International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 08 |Aug -2016 www.irjet.net p-ISSN: 2395-0072

CypherDB: An Encryption Technique for Secure Database Processing

Mr. Febin Baby[1], Mr. Arun R[2]

[1]MTech Cyber Security, Dept. of CSE, SNGCE, Kadayiruppu, Kerala, India
[2]Asst. Prof. MTech Cyber Security, Dept. of CSE, SNGCE, Kadayiruppu, Kerala, India.

Abstract – *Security models are the basic theoretical tools to start when developing a security system. This seems to be an issue which is insufficiently understood and it may be an explanation for the actual security crisis of information system. Most cooperation try to address there security problem by simply patching their system to eliminate identified vulnerabilities. In most cases, this is already too late and long term strategy. CypherDB addresses the problem of protecting the confidentiality of database stored externally in a cloud and enabling efficient computation over it to thwart any curious-but-honest cloud computing service provider. It works by encrypting the entire outsourced database and executing queries over the encrypted data.*

Key Words: Database Security, Cloud Security, Encryption, Decryption, Attribute Encryption Seed, Confientiality

1. INTRODUCTION

Organizations are becoming more concerned about data security, especially as the intrinsic value of our data continues to increase. However, database security often gets overlooked. Managing organizational assets such as data, as well as overall information security concerns, are two of the key technology areas having a large effect on companies today. Although it is often difficult to put an exact price tag on the data we store, we do know data is an extremely valuable asset, and the compromise and/or exposure of such information can cause significant damage to business and company reputation.

Database security strives to insure that only authenticated users perform authorized activities at authorized times. It includes the system, processes, and procedures that protect a database from unintended activity.

1.1 Database Security Strategy

If regulatory or contractual requirements are not enough reason to address database security as part of an overall security strategy, let's look at a few other key facts that encourage us to consider database security as a part of our security strategy. Databases are increasingly being targeted by attackers. Data plays an extremely important role in a typical organization's environment. Security has historically addressed keeping the external attackers out of networks and operating systems. More recently, the focus has been on security of applications. Organizations spend large amounts of resources adding firewalls, IDS, IPS, policies, operating systems controls, access controls and other security controls to end points and on the network. By doing so, organizations believe they are protected. However, without controls directly around the data, they have left open an opportunity for an internal attacker who is authorized to access and transfer data from the database.

1.2 Making Databases a Priority

Databases, by their nature, are complex. Many security professionals simply do not have the background to understand the risk and security issues related to various brands and versions of databases. This leaves security in the hands of DBAs, who spend less than five percent of their time on database security, according to a Forrester Research report. The report stated that many enterprise DBAs are unaware of which databases, tables, and columns contain sensitive data, either because these are legacy applications and/or because no documentation of the data models and their properties exists.

Even with full knowledge of database assets, databases are more difficult to protect uniformly because there are unique security implementation procedures for the databases themselves, as well as with the applications interacting with them. Forrester estimates that more than 90 percent of enterprises support more than one type of database in their environment [1]. Enterprises today have to support hundreds and thousands of production databases with various business applications running on them. Business applications interacting with the databases can pose significant risks as additional application layer vulnerabilities may be introduced.

1.3 Database Security Considerations

We know we need to address database security as part of our overall security strategy. So, the question becomes, what key areas should be addressed? The following areas are critical areas we discuss throughout the remainder of this paper:

- Access controls
- Encryption

- Auditing
- Separation of environments
- Secure configuration

2. BACKGROUND

The importance of security in database research has greatly increased over the years as most of critical functionality of the business and military enterprises became digitized. Database is an integral part of any information system and they often hold sensitive data. The security of the data depends on physical security, OS security and DBMS security. Database security can be compromised by obtaining sensitive data, changing data or degrading availability of the database. Over the last 30 years the information technology environment have gone through many changes of evolution and the database research community have tried to stay a step ahead of the upcoming threats to the database security. The database research community has thoughts about these issues long before they were address by the implementations. This paper will examine the different topics pertaining to database security and see the adaption of the research to the changing environment. Some short term database research trends will be ascertained at the conclusion.

2.1 Sequential Systems

Initial computer systems processing data were based on the pre-existing manual systems. These were sequential systems, where individual files were composed of records organized in some predetermined order. Although not a database because there was no one integrated data source, electronic file processing was the first step towards an electronic data based information system [2]. Processing required you to start at the first record and continue to the end. This was quite good for producing payroll and monthly accounts, but much less useful when trying to recall an individual record. As a result it was not uncommon to have the latest printout of the file which could be manually searched if required kept on hand until a new version was produced.

2.2 Hierarchical Databases

Although hierarchical databases are no longer common, it is worth spending some time on a discussion of them because IMS, a hierarchical database system is still one of IBM's highest revenue products and is still being actively developed [2]. It is however a mainframe software product and may owe some of its longevity to organizations being locked in to the product from the early days of mainstream computing. The host language for IMS is usually IBM's PL/1 but COBOL is also common. Like a networked databases, the structure of a hierarchical database relies on pointers.

