SECURING THE ENTERPRISE DATABASE

V Radha, Ved P Gulati
Institute for Development and Research in Banking Technology,Hyderabad, India

N Hemanth Kumar
Institute for Development and Research in Banking Technology,Hyderabad, India

Keywords: Database, Security, and Enterprise Security

Abstract: Security is gaining importance once computers became indispensable in every organization. As the new concepts like E-Governance in Government and E-Commerce in business circles etc are heading towards reality, security issues penetrated even into the legal framework of every country. Database security acts as the last line of defence to withstand insider attacks and attacks from outside even if all the security controls like perimeter, OS controls have been compromised. Data protection laws such as HIPAA (Health Insurance Portability and Accountability Act), Gramm-Leach-Bliley Act of 1999, Data protection Act, Sarbanes Oxleys Act are demanding for the privacy and integrity of the data to an extent that the critical information should be seen only by the authorized users which means the integrity of the database must be properly accommodated. Hence, we aim at providing an interface service in between enterprise applications and enterprise database that ensures the integrity of the data. This service acts as a security wrapper around any enterprise database.

1 INTRODUCTION

A single appliance or a device cannot achieve security. Hence, a layered security *solution* is to be followed which includes physical security, perimeter security, Access controls, and OS based controls and finally database security. Database security is as important as the other layers since actual data resides on the database. The sensitive data must be protected from viewing and modifying by the unauthorized people. Even the administrator should not have privileges to view/modify the database. However, the administrator has sweeping privileges. An attacker can obtain administrative privileges by launching buffer overflow attacks and gains full access rights not only on the tables but also on the procedures, triggers, events and the configuration settings.

In present enterprise application development, the database and application are not being integrated tightly. The developers treat them separate and develop the final application in a much-disintegrated fashion. Main reason for this is due to lack of standards. Except SQL, ODBC and JDBC standards, which allow the data to be interchanged/queried across any kind of database, there is no standard for triggers, procedures, access controls etc. To make the application portable across any database, the developers tend to use only standard features of the database and ignore the special security features like access controls, encryption etc offered by a commercial RDBMS. So ultimately, the database has become just a data store and developers depend heavily on OS security controls even for database security.

Section 2 deals with the problem description and the motivation behind this paper and brief account on the legal aspects. Section 3 gives a brief account on the research work in this area. Section 4 deals with the survey of products and their detail explanations. Section 5 deals with our three proposals, their architectures to ensure data integrity. Section 6 deals with the simulations and performance evaluations of first two solutions. Section 7 deals with the conclusion and future work that we want to implement.

Radha V., P Gulati V. and Hemanth Kumar N. (2005).
SECURING THE ENTERPRISE DATABASE.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 76-83
DOI: 10.5220/0002530600760083
Copyright © SciTePress

2 MOTIVATION

Tampering and injecting or deleting certain fields in the database by an administrator or a hacker with administrative privileges in spite of various security measures such as database access controls is a serious issue. In this communications world, where most of the transactions are done over the web/internet, there is a need for the applications to know whether the data they are retrieving from the database is authentic or not. In addition, a number of legislative and commercial initiatives are requiring increased attention to the privacy, confidentiality and authenticity/integrity of electronic stored data to safeguard non-public personal information (NPI) and other sensitive enterprise data. Information security requirements associated with these measures include:

HIPAA (Health Insurance Portability and Accountability Act) (Arup, 2004) introduced in United States to eliminate the problems concerning the Health care access and made mandatory for all healthcare companies to be HIPAA compliant by 2003. The main aim of HIPPA is to make patient information available to *authorized* users only and to maintain the Privacy of information.

The Gramm-Leach-Bliley Act (GLB Act, 2004), also known as the Financial Modernization Act of 1999, is a federal law enacted in the United States to control the ways that financial institutions deal with the private information of individuals. It stresses on the collection and disclosure of private financial information and says that the financial institutes must implement security programs to protect such information so that the private information is not accessed through false pretences.

Sarbanes-Oxley Act – Passed in 2002, places strict requirements on company Boards and Officers to proactively prevent mishandling of information (Sarbanes, 2004). The Sarbanes-Oxley Act has changed how companies must disclose information regarding the responsibilities of corporate directors, officers and reporting obligations. Public companies must comply with these regulations or face significant penalties.

