Data Security and Performance with Cipher-Base

Er. Seema Rani,
Assistant Professor
Computer Science and Engineering,
CDLSIET, Sirsa, India.

Abstract: Data confidentiality is the main concerns for all individuals and companies worldwide. Over the past ten years, there has been an exponential increase in the number of attackers working both independently and in organized organizations due to the advancement of technology and the ease with which they may obtain attacking tools and algorithms online. The primary goal of these attackers is to obtain financial benefit by stealing information that impacts millions of individuals and businesses, frequently resulting in significant financial damages. Regardless of how simple or complicated the problem, today's world demands the use of optimal and intelligent problem solving techniques in every industry. Cipher-base is a full-featured relational database system that processes and stores encrypted data using cutting-edge, specially designed hardware. In addition to outlining the range of physical design options available to Cipher-base, This article demonstrates how developers can specify the encryption method for storing static data and the amount of information leakage that is acceptable for processing data during runtime, in order to meet their data confidentiality needs. The goal is to provide a physically designed database that maximizes performance while meeting the application's requirements for secrecy. Thus, this naive approach investigates techniques that can be applied to fortify ciphers and guarantee data security by offering a user-friendly interface that is simple to implement.

IndexTerms- Cipher-base, Data Security, Performance and Encryption.

I. Introduction

A primary worry of user's contemporary information systems is data confidentiality. For consumers of public cloud services, methods for safeguarding data from inquisitive attackers are especially crucial [1]. If a biologist, for example, does years-long in-vitro research and saves the data for future analysis in a publicly accessible cloud database service [2]. She wants to ensure that her rival cannot obtain her results before she can publish them. But in private clouds, where competitors may buy off database administrators to steal confidential data, data privacy is much more important. An add-on for Microsoft SQL Server called Cipher-base was created expressly to assist businesses in utilizing a productive "database-as-a-service" while defending their data from "honest-but-curious" opponents. To be more precise, Cipher-base can protect data against root-level access privilege administrators on the computers that host database server processes. To do tasks like building indexes, repartitioning data, updating system security, and other tasks, administrators require certain access levels. These administrators do not, however, require access to or interpretation of the information kept in the machines' databases. With the help of specialized hardware (based on FPGAs) and clever software/hardware co-design, Cipher-base securely decrypts, processes, and re-encrypts data without allowing outside parties with system access to sniff the associated plaintext [3]. Cipher-base's support for adjustable security is one of its unique features. All information kept in DB is not private. Even if some information, such vendor addresses, may be private, a rudimentary encryption method might be adequate. Data leaking of that kind is embarrassing but not catastrophic. Strong encryption is required for other data, including personally identifiable information (PII) about customers. With Cipher-base, functionality of a DB is preserved but the confidentiality requirements of all data may be declaratively specified.