2.3 Network Databases

In a network database such as UNIVAC's DMS 1100, you have one record which is the parent record. This is defined with a number of attributes (for example Customer ID, Name, Address). Linked to this are a number of child records, in this case orders which would also have a number of attributes. It is up to the database design to decide how they are linked. The default was often to 'next' pointers where the parent pointed to the first child, the first child had a pointer to the second child and so on. The final child would have a pointer back to the parent. If faster access was required, 'prior' pointers could be defined allowing navigation in a forward and backward direction. Finally if even more flexibility was required 'direct' pointers could be defined which pointed directly from a child record back to the parent. The trade-off was between speed of access and speed of updates, particularly when inserting new child records and deleting records. In these cases pointers had to be updated.

2.4 Relational Databases

They arose out of Edgar Codd's 1970 paper 'A Relational Model of Data for Large Shared Data Banks" (Codd 1970) [2]. What became Oracle Corporation used this as the basis of what became the biggest corporate relational database management system. It was also designed to be platform independent, so it didn't matter what hardware you were using. The basis of a relational system is a series of tables of records each with specific attributes linked by a series of joins. These joins are created using foreign keys which are attributes containing the same data as another tables primary key. A primary key is a unique identifier of a record in a table. This approach to data storage was very efficient in terms of the disk space used and the speed of access to records.

2.5 Object Oriented Databases

Most programming today is done in an object oriented language such as Java or C++. These introduce a rich environment where data and the procedures and functions need to manipulate it are stored together. Often a relational database is seen by object oriented programmers as a single persistent object on which a number of operations can be performed. However there are more and more reasons why this is becoming a narrow view.

One of the first issues confronting databases is the rise of non-character (alphanumeric) data. Increasingly images, sound files, maps and video need to be stored, manipulated and retrieved. Even traditional data is being looked at in other ways than by traditional table joins. Object oriented structures such as hierarchies, aggregation and pointers are being introduced. This has led to a

number of innovations, but also to fragmentation of standards.

3. PROPOSED SYSTEM

In this section, we discuss how the database owner encrypts the database before outsourcing it to the Cloud. The database is encrypted in logical level to allow scalable and parallel query processing. The choice of encryption scheme is to allow efficient processing in our CypherDB secure processor. The entire database is protected by encrypting each field of a database table. We refer a field and a row of a database table as attribute and record respectively in the rest of the paper.

3.1 Attribute Encryption

Each attribute is encrypted with AES in one of two modes: 1) counter mode (AES-CTR) or, 2) output feedback mode (AES-OFB). These two encryption modes are chosen due to two important objectives: 1) offloading the encryption/decryption work, and 2) transforming block cipher into stream cipher.

Algorithm 1 explains the operation of these two encryption modes in pseudo-code. AES-CTR encrypts any attribute less than or equal to 128 bits. The function Encctr encrypts the l-bit long attribute, denoted as a, using the 128-bit attribute seed s and database key Kdb. The encryption is done by XOR-ing the most significant "l" bits of the encryption pad with the data (line 3-5). To encrypt the attribute longer than 128 bits, AES-OFB function Encofb generates a series of 128-bit encryption pads to be encrypted with the data (line 14-16). The most significant "l" bits of the concatenated encryption pad p0,p1,...pm is then used to encrypt the attribute data (line 17-19). Decryption is done by XOR-ing the cipher text with the same encryption pad (line 6 and 20). Note that AES-CTR and AES-OFB both use AES encryption but in different operation modes (line 4 and line 15).

Algorithm 1 Pseudo-code of attribute encryption

1: /* CTR encryption of any attribute less than or equal to 128 bits in a tuple */
2: **function** Encctr(s; a; Kdb)
3: **for** i = 1,... , l **do**
4: p = AES(s; Kdb)
5: yi = ai⊕pi
6: /* Decryption: ai = yi⊕pi */
7: return (s; y)
8: **end for**
9: **end function**
10: /* OFB encryption of any attribute larger than 128 bits in a tuple */
11: **function** Encofb(s; a; Kdb)
12: p0 ← s
13: m ← [l/128]
14: **for** h = 1, ... , m **do**
15: ph = AES (ph-1; Kdb)
16: **end for**
17: p = p0, p1,...., pm
18: **for** i = 1,..., l **do**
19: yi = ai⊕ pi
20: /* Decryption: ai = yi⊕ pi */
21: **end for**
22: return (s; y)
23: **end function**

3.2 Attribute Encryption Seed

The challenge of encrypting the attribute with AES-CTR and AES-OFB is to maintain the uniqueness of the seed s under the same database key Kdb [3]. In other words, each attribute across a database must own a distinct seed to each other (spatial uniqueness), whereas the seed for the same attribute must not repeat for every update operation on that attribute (temporal uniqueness).