The Data protection Act, 1998, came into effect in March 2000 in United Kingdom. Data protection act serves to protect people from their personal information being treated, and used in a harmful manner (DP Act, 2004). The main objective of Data Protection law is to ensure that the fundamental right to privacy is not infringed through the abuse of today's technology. This act says the personal data should be collected for specified lawful means and should be processed fairly and lawfully. Non-compliance with the data protection provisions may result in exposing the institution to civil and or criminal liability in addition to the related negative publicity.

3 RELATED RESEARCH WORK

R Graubart in his paper on "The Integrity-Lock Approach to Secure Database Management" (Richard, 1984) has proposed the concept of using checksums at record level and field level for integrity purpose. In this architecture, a trusted front end is introduced between the user/client and the untrusted DBMS for the verification of checksums. The paper also analysed the advantages and disadvantages of this approach when checksums are used at record level and at field level.

E Mykletun and M Narasimha proposed a new scheme using Merkley's Hash Trees for Integrity and Authentication in Outsourced Databases (Mykleuton, 2003a). Here a Hash Tree is constructed and stored at the database in addition to the records. All the records are placed at the leaves of the tree, the interior nodes are the hashes of the data at sons of that node, and the owner of data signs the root node. Whenever the client queries the database, all the relevant records and the necessary hashes unto the root are sent to the client. The client verifies the signature of the root and reconstructs the tree using the data sent to it and checks all the hashes of the Hash Tree. This approach solves the problem of *completeness of query replies* in addition to data integrity problem.

In a later paper by the same authors, they proposed two new schemes using Condensed RSA and BGLS signatures (Mykleuton, 2003b). The first scheme is for single owner, multiple querier models, and second one for multi owner and multi querier model. Both the schemes concentrated on reducing the amount of extra information to be transferred from database sever to client for verification of integrity of data.

Another recent approach by C N Zhang, proposed an integrated approach for integrity of database and Fault Tolerance (Chang, 2004). This approach utilizes the redundant residue number systems and Chinese remainder Theorem for checksum generation and verification. This approach also detects and corrects a single error in the data. However, this approach requires finding n number of big relatively primes where n is the number of fields in the record and also the approach requires lot of security analysis to be done.

4 SURVEY OF EXISTING SOLUTIONS

Already many commercial products are available to ensure database security. We studied the following products:

4.1 DBMS_OBFUSCATION Toolkit

This toolkit is a built-in package delivered with Oracle (Arup, 2004 & AppSecInc, 2004). This is an Oracle crypto package, which provides functions that provide raw encryption and decryption capability. The encryption and *hashing* algorithms provided in the package are DES, 3DES, MD5. The disadvantages of this toolkit are 1.The encryption, decryption and *hashing* functions can be used only for a few data types. 2. As it is a built-in package, it works only for Oracle databases. 3. There is no Application transparency.

4.2 Db_Encrypt

Application Security Incorporation designed this product for protection of data. DbEncrypt (DbEncrypt, 2004 & AppSecInc, 2004) focuses on encryption of data at column level so that attackers cannot view the sensitive data in plain text. It provides a vast number of industry standard encryption algorithms offering strong encryption and increased performance. It also provides a point and click GUI interface where the table owner can specify the columns to be encrypted or *hash*ed, the encryption algorithm, the key size and the users for whom the columns to be decrypted. DbEncrypt itself takes care of Key Management.

DbEncrypt provides data integrity by providing signing and verifying functions to sign data at rest with a digital signature. The advantages of this product are: 1.DbEncrypt itself takes care of Key Management. 2. It provides application transparency. 3. It is a HIPAA compliant product. As DbEncrypt utilizes triggers and procedures internally, it is database dependent and presently, the product is available for Oracle and MS Sql Server.

4.3 XP_CRYPT

Active Crypt developed this product. It supports encryption algorithms like AES, DES, RC4 and *hashing* algorithms like MD5, SHA1 and supports ORACLE and SQL Server databases. XP_Crypt (XP_Crypt, 2004) is compatible with OpenSSL library and hence users can develop their own 3rd party application that can work with encrypted data. This product provides data integrity through *hashing* algorithms and RSA. However, the key management is complex, as it does not provide any GUI.

4.4 Nshield

This product is developed by nCipher, which is Public Key Cryptographic Standard #11 (PKCS #11) compliant device that generates and store certificate authority (CA) keys. The integration of nCipher’s encryption–based technology with RSA Keon security infrastructure allows organizations to protect electronic assets and intellectual property, as well as take steps to ensure the integrity and privacy of customer information. NShield (nCipher, 2004) provides efficient key management through the use of the centralized server that separates the encryption keys from the encrypted data. This centralized software is built with RSA BSAFE encryption software, which provides encryption services to all enterprise databases and to different applications.