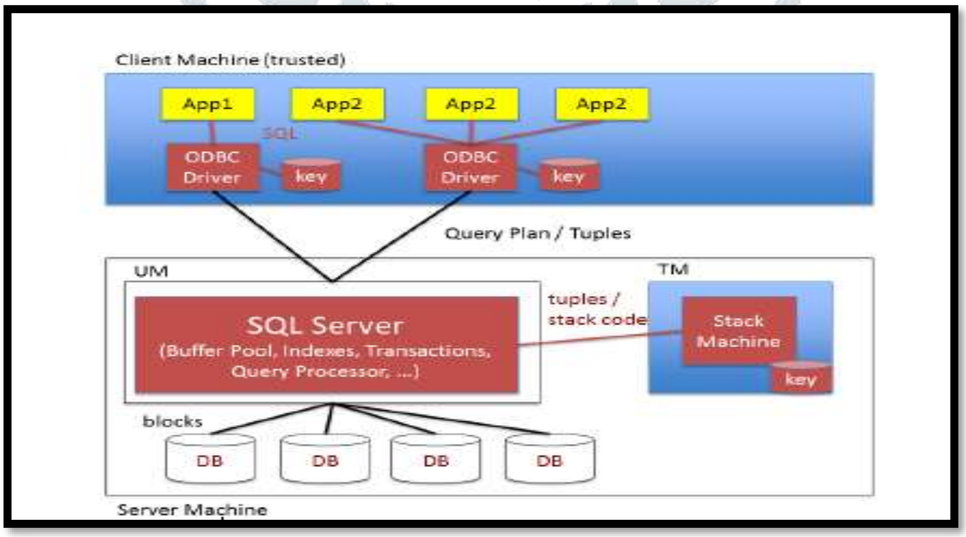

Figure 1: Cipher-base Architecture

Figure 1 depicts the architecture of Cipher-base. To put it briefly, Cipher-base enhances every significant feature of Microsoft SQL Server because all encryption and decryption are visible and programs just view clear text values, this technique saves programs from the need to be altered. The ODBC driver has an extension that allows it to store encryption keys. Highly encrypted data is processed by the SQL Server on using the trusted machine (TM). The majority of the SQL Server code remains intact and is still operational on commodity servers (such x86 processors), denoted in Figure 1 as UM for untrusted machine. TM can process encrypted data without disclosing the clear text, making it similar to a "oracle". Due to its implementation, the TM is able to evaluate any conditions on encrypted data. (Figure 1). Plaintext that corresponds to encrypted data is hidden in the UM since the encryption keys are likewise safely held in the TM. By leveraging the reliable SQL Server code base, any kind of SQL query or change may be supported by cipher-base while preserving transactional semantics. Consequently, in order to maintain confidentiality, we are able to perform fine-grained operations on encrypted data in the trusted computer. This paper's goal is to provide information on the data security and performance with the cipher base architecture. The research paper is organized as, First section explain introduction. Second section provides related work. Third section describes static data security, Fourth section explain run time security. Section V defines other tuning functions. Last section concludes the paper.

II RELATED WORK

Working with encrypted data in the cloud is necessary to provide an effective database-as-a-service. There exist encryption methods that allow performing specific DB operations on encrypted data, even though generic holomorphic encryption techniques that allow unrestricted computing on encrypted data are not yet feasible [4]. One way to find out if two columns are encrypted using deterministic encryption is to evaluate the join on the corresponding encrypted columns.

These partial holomorphic characteristics of encryption schemes are used by recent system called Crypt DB. There are a number of serious disadvantages to this tactic. Firstly, Crypt DB does not support generic or ad hoc SQL queries. In order to assess the aggregate and reduce the cost-benefit trade-offs related to using the cloud, each query that computes such aggregates across a database must submit the table to a trusted proxy for data encryption. Secondly, the security cannot be altered, not even for the types of operations that Crypt DB permits. For example, Crypt DB would reorder the column to be saved using order-preserving encryption if a query needed it to be sorted [5].

Trusted DB's loosely connected architecture consists of an IBM secure co-processor (SCP) and a commodity server, on which Trusted DB runs a MySQL database with enhanced features and trivial SQLite DB, is fundamentally different from the fine-grained integration of the UM/TM [6]. Cipher-base's fine-grained integration allows for new optimizations and a reduced footprint for the reliable hardware module. For example, the loosely connected technique is confined to use the few computational resources at hand if all columns were encrypted.

As opposed to this, Cipher-base uses a closely connected architecture, factoring and implementing just the essential primitives from each database system module that must function on encrypted data in TM. It's interesting to note that query processing, concurrency control, and other database operations can be obtained with a small class of primitives that involve encryption, decryption, and expression evaluation [7]. The hash join operator in the hash join example computes hashes and verifies equivalency using the TM. Otherwise, memory management, writing hash buckets to disk, and reloading them all are all done in the UM as part of the remaining hash join logic.

III STATIC DATA SECURITY

Cipher-base allows customers to fine-tune their physical architecture for optimal performance and supports a wide range of encryption mechanisms to satisfy a multitude of privacy requirements. Every piece of data should be strongly and probabilistically encrypted (e.g., using AES in CBC mode) if privacy were the main consideration. In many cases the performance would be subpar and the overhead would be significantly higher than what is required to meet the confidentiality requirements (some data, as previously mentioned, need not be encrypted because it is public). The following encryption methods are presently being incorporated into Cipher-base:

• **AES in ECB mode:** Deterministic encryption with AES in ECB mode enables equality predicates to be processed on encrypted data without the need to decrypt it. In other words, all equality predicates are fully processed in the UM without requiring data to be shipped to or processed in the Trusted Machine.

