202	4	0.73	57	0
Tabella 100k tuple						
Prima	NO_ENC	206	5	0.82	58	1
	AES128	202	4	0.68	57	1
	AES192	203	5	0.82	56	1
	AES256	206	4	0.78	58	0
	COLUMN	205	5	0.68	55	0

	3DES	202	4	0.73	57	0
Durante	NO_ENC	206	29	1.34	58	21
	AES128	202	56	1.54	57	34
	AES192	203	58	1.56	56	38
	AES256	206	55	1.48	58	41
	COLUMN	205	49	1.32	55	29
	3DES	202	61	1.67	57	42
Dopo	NO_ENC	206	4	0.73	58	0
	AES128	202	4	0.68	57	0
	AES192	203	5	0.78	56	0
	AES256	206	5	0.82	58	0
	COLUMN	205	4	0.73	55	0
	3DES	202	5	0.78	57	0

Tabella D.33: CryptoDatabaseAdapter - Stato sistema per la INSERT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	208	5	0.71	82	0
	AES128	206	20	1.61	80	0
	AES192	200	4	0.71	72	0
	AES256	199	4	0.76	73	0
	AES128COL	198	5	0.68	76	0
	AES192COL	200	4	0.75	73	0
	AES256COL	200	4	0.79	72	0
Durante	NO_ENC	209	13	1.53	86	0
	AES128	207	15	1.91	83	0
	AES192	201	11	1.23	75	0
	AES256	200	13	1.23	76	0
	AES128COL	199	14	1.12	77	0
	AES192COL	201	14	1.12	75	0
	AES256COL	201	18	1.23	74	0
Dopo	NO_ENC	208	7	0.86	85	0
	AES128	206	19	1.26	80	0
	AES192	200	4	0.62	72	0
	AES256	199	4	0.70	73	0
	AES128COL	198	4	0.59	75	0
	AES192COL	200	5	0.76	73	0
	AES256COL	200	4	0.69	72	0
Tabella 100k tuple						
Prima	NO_ENC	208	6	0.77	82	0
	AES128	206	17	1.29	80	0
	AES192	200	5	0.75	72	0
	AES256	199	4	0.56	73	0
	AES128COL	198	4	0.55	76	0
	AES192COL	200	5	0.66	73	0
	AES256COL	200	5	0.69	72	0
Durante	NO_ENC	209	24	1.64	87	0
	AES128	207	43	2.01	82	0
	AES192	201	32	1.67	75	0
	AES256	200	30	1.77	76	0
	AES128COL	199	32	1.67	78	0
	AES192COL	201	42	1.89	78	0
	AES256COL	201	40	1.78	76	0
Dopo	NO_ENC	208	7	0.82	85	0

	AES128	206	19	0.94	76	0
	AES192	200	6	0.72	71	0
	AES256	199	5	0.80	73	0
	AES128COL	198	4	0.74	4	0
	AES192COL	200	4	0.72	73	0
	AES256COL	200	4	0.68	72	0