5 PROPOSED SOLUTIONS

The products mentioned above concentrated much on data confidentiality rather than on data integrity. The emphasis of the problem proposed is on data authenticity/integrity. There is no single appliance as of now that solves the data integrity problem that is a just plug-in product applicable for any database and any operating system. The need of the hour is to design a system that is application transparent, and database, operating system independent.

Our solution aims at empowering the applications to accept or reject the data they pull from the database based on the integrity verification i.e., just because the data is in database, the applications will not act upon it unless it has been proved that the data is authentic. Therefore, in our solution, *hash* will be taken for each record and *hash* along with the primary key will be stored on a file called *hash files*, which will be at service/interface. Separate files are maintained for each table to store the *hash*. Whenever a record is to be written onto the database, the *hash* must be computed for the record. The *hashed value* along with the primary key is stored in the file. If a record is to be read from the database, the record is retrieved and *hash* is computed and checked with the corresponding value stored in the file. If not tallied, a tampered message is sent to the application. The solution presented here can only detect the unauthorized changes made

to the database at record level but cannot be prevented.

The following are the proposed solutions.

5.1 Security Wrapper Library

The architecture of *Security Wrapper Library* is as specified in fig.1, which includes a package that provides two APIs, namely *readAPI* and *writeAPI*. The application must import this package and call these APIs during each *write* or *read* on the database. The format of readAPI is *boolean readAPI (file_name, primary_key, record)* and that of writeAPI is *boolean writeAPI (file_name, primary_key, record)*. In addition, *delAPI* is provided in the package for handling deletions of records in the file. For modification of any record *writeAPI* can be used to modify the corresponding record in the *hash file*.

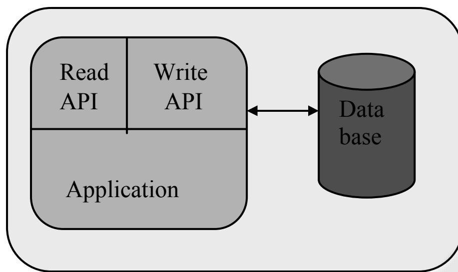

Figure 1: Security Wrapper Library architecture

For *writing/modification* into database,

- The application writes/modifies the record into the database and calls the *write API.*
- The *write API* takes the table name, record name and the primary key as input arguments.
- The API opens the corresponding *hash file* for the table.
- The *hash* value is stored in the file along with primary key.
- The *write API* returns true to the application whenever this process succeeds else an error message is thrown onto the application.

For *reading* from the database:

- The application reads the record from the database and calls the *read API.*
- The *read API* opens the *hash* file, and computes *hash* on the retrieved record.
- This *hash* value is compared with the *hash* value stored in the file and returns to the application whether the record is tampered or not.
- Now the file is closed with the corresponding message returned to application.

For *deletion* from the database:

- The application deletes the record from the database and calls the *delAPI*
- The *delAPI* opens the *hash file*, and searches for the record with that primary key.
- If the record is found in the *hash file*, the record is deleted in the file and the *hash file* is rearranged.
- The file is and returns true to the application.
- If the record is not found then an exception is thrown onto the application.

What happens when two applications are accessing the same file at the same time? Since each application will run on their own JVM (Java Virtual Machine) instance, the resources used by one JVM instance cannot be accessed by other JVM instance until it is released. Hence, there is no chance of overlap writes or reads on the file. In this architecture, the application must execute the operations (write/modify/read/delete) onto the database first and after receiving acknowledgement, the corresponding changes must be done onto the file. Hence, the application providers must maintain the synchronization between database and the file.

The idea here is to separate the *hash file* from the database so that DBA has no control over the *hash file*, hence any modifications done by the administrator or hacker with privileges of DBA cannot do the corresponding changes in the file, and hence the modifications can be detected. This is similar to the policy of 'Separation of Duties'.

5.2 Security Wrapper Service

In the above solution, every time when the API is called, the file will be opened and after doing the necessary computations, the file is closed. If the API is called 1000 times, the file will be opened and closed 1000 times. Another drawback of the above approach is that the applications cannot access the file unless it is not accessed/used by any other application at that time. These became a bottleneck for the Security Wrapper Library Architecture. To avoid the above problems, a new proposal is made, called *Security Wrapper Service* whose architecture is given in fig.2.