• **ROP:** This order-preserving encryption method enables processing of Top N operations and range sorting completely within the UM. However, all arithmetic operations, including aggregation, and SQL intrinsic functions must be performed within the TM[4].

• **Holomorphic**: Workable encryption methods are known for specific arithmetic operators (such as addition and multiplication). Even with generic holomorphic encryption, which allows for addition as well as multiplication, it is still impractical.

• **AES in CBC mode:** A probabilistic encryption method is AES in CBC mode. Any processing that is done on this data must be done within the TM.

Our methodology is predicated on the realization that an application's requirements for data secrecy are constant and independent of the workload associated with queries and updates. For example, compliance standards may dictate that certain data—like patient diseases in a health care domain—be encrypted robustly while other data—like patient cities—should not be encrypted at all. The application's execution has no effect on these statically known restrictions. Therefore, as part of the SQL DDL, the schema should be annotated with the data confidentiality requirements. Crypt DB design, a related database system for maintaining sensitive data, differs from this one in that it adapts the encryption strategy dynamically to the query demand[8]. This design specifies confidentiality statically. As mentioned in Section 2, As a result of running a query where the column is now strongly encrypted and which requires the usage of an order-preserving encryption technique, Crypt DB will degrade the encryption strategy. This architecture may cause unintentional information leakage (degrading the encryption) and performance jitter (rearranging an entire table while performing, say, a very basic range query).

A basic schema for patient and disease data is shown in Figure 2. Every piece of patient data is regarded as private. Since the patient names are extremely private, Figure 2's design requires that they be encrypted with AES. In the Patient database, names are unique. So deterministic encryption is adequate. Patient age is less important, particularly if it isn't associated with a name. Therefore, a weaker, order-preserving method like ROP can be used to encrypt patient age. Information about diseases is available to the public, but diagnosis details are extremely private since they belong to the sufferers. Figure 2 illustrates how any type of constraint can be applied. However, the efficiency of upholding integrity restrictions is affected by the encryption technology selected, just like in the case of standard query processing. Because an order-preserving encryption mechanism was adopted, the check constraint of Patient age in the schema of Figure 2 is checked by Untrusted Machine without requiring the information to be decrypted whenever a new patient is changed. The TM would need to verify the integrity constraint's predicate if the age was encrypted using AES. There are several possible ways to test for referential integrity and encrypt primary and foreign keys. Two scenarios are shown in Figure 2.

```
Create table Patient
(name : VARCHAR(50) AES_ECB primary key,
age : VARCHAR(50) ROP check >= 0);

create table Disease (
name : VARCHAR(50) primary key,
descr : VARCHAR(256));

create table Diagnosis (
patient : VARCHAR(50) AES_CBC references Patient,
disease : VARCHAR(50) AES_ECB references Disease,
date : DATE check not null);
```

Figure 2: Identifying Confidentiality Needs

Diagnosis patient may have been created to conceal from possible attackers the fact that certain patients are ill more frequently than others. It costs money to do this: The TM is required to do the predicate evaluation of each Patient ◂ ▸ Disease. The full join can be finished in the UM without requiring the join keys to be decoded if the key and the foreign key are encrypted using the same deterministic technique (such as AES in ECB mode). Therefore, application developers should exercise caution when selecting the encryption methods for foreign keys and should only deviate from the primary key's encryption method when absolutely required.

Figure 2's second situation deals with the foreign key / key interaction between Diagnosis disease and Disease name. In this instance, the referenced main key is not encrypted whereas the foreign key is. Once more, the idea behind encrypting the foreign key in this case may be to prevent any potential adversary from learning about the existence of specific diseases. Once more, the fact that multiple encryption algorithms are used for the join keys leads to costly Diagnosis Patient joins because the TM must assess the join predicate in order to decrypt Diagnosis disease. Strangely, in this case, encrypting Disease.name with AES in ECB mode yields better results in terms of performance. That is to say, even though the data is public and does not require protection, it could still be advantageous to encrypt it for performance-related reasons. Foreign keys and primary keys can be encrypted in a variety of ways with varying effects on performance.