Tabella D.34: CryptoDatabaseAdapter - Stato sistema per la SELECT.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	204	6	0.88	85	0
	AES128	202	9	0.83	72	0
	AES192	200	4	0.51	71	0
	AES256	200	4	0.59	75	0
	AES128COL	197	7	0.95	74	0
	AES192COL	198	4	0.64	72	0
	AES256COL	197	5	0.78	70	0
Durante	NO_ENC	205	8	0.97	86	0
	AES128	203	14	1.12	75	0
	AES192	201	5	0.78	73	0
	AES256	201	24	1.23	77	0
	AES128COL	198	15	1.34	78	0
	AES192COL	199	12	1.05	73	0
	AES256COL	198	18	1.34	72	0
Dopo	NO_ENC	204	4	0.78	85	0
	AES128	202	6	0.83	72	0
	AES192	200	4	0.74	71	0
	AES256	200	4	0.80	75	0
	AES128COL	197	7	0.87	74	0
	AES192COL	198	4	0.75	72	0
	AES256COL	197	4	0.77	70	0
Tabella 100k tuple						
Prima	NO_ENC	204	4	0.79	85	0
	AES128	202	4	0.53	72	0
	AES192	200	5	0.78	71	0
	AES256	200	4	0.70	75	0
	AES128COL	197	4	0.68	74	0
	AES192COL	198	5	0.67	72	0
	AES256COL	197	4	0.78	70	0
Durante	NO_ENC	205	12	1.12	87	0
	AES128	203	30	1.67	73	0
	AES192	201	24	1.65	73	0
	AES256	201	41	2.12	87	0
	AES128COL	198	45	1.98	81	0
	AES192COL	199	34	1.54	81	0
	AES256COL	198	45	2.03	75	0
Dopo	NO_ENC	204	5	0.82	85	0
	AES128	202	4	0.69	72	0
	AES192	200	4	0.74	71	0
	AES256	200	5	0.76	74	0
	AES128COL	197	7	0.66	74	0
	AES192COL	198	4	0.67	72	0
	AES256COL	197	5	0.82	72	0

Tabella D.35: CryptoDatabaseAdapter - Stato sistema per la UPDATE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	202	4	0.77	81	0
	AES128	200	4	0.68	72	0
	AES192	200	4	0.78	71	0
	AES256	203	4	0.75	73	0
	AES128COL	200	6	0.48	75	0
	AES192COL	198	4	0.65	72	0
	AES256COL	201	5	0.56	72	0
Durante	NO_ENC	203	7	0.98	81	0
	AES128	201	15	1.23	72	0
	AES192	201	15	1.12	71	0
	AES256	204	12	1.23	73	0
	AES128COL	201	12	0.78	75	0
	AES192COL	199	6	0.88	72	0
	AES256COL	202	7	0.64	72	0
Dopo	NO_ENC	202	4	0.78	81	0
	AES128	200	3	0.52	72	0
	AES192	200	4	0.87	71	0
	AES256	203	4	0.75	73	0
	AES128COL	200	4	0.72	75	0
	AES192COL	198	5	0.65	72	0
	AES256COL	201	6	0.63	72	0
Tabella 100k tuple						
Prima	NO_ENC	202	4	0.77	81	0
	AES128	200	3	0.78	72	0
	AES192	200	4	0.87	71	0
	AES256	203	5	0.76	73	0
	AES128COL	200	4	0.81	75	0
	AES192COL	198	4	0.68	72	0
	AES256COL	201	4	0.68	72	
Durante	NO_ENC	203	10	1.12	81	0
	AES128	201	25	1.65	72	0
	AES192	201	25	1.54	71	0
	AES256	204	37	1.89	73	0
	AES128COL	201	19	1.34	75	0
	AES192COL	199	7	0.93	72	0
	AES256COL	202	7	0.72	72	0
Dopo	NO_ENC	202	5	0.68	81	0
	AES128	200	3	0.71	72	0
	AES192	200	6	0.64	71	0
	AES256	203	4	0.68	73	0
	AES128COL	200	4	0.77	75	0
	AES192COL	198	6	0.72	72	0
	AES256COL	201	5	0.63	72	0

Tabella D.36: CryptoDatabaseAdapter - Stato sistema per la DELETE.

		#Processi Attivi	CPU(%)	CPU(GHz)	RAM(%)	Disco(%)
Tabella 10k tuple						
Prima	NO_ENC	204	9	1.14	77	0
	AES128	199	5	0.70	72	0
	AES192	201	4	0.79	70	0
	AES256	201	4	0.51	75	0