In order to overcome the aforementioned performance and efficiency bottlenecks, this architecture provides them as a service at a specific port. The clients have to connect to the service at the specific port and authenticate themselves by the username and password of the application. Hence any number of clients can be connected and get the services concurrently. Using multithreading concept and creating a separate thread for each client connecting to the service can do this. The write/read

overlaps are also avoided since only the service is working on the file and not the clients.

The files are in encrypted form while they are not in use by the service and the service can only decrypt those files by using a secret key known to the service. The service locks the file after decryption so that no other application can view/access the content of the file in plain text. While closing, the service will first encrypts the file and unlocks it. This adds an extra layer of protection in addition to access controls to the hash files.

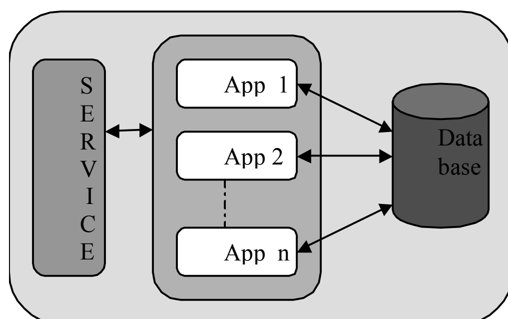

Figure 2: Security Wrapper Service Architecture

Registration at the service:

The applications have to register at the service initially. In order to register, the application provider must contact the service and must provide the application name and username, and password for logging into the service. The application provider at the time of registering must provide the following configuration information to the service: the size of the primary key and size of the hashed data which is determined by choosing one of the hashing algorithms provided by the service. The application provider can specify multiple configurations each will be distinguished by an id, returned by the service. A single file is maintained for each id by the service.

In order to connect to the service the clients have to use the following formats in their code:

Connection(username, password): The service authenticates the client using the username and password and establishes a connection.

Open(id): The service opens the file determined by the id.

Read(id, primary key, record): The service calculates the hash based on primary key, record and compares it with the hash retrieved from the file specified by id.

Write(id, primary key, record): The service computes hash on the record and stores the primary key and hash on the file determined by id.

Close(id): Closes the file determined by id.

CloseConnection( ): The service closes the connection with the client.

Service role:

- The service is deployed at a specific port.
- During start up, the service opens the *hash files,* decrypts them, locks these files and listens to the clients.
- The files will be closed whenever the service is stopped or when the file is not used by the service for a specific period.
- The lock on the file is released and the *hash file* is encrypted while closing.

Client role:

For *writing* into database:

- The applications connect to the service as clients.
- The application writes data into the database.
- The application sends the database name, table name, record name, and primary key as the parameters with the *write* command to the service.
- Service computes the *hash* value on the record and stores it in the *hash* file along with primary key.

For *reading* into the database:

- The applications connect to the service as clients.
- The application reads the data from the database
- Application sends the database name, record name, and primary key as the input parameters with the read command to the service.
- The service calculates the *hash* value on the retrieved record and compares it with the value stored in the file.
- If both the values match then a flag with value set to *true* be returned else *false* is returned to the application.

For *modifying* into database:

- The applications connect to the service as clients.
- The application modifies the data at the database.
- The application sends the database name, table name, record name, and primary key as the parameters with the modify command to the service.
- The service searches in the hash file for the primary key.
- The hash is computed for the record and modifies the data at the node where the primary key is found.

The advantages of this solution over the previous one are: 1.The files need not be opened and closed every time the service is called. 2. Multiple clients can be connected and utilize the service concurrently. 3. This service can be hosted as a trusted third party service with in an enterprise or across enterprises.

5.3 Enterprise Security Wrapper

The above two solutions need the application to utilize the services or the APIs provided and hence has to modify the code of the application. In order to make this service an application transparent, we propose a new architecture called *Enterprise security wrapper* architecture, which is given in fig.3.