IV RUNTIME DATA SECURITY

The performance implications of runtime data security are covered in this section. In order to drive this conversation, consider the following straightforward example, which shows how data processing can leak information even when the UM never sees any data that hasn't been encrypted.

Table 1: Patient diagnosis table

Patient	Disease
Alice	Asthma
Bob	Diabetes
Chen	Cancer
Dana	AIDS

A table of diagnoses is shown above. While the date column is heavily encrypted, it is presumed that the patient column is not. In other words, patient names are available to the public, but the specific diseases that each patient has been diagnosed with should remain private. Now imagine a question that inquires about the total number of individuals who have received a diagnosis of a specific illness (AIDS, for example). The disease field is encrypted, thus the TM must perform the predicate of this query. However, a system administrator who can watch the query execution can learn information from a naïve execution of this question where the TM checks each tuple separately. Each encrypted tuple is provided to the TM when the filter operator is called naively. The TM decrypts the tuples, assesses the predicate, and returns either true or the original tuple, which evidently contains the patient's name. The TM receives each encrypted tuple when the filter operator is called naively. The patient name appears in clear text in the initial tuple, which is returned either true or false by the TM after decrypting the tuples and evaluating the predicate. Consequently, a system administrator who is able to monitor the data transmission between the TM and UM can:
1) Deduce that the many patients that the filter operator outputs are all suffering from the same illness.
2) If the filter operator knows further background information indicating the illness in issue is AIDS, deduce the patient's illness from their output. Observe that this is true even though the column is never decrypted outside of the TM and is stored in a highly encrypted state.

The main finding is that some types of information are leaked by query processing algorithms and procedures. Using a similar annotation method to static data security, Cipher-base enables application developers to identify the types of data that could be exposed during query processing. For example, the creator of the Diagnosis table's schema could stipulate that the disease column cannot have any dynamic information leakage. Consequently, Cipher-base would assess a query asking for the total number of AIDS patients using a suitable algorithm.

The runtime data security settings supported by Cipher-base are listed in table 2. The range of query processing methods is more limited and more work must be done in the TM the greater the secrecy constraints. Stated differently, under many circumstances, performance suffers in proportion to the degree of confidentiality required. We give a quick rundown of the different levels and the performance issues they raise in remaining part of the section.

Table 2: Confidentiality Levels for Runtime Data Security

Dynamic Leakage	*Computation in TM*
Default	Expressions
Card	Expressions, encrypt output
Blob	Whole operators, process blocks

Default: The data on disk is protected by this default degree of runtime security, which also shields it from attackers who can access the database server machine's main memory. Because of this, this level necessitates the encryption of all data stored in main memory, with the possibility of only decrypting data as needed (in the TM) to execute queries. This level permits the evaluation of predicates of tuples using the naïve approach, but in order to do, say, predictions or aggregation on encrypted data, trustworthy hardware (i.e., TM) must be used.

Cardinality: The only information released at this time is the cardinality of the intermediate and final query results, which contain the sensitive column. As a result, if a query requests for all patients who have received an AIDS diagnosis, an attacker may find out the overall number of patients who have received a diagnosis but could not infer the names of specific individuals. To accomplish this level, cipher-base postpones the evaluation of a predicate for a tuple [9]. In Example 1, for instance, if Alice had been diagnosed with AIDS and the TM was asked to process her, she would not immediately be returned as a match. Rather, the TM would first record all matching tuples before beginning to return matching tuples with all columns—including any fields that contain plain text—encrypted. Additionally, the returned tuples' order is determined at random. For example, analyzing the TM tuple corresponding to Dana could give Alice (in encrypted form).

Blob: This stage is equivalent to blob-formatting the column for storage. To reach this level, cipher-base needs to implement every operation in the relational algebra. To execute queries, the TM receives encrypted tuples. The TM partitions these tuples for a hash join and other bulk operations after decrypting these blocks. Additionally, the TM only returns encrypted tuple blocks, possibly including the empty block of tuples, meaning that its data cannot be used to infer any information. Because in order to prevent the cardinality of the filter operator, even in Example 1, it must generate a constant number of blocks as output, regardless of the predicate's selectivity from being found, this option should only be used for columns that require high confidentiality.