	AES128COL	200	4	0.78	75	0
	AES192COL	199	4	0.55	73	0
	AES256COL	203	5	0.71	72	0
Durante	NO_ENC	205	12	1.32	77	0
	AES128	200	10	1.12	72	0
	AES192	202	9	1.02	70	0
	AES256	202	12	0.96	75	0
	AES128COL	201	12	1.12	75	0
	AES192COL	200	6	0.85	73	0
	AES256COL	204	7	0.86	72	0
Dopo	NO_ENC	204	9	1.19	77	0
	AES128	199	4	0.70	72	0
	AES192	201	4	0.75	70	0
	AES256	201	6	0.66	75	0
	AES128COL	200	4	0.62	75	0
	AES192COL	199	5	0.71	73	0
	AES256COL	203	4	0.78	72	0
Tabella 100k tuple						
Prima	NO_ENC	204	8	1.09	77	0
	AES128	199	5	0.70	72	0
	AES192	201	4	0.75	70	0
	AES256	201	4	0.58	75	0
	AES128COL	200	4	0.69	75	0
	AES192COL	199	4	0.55	73	0
	AES256COL	203	4	0.64	72	0
Durante	NO_ENC	205	35	1.89	77	0
	AES128	200	18	1.54	72	0
	AES192	202	21	1.65	70	0
	AES256	202	14	1.02	75	0
	AES128COL	201	15	1.34	75	0
	AES192COL	200	6	0.91	73	0
	AES256COL	204	7	0.1	72	0
Dopo	NO_ENC	204	7	1.07	77	0
	AES128	200	4	0.77	72	0
	AES192	201	4	0.78	70	0
	AES256	201	6	0.72	75	0
	AES128COL	200	5	0.83	75	0
	AES192COL	199	4	0.68	73	0
	AES256COL	203	5	0.68	72	0

Bibliografia

[1] L. Bouganim and Y. Guo, "Database encryption", Hal, September 2011, DOI 10.1007/978-1-4419-5906-5_677. https://hal.archives-ouvertes.fr/hal-00623915/document

[2] Strephonsays, "Differenza tra MAC e DAC", https://it.strephonsays.com/difference-between-dac-and-mac

[3] Ionos, "Descrzione RBAC", https://www.ionos.it/digitalguide/server/sicurezza/cose-il-role-based-access-control-rbac/

[4] ecomputernotes, "Struttura e componenti di un DBMS", https://ecomputernotes.com/fundamental/what-is-a-database/components-of-dbms

[5] Oracle, "Documentazione linguaggio PL/SQL", https://www.oracle.com/it/database/technologies/appdev/plsql.html

[6] Microsoft, "Documentazione delle funzioni Transact-SQL", https://docs.microsoft.com/it-it/sql/t-sql/language-reference?view=sql-server-ver15

[7] PostgreSQL, "Documentazione del linguaggio PL/pgSQL", https://www.postgresql.org/docs/current/plpgsql.html

[8] C. Fiandrino, "Descrizione della Query Optimization", http://claudiofiandrino.altervista.org/Master_degree/Database_management_system/query_optimization.pdf

[9] I. Basharat, F. Azam, and A. Muzaffar, "Database security and encryption: A survey study", International Journal of Computer Applications, vol. 47, June 2012. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.258.8172&rep=rep1&type=pdf

[10] A. Shulman, "Top Ten Database Security Threats", http://docshare01.docshare.tips/files/10048/100489754.pdf

[11] M. Turkanović, T. W. Družovec, and M. Hölbl, "Inference Attacks and Control on Database Structures", 2000, https://www.temjournal.com/content/41/01/temjournal4101.pdf

[12] J. Domingo-Ferrer, "Inference control in statistical databases", Encyclopedia of Database Systems (L. LIU and M. T. ÖZSU, eds.), Boston, MA, 2009, pp. 1472–1476, DOI 10.1007/978-0-387-39940-9_203

[13] E. Shmueli, R. Vaisenberg, and E. Gudes, "Implementing a database encryption solution, design and implementation issues", computers & Security, vol. 44, July 2014, pp. 33–50, DOI 10.1016/j.cose.2014.03.011

[14] F. Moulianitakis and K. Asimakopoulos, "Bench-marking Framework for Transparent Data Encryption Systems", https://www.diva-portal.org/smash/get/diva2:1347966/FULLTEXT01.pdf

[15] Ryadel, "Data Encryption", https://www.ryadel.com/data-encryption-in-transit-at-rest-definizioni-best-practices-guida-tutorial/