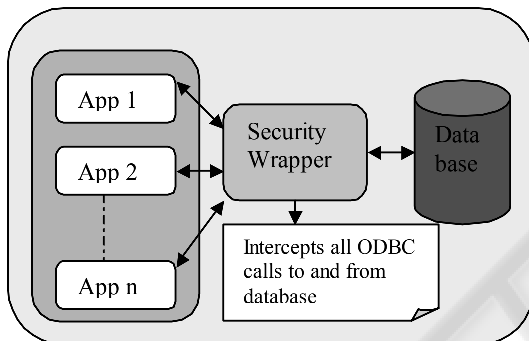

Figure 3: Enterprise security wrapper Architecture

In this solution a *Security Wrapper* acts as intermediary between the database and applications. The applications instead of connecting to database connect to the wrapper that in turn connects to the database. Hence, The Security Wrapper *intercept*s all the *ODBC* calls from the applications and categorizes them into one of the DDL/DML statements and processes them as done in the service for each category and redirect the query to the database through ODBC connection and get the results and again necessary processing is done. The result is returned to the application only when the retrieved records are authentic else error messages are thrown onto the application.

- Once the application is started, it is connected to the *Security Wrapper* instead of connecting to database.
- The *Security Wrapper intercepts the odbc call to the database.*
- These calls are classified as Insert or Select or Update or Delete, then is processed accordingly.
- For Insertion,
  The wrapper connects to the database through *odbc* and writes the record into the database. Upon successful write on the database, the wrapper computes the *hash* value on the record. The security wrapper stores the hash value along with the primary key in the *hash* file.
- For *Select* operation,
  The wrapper retrieves the record from the database through the *odbc* and computes the *hash* value. This *hash* value is compared with the value already stored in the *hash file.* If both the values are same then the record is returned, else a "*TAMPERED MESSAGE*" is retuned to the respective application.
- For *Updating*,
  The *Wrapper* writes the record into the database. Upon successful write the *hash* value is computed on the record. This new *hash* value is replaced with the previous *hash* value in the *hash file.*
- For *deletion*,
  The wrapper connects to the database deletes the record. Upon successful deletion the *hash* value for that particular record present in the *hash* file is deleted.

Hence, this architecture accepts the modifications that are done only through this interface and notify the applications if any modifications are done through any other means. This architecture also provides database and OS independency and application transparency where the applications can just plug-in to this interface. This security wrapper service acts as a firewall between database and applications.

We can call the Enterprise Security Wrapper as 'Database Firewall' due to following reasons: 1. It acts as an agent for the applications to access the other side of wrapper i.e., the database. 2. As Firewall filters outgoing traffic, the Enterprise Security Wrapper also filters the data and allows only authentic data to pass through it and blocks the unauthentic data. 3. Extensive security/ validity scans can be done at Firewall and in this wrapper scans for the integrity of data is done. 4. Authentication, logging and auditing can be done at the Enterprise Security Wrapper.

Implementation Considerations

MD5, SHA-1, SHA-256, SHA-384, SHA-512 etc. can be used as hashing algorithms that are called internally by the API's of the package/library or by the service. The next question is what the *hash file* contains and how they are maintained for faster retrieval. The hash file contains the primary key and the hash of the records. To make the searching,

insertion, or deletion of a record computationally faster, they must be maintained in a data structure in the file. The strong contenders for this purpose are B-trees and B+ Trees. We selected B-trees for the following reasons. 1. B-trees store data in the internal nodes also whereas the data is stored at leaves only in B+ trees. 2. B+ Trees has added advantages of faster retrieval than B-trees but it will take more amount of space. 3. As our records contain primary key and hash that is of small and fixed size, B-Trees save more space.

6 PERFORMANCE EVALUATION

For performance Evaluation, a simple application is developed that takes the number of records to be generated as input and generate those many records continuously. The application was developed using java servlets and is run on Apache Tomcat 5(Apache Jakarta). These records, of size 512KB, are used to *write* into and *read* from the database. The time taken to write the selected number of records into the database in terms of seconds is noted and a graph is drawn with the number of records on X-axis and the time taken on Y-axis. Similarly, a graph is drawn to perform *read* operation on the database. The performance of the two solutions (Standalone & File Service solution) is evaluated and results are compared graphically. In this evaluation SHA-1 160 bit (SHS, 1995) is used as *Hashing* algorithm for both the solutions. The experiments are performed on a PC with Celeron 2.4GHz and 128 Mbytes of main memory under the operating system Windows XP.