3.2.1 Seed Components

Due to these security and performance concerns, a logical schema of the database to formulate the seed. In the structure of a logical schema, if each element of the schema has its own identifier (ID), each attribute can be identified by (databaseID; tableID; rowID; columnID) which is spatially unique across various databases and tables. Temporal uniqueness can be achieved by appending a global incremental counter **cntr** to each record, which is shared by each attribute within that record.

3.2.2 Seed Formulation

In a typical database application, logical schema is used in most operating layers and is eventually translated into its physical schema in order to locate the record in the database file. The formation of the attribute seed can thus embedded into the logical-to-physical schema translation software process. In other words, the actual program execution is able to "generate" the encryption seeds by re-using some software execution parameters, at run-time.

3.3 Index Protection

Encrypting the attribute with AES in either CTR or OFB mode prohibits the B+-tree indexing, which is one of

the most commonly used indexing strategies in database system. To allow remote indexing and protect the indices at the same time, we adopt order-preserving encryption (OPE) to encrypt the indices in the same way as suggested in [4].

OPE is an encryption scheme that can perform order operations on ciphertexts in the same way as plaintexts (i.e. Enc(x) > Enc(y) iff x > y) and is well proven to reveal no additional information about the plaintext values beside their order [4].

4. SECURITY ANALYSIS

In this section, the performance of CypherDB on encrypted database is compared with that without encryption. The total execution cycles are measured in both cases and the resulting slowdown percentage is recorded. CypherDB incurs performance overhead in two places: 1) extra instruction executions when copying the database seeds into Regseed to perform data decryption/encryption; 2) extra memory accesses when fetching the counter value of the attribute seeds. Our discussion is focused on protecting confidentiality of the data because the goal of CypherDB is to prevent information leakage through passive attack.

The security of AES-CTR and AES-OFB is well-proven [5, 6], except that they pose a strong requirement on the encryption seed - must be unique for each datum under a single encryption key. Otherwise, the confidentiality of the data may be compromised due to the "two-time" pad attack caused by re-using the same encryption pad. In our proposed encryption scheme, each attribute seed is spatially and temporally unique across the databases for the same database owner. Various database owners have their own unique database encryption keys Kdb such that the seed uniqueness concern is confined to a single party. It therefore greatly simplifies the attribute seed management and is relied on the CypherDB supported software to handle the seed uniqueness. Re-encrypting the database with a new encryption key may be necessary when the attribute seeds, either the logical schema ID or tuple counters, overflows. These parameters are set to a sufficient large value to avoid frequent re-encryption. CypherDB employs three different encryption keys for various encryption purposes. The database encryption key is therefore isolated for the ease of maintaining the seed uniqueness. The use of these encryption modes however provides some security strength. It is because the encrypted data are non-deterministic due to the unique encryption seed used. It means that even two attributes are of the same value, the encrypted data looks completely different.

5. CONCLUSIONS

Recent research on security systems for various sizes of data groups focused on several requirements related to data size. However, it could not assure data confidentiality in databases. In addition, in defining data groups, overhead could occur, and adding the policy could also cause a decrease of performance efficiency and duplication of the policy. Moreover, integrated management would not be possible for various databases. Here presents a novel processor architectural design to perform secure and efficient query processing on an encrypted database. With minimal modifications to the database application software, our proposed processor architecture, CypherDB, can achieve a higher security and performance efficiency when compared with solutions using homomorphic encryption or trusted coprocessor. Our work is being extended in several directions. One interesting direction would be to incorporate our system into an In-Memory database environment, which potentially is more efficient in accessing data. Another direction relates to the use of vector processing in the modern processor systems.

REFERENCES

[1] Tanya Baccam, Making Database Security an IT Security Priority-A SANS Whitepaper – November 2009

[2] P. Lake, P. Crowther, *Concise Guide to Databases*, Undergraduate Topics in Computer Science, DOI 10.1007/978-1-4471-5601-7_2, © Springer-Verlag London 2013

[3] M. Dworkin, "Recommendation for block cipher modes of operation, National Institute of Standards and Technology, Tech. Rep., 2001.

[4] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in Proceedings of the 2013 IEEE Symposium on Security and Privacy, ser. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 463–477.

[5] T. G. Wayne Jansen, "Guidelines on security and privacy in public cloud computing," National Institute of Standards and Technology, Tech. Rep., Dec 2011.

[6] Bony H. K. Chen, Paul Y. S. Cheung, Peter Y. K. Cheung, and Yu-Kwong Kwok "CypherDB: A Novel Architecture for Outsourcing Secure Database Processing" DOI 10.1109/TCC.2015.2511730, IEEE Transactions on Cloud Computing