[16] IETF, RFC 8846, https://datatracker.ietf.org/doc/html/rfc8446

[17] IETF, "RFC 4253", https://datatracker.ietf.org/doc/html/rfc4253

[18] Pentasecurity, "Database Encryption", https://www.pentasecurity.com/product/encryption/

[19] GDPR, "Articolo 33", https://gdpr-info.eu/art-33-gdpr/

[20] GDPR, "Articolo 34", https://gdpr-info.eu/art-34-gdpr/

[21] G. della privacy, "Definizione Data Breach", https://www.garanteprivacy.it/regolamentoue/databreach

[22] A.Deshmukh and R. Qureshi, "Transparent data encryption- solution for security of database contents", International Journal of Advanced Computer Science and Applications, vol. 2, March 2011. https://arxiv.org/ftp/arxiv/papers/1303/1303.0418.pdf

[23] Thales, "Database Encryption Approaches", https://cpl.thalesgroup.com/encryption/selecting-right-encryption-approach
[24] Oracle, "Advanced Security Guide", https://docs.oracle.com/en/database/oracle/oracle-database/21/asoag/index.html
[25] Microsoft, "Transparent Data Encryption", https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-ver15
[26] MySQL, "InnoDB Data-at-rest protection", https://dev.mysql.com/doc/refman/8.0/en/innodb-data-encryption.html
[27] MariaDB, "Data-at-Rest Encryption", https://mariadb.com/kb/en/securing-mariadb-data-at-rest-encryption/
[28] MyDiamo, "Types of Database Encryption methods", https://mydiamo.com/types-of-database-encryption-methods
[29] Oracle, "DBMS_CRYPTO", https://docs.oracle.com/en/database/oracle/oracle-database/21/arpls/DBMS_CRYPTO.html
[30] Microsoft, "Descrizione delle funzioni crittografiche (Transact-SQL)", https://docs.microsoft.com/en-us/sql/t-sql/functions/cryptographic-functions-transact-sql?view=sql-server-ver15
[31] PostgreSQL, "PGCRYPTO", https://www.postgresql.org/docs/current/pgcrypto.html
[32] MariaDB, "Descrizione delle Security Functions", https://mariadb.com/kb/en/encryption-hashing-and-compression-functions/
[33] MySQL, "Descrizione delle Security Functions", https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html
[34] R.Curtmola, J.Garay, S.Kamara, and R.Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions", Cryptology ePrint Archive, 2010. https://eprint.iacr.org/2006/210.pdf
[35] M. Ocenas, I. Homoliak, and P. Hanacek, "Security and encryption at modern databases", ICCSP 2020: Proceedings of the 2020 4th International Conference on Cryptography, Security and Privacy, January, 2020, pp. 19–23, DOI 10.1145/3377644.3377662
[36] Microsoft, "Always Encrypted", https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine?view=sql-server-ver15
[37] Solarwinds, "Types of Database Encryption Methods", https://www.solarwindsmsp.com/blog/types-database-encryption-methods
[38] Netlib, "Differences between whole database and column encryption", https://netlibsecurity.com/white-papers/differences-between-whole-database-and-column-encryption/
[39] T. Security, "The Definitive Guide To Encryption Key Management Fundamentals", https://info.townsendsecurity.com/definitive-guide-to-encryption-key-management-fundamentals
[40] P. S. S. Council, "PCI DSS", https://www.pcisecuritystandards.org/
[41] C. Group, "Descrizione di un ePHI", https://compliancy-group.com/hipaa-ephi-electronic-protected-health-information/
[42] H. Journal, "HIPAA and HITECH", https://www.hipaajournal.com/hipaa-and-hitech/
[43] SOX LAW, https://www.soxlaw.com/
[44] Cloud Security Alliance, https://cloudsecurityalliance.org/
[45] Security Guidance For Critical Areas of Focus In Cloud Computing, https://cloudsecurityalliance.org/artifacts/security-guidance-v4/
[46] GDPR, https://gdpr-info.eu/
[47] CAP 486, https://www.elegislation.gov.hk/hk/cap486
[48] APPI, https://www.cas.go.jp/jp/seisaku/hourei/data/APPI.pdf
[49] Privacy Amendment (Private Sector) Act 2000, https://www.legislation.gov.au/Details/C2004A00748
[50] NIST, "Descrizione del ciclo di vita delle chiavi crittografiche", https://csrc.nist.gov/CSRC/media/Events/Key-Management-Workshop-2001/documents/lifecycle-slides.pdf