V OTHER TUNING OPTIONS

As mentioned in the earlier sections, important choices concerning the physical architecture of DB such as Cipher-base are which encryption technique to use and how much information is leaked. Additional factors for a strong physical design in Cipher-base are covered in this section.

Clustering horizontally: Certain schemas have a lot of flags (like is vegetarian) or are short fields (like status) that can be represented with a few bits. It would be inefficient to encrypt each of these fields separately. For instance, encrypting a flag (1 bit) with AES (128 bits) can result in a two-fold increase in database size. Group and encrypt the properties that make up a row in order to reduce this overhead space without sacrificing confidentiality.

Vertical clustering: Values from the same column are clustered vertically as an alternative to the encryption of multiple fields within a row and the horizontal clustering of data across many rows. To create a single cipher, for example, the status of sixteen records may be aggregated and encrypted. This method is similar to the PAX storage configuration that suggests [10]. Vertical clustering is useful for tables with a single tiny column, but integrating it into a query processor that isn't already built on a PAX storage structure will be more challenging.

Indexing and Materialized Views: Using cipher-base, define materialized views and indexes. These materialized views and indexes consider the limitations on dynamic information leaking and inherit the encryption method of the referenced data..

Setting Isolation Levels: In any database system, the isolation level is a crucial tuning parameter. Cipher-base is likewise subject to this observation. However, it is especially significant for Cipher-base since it affects which operations must be performed in the TM and which can be performed in the UM. To verify predicates on encrypted data for key range locking through serializability, for instance, the lock manager needs to establish a connection with the TM. All of the lock manager's activities can be finished inside the UM by using lesser isolation levels, like snapshot isolation.

Statistics: In general, information leakage can occur from the existence and upkeep of statistics. However, the way Cipher-base is currently designed, this kind of information leakage is prevented. The server stores all statistics in an encrypted format, while the (trusted) client computers perform query optimization, which calls for this information in clear text.

VI CONCLUSION

The Cipher-base system's objective is to assist developers in maintaining high speed while shielding sensitive data from snooping adversaries (such database administrators). To achieve high performance, the physical database design is essential. Most important physical design chooses a unique system like Cipher-base were covered in this study. This study specifically covered the performance implications of runtime data security, static data security (the choice of encryption technique), and other tuning factors like clustering rows and columns. As we prototype cipher-base, we are in the process of assessing as well as measuring the trade-offs associated with utilizing several physical design alternatives described here. When achieving the necessary secrecy at the lowest feasible cost is the main objective, we think cipher-base is especially well-suited to serve as the foundation of a secure database-as-a-service.

REFERENCES

[1] Hacigumus, H. Mehrotra, S and Iyer, B R. 2002. Providing database as a service. ICDE. 29–38.

[2] Microsoft Corporation. SQL Azure. http://www.windowsazure.com/en-us/home/features/sql-azure/.

[3] Eguro K. and Venkatesan. R. 2012. FPGAs for trusted cloud computing. FPL.

[4] C. Gentry. 2010. Computing arbitrary functions of encrypted data. Commun. ACM. 53(3).

[5] Boldyreva A. Chenette, N. Lee, Y. and O'Neill. A. 2009. Order-preserving symmetric encryption. In EUROCRYPT '09.

[6] S. Bajaj and R. Sion. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In SIGMOD, 2011.

[7] Arasu, A. 2013.Orthogonal security with cipherbase. In CIDR.

[8] Popa, R. A. Redfield, C. M. S. and Zeldovich, N. 2011. Cryptdb: Protecting confidentiality with encrypted query Processing. In SOSP, pages 85–100, 2011.

[9] Ailamaki, A. DeWitt, D. J. Hill, M. D. and Skounakis, M. 2001. Weaving relations for cache performance. VLDB. pp. 169–180.

[10] Hildenbrand, S. Kossmann, D. Sanamrad, T. Binning, F. Faerber, C. and Woehler, J. 2011. Query processing on encrypted data in the cloud. In Technical Report No. 735. ETH Zurich.