[51] NIST, "Recommendation for Key Management- Part1", https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf
[52] NIST, "Descrizione del Cryptographic Module Validation Program", https://csrc.nist.gov/projects/cryptographic-module-validation-program
[53] NIST, "Recommendation for Key Management- Part2", https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt2r1.pdf
[54] F. Demaertelaere, "Slide di descrizione degli Hardware Security Module", https://web.archive.org/web/20150906093444/http://secappdev.org/handouts/2010/Filip%20Demaertelaere/HSM.pdf
[55] Advantio, "How does Hardware Security Module (HSM) protect Payment Card Data?", https://www.advantio.com/blog/hardware-security-module-hsm-what-is-it-and-what-is-its-role-in-protecting-payment-card-data
[56] ISO, "ISO 15408", https://www.iso.org/standard/50341.html
[57] NIST, "Derived Test Requirements for FIPS PUB 140-2", https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402DTR.pdf
[58] NIST, "Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program", https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf
[59] HashiCorp, "Sito web del Vault project", https://www.vaultproject.io/
[60] IETF, "RFC7292", https://datatracker.ietf.org/doc/html/rfc7292
[61] Wikipedia, "PKCS#12", https://en.wikipedia.org/wiki/PKCS_12
[62] OASIS, "Standard PKCS #11 Cryptographic Token Interface Base Specification Version 2.40", http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html
[63] OASIS, "Key Management Interoperability Protocol Specification Version 1.2", http://docs.oasis-open.org/kmip/spec/v1.2/os/kmip-spec-v1.2-os.html
[64] Oracle, "Descrizione CDB & PDB", https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/CDBs-and-PDBs.html
[65] O. D. I. Mechanism, "Descrizione dell'Index Lookup", https://databaseinternalmechanism.com/oracle-database-internals/index-lookup-unique-scanrange-scan-full-scan-fast-full-scan-skip-scan/
[66] Database and D. M. G. (PoliTO), "Slide relative all'accesso fisico ai dati", https://dbdmg.polito.it/wordpress/wp-content/uploads/2018/11/13-PhysicalAccess.pdf
[67] Oracle, "Descrizione dei tipi di dato supportati", https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/Data-Types.html#GUID-1A71C635-188E-4EC9-B821-1DBEC2B45451
[68] Microsoft, "Procedura su come usare la protezione dati", https://docs.microsoft.com/it-it/dotnet/standard/security/how-to-use-data-protection
[69] Microsoft, "SQL Server and Database Encryption Keys", https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/sql-server-and-database-encryption-keys-database-engine?view=sql-server-ver15
[70] Microsoft, "Extensible Key Management (EKM)", https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/extensible-key-management-ekm?view=sql-server-ver15
[71] Microsoft, "Descrizione dell'architettura delle CryptoAPI di sistema", https://docs.microsoft.com/en-us/windows/win32/seccrypto/cryptoapi-system-architecture
[72] Microsoft, "Scelta dell'algoritmo crittografico", https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/choose-an-encryption-algorithm?view=sql-server-ver15
[73] Microsoft, "Descrizione della funzione CREATE CERTIFICATE", https://docs.microsoft.com/en-us/sql/t-sql/statements/create-certificate-transact-sql?view=sql-server-ver15
[74] IETF, "RFC 5280", https://datatracker.ietf.org/doc/html/rfc5280
[75] Microsoft, "Descrizione della funzione CERTPRIVATEKEY", https://docs.microsoft.com/it-it/sql/t-sql/functions/certprivatekey-transact-sql?view=sql-server-ver15

[76] Microsoft, "Descrizione della CERTENCODED", https://docs.microsoft.com/it-it/sql/t-sql/functions/certencoded-transact-sql?view=sql-server-ver15
[77] MariaDB, "Compressione della pagina in InnoDB", https://mariadb.com/kb/en/innodb-page-compression/
[78] MariaDB, "Gestione delle chiavi crittografiche", https://mariadb.com/kb/en/encryption-key-management/
[79] Oracle, "Oracle Key Vault", https://docs.oracle.com/en/database/oracle/key-vault/
[80] Thales, "Vormetric Data Security Platform", https://cpl.thalesgroup.com/encryption/vormetric-data-security-platform
[81] Fornetix, "Vaultcore", https://www.fornetix.com/solutions/vaultcore/
[82] Amazon, "Amazon Web Services Key Management Service", https://aws.amazon.com/it/kms/features/
[83] MySQL, "FAQ relative alla InnoDB TDE", https://dev.mysql.com/doc/refman/8.0/en/faqs-tablespace-encryption.html
[84] Microsoft, "Descrizione del comando tasklist", https://docs.microsoft.com/it-it/windows-server/administration/windows-commands/tasklist
[85] A. A. Tamimi, "Performance Analysis of Data Encryption Algorithms", https://www.cs.wustl.edu/~jain/cse567-06/ftp/encryption_perf/index.html#2_5
[86] M.Abirami and S. Chellaganeshavalli, "Performance analysis of aes and blowfish encryption algorithm", International Journal of Innovative Research in Science, Engineering and Technology, 2013. http://www.ijirset.com/upload/2013/november/80_Nov13.pdf
[87] Oracle, "Documentazione della classe java.security.Keystore", https://docs.oracle.com/javase/8/docs/api/java/security/KeyStore.html
[88] Oracle, "Descrizione del metodo nanoTime", https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#nanoTime--
[89] Oracle, "Descrizione del metodo getProcessCPUTime", https://docs.oracle.com/javase/8/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html#getProcessCpuTime--
[90] Oracle, "Descrizione della classe System", https://docs.oracle.com/javase/8/docs/api/java/lang/System.html
[91] Oracle, "Descrizione dell'interfaccia OperatingSystemMXBean", https://docs.oracle.com/javase/8/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html
[92] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers, "Format-preserving encryption", Selected Areas in Cryptography (M. J. Jacobson, V. Rijmen, and R. Safavi-Naini, eds.), Berlin, Heidelberg, 2009, pp. 295–312, DOI 10.1007/978-3-642-05445-7_19
[93] Oracle, "Specifiche del file JAR", https://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html
[94] Git, https://git-scm.com/
[95] Microsoft, "Scaricare Microsoft JDBC Driver per SQL Server", https://docs.microsoft.com/it-it/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver15
[96] A. Commons, "Apache Commons Lang", https://commons.apache.org/proper/commons-lang/
[97] A. Commons, "Descrizione della classe ImmutablePair", https://commons.apache.org/proper/commons-lang/javadocs/api-3.1/org/apache/commons/lang3/tuple/ImmutablePair.html
[98] Oracle, "Descrizione della classe java.sql.DriverManager", https://docs.oracle.com/javase/8/docs/api/java/sql/DriverManager.html
[99] Oracle, "Documentazione dell'interfaccia java.sql.Connection", https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html
[100] Oracle, "Documentazione dell'interfaccia java.sql.PreparedStatement", https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html
[101] Oracle, "Descrizione del metodo addBatch", https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html#addBatch()
[102] IETF, "RFC 5116", https://datatracker.ietf.org/doc/html/rfc5116

[103] cryptography.io, “Descrizione delle Key derivation function”, https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/
[104] Nist, “Descrizione Nonce”, https://csrc.nist.gov/glossary/term/nonce
[105] Dzone, “Descrizione del Builder Pattern in Java”, https://dzone.com/articles/the-builder-pattern-for-class-with-many-constructo