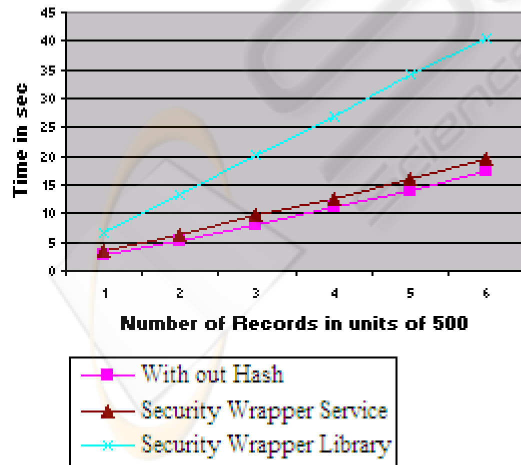

Figure 4: performance evaluation while writing into database

Fig 4 shows the performance graph of the first two solutions while writing into the database, taking number of records as a multiple of 500 on X-axis and time in seconds on Y-axis (in units of 5 sec). The Blue colored line represents the readings for *Security Wrapper Library*, brown for the *Security Wrapper Service* and the pink represents retrieval with out using *hash*. Fig 5 shows the performance graph of the first two solutions while reading from the database, taking number of records as a multiple of 500 on X-axis and time in milliseconds on Y-axis (in units of 1 sec). The Blue colored line represents the readings for *Security Wrapper Library*, brown for the *Security Wrapper Service* and the pink represents retrieval with out using *hash*.

7 CONCLUSIONS

Data privacy and protection is the need of the hour to protect the sensitive data from malicious attacks. The outsourcing countries like US also stress the importance of them. Hence, in the near future acts like Data Protection Act, HIPAA Act will be enforced in India also.

We have worked on the first two *solution*s and shown the performance evaluations and we would like to extend our ideas to the *Enterprise security wrapper* and make a product-based solution. We are presently working on the records, however we have to integrate the above solution to procedures, triggers and configuration files. We can also extend the solution to include Encryption algorithms, which the applications can use for maintaining confidentiality of certain sensitive information. Once the acts are enforced in India, all the banking and financial applications can go for a product of this architecture.

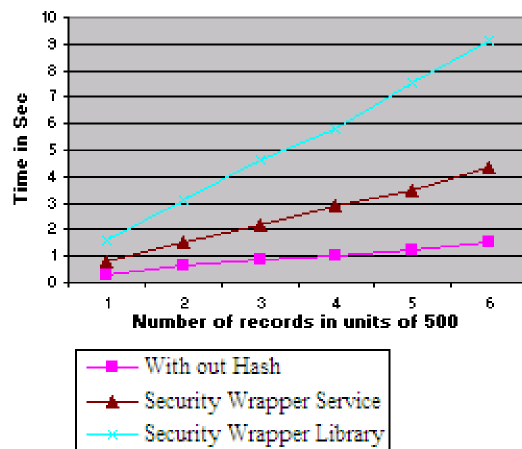

Figure 5: performance evaluation while reading from the database

REFERENCES

Arup Nanda & Donald, K.B. (2004), *Oracle Privacy Security Auditing*, Rampant TechPress.

Apache Jakarta Tomcat Server. Retrieved August 8, 2004, from http://jakarta.apache.org/tomcat/index.html

Chang, N.Z. & Honglan, Z. (2004), *An Integrated Approach for Database Security and Fault Tolerance,* in Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC ’04).

Data Protection Act. (n.d.). Retrieved August 8, 2004, from http://www.dataprivacy.ie/6ai.htm.

DbEncrypt Product Details, (n.d.). Retrieved August 8, 2004, from http://www.appsecinc.com/products

Dave, D & Susan, D. (2004), *Review: DB Confidential.* Retrieved August 8, 2004, from http://nwc.securitypipeline.com/showArticle.jhtml?articleID=18901525

Hacigumis, H & Iyer, B & Mehrotra, S. (2002a, March), *Providing database as a service,* in Proceedings of the 18th International Conference on Data Engineering (ICDE’02).

Hacigumis, H & Iyer, B & Mehrotra, S. (2002b), *Encrypted Database Integrity in Database Service Provider Model,* in International Workshop on Certification and Security in E-Services.

Jef Poskanger, *ACME Crypto Library.* Retrieved August 8, 2004, from http://www.acme.com/java/software/Package-Acme.Crypto.html

Java Documentation. Retrieved August 8, 2004, from http://java.sun.com/j2se/

Gramm-Leach-Bliley Act. (n.d.) Retrieved August 25, 2004,from http://www.ftc.gob/privacy/glbact/glbsub1.html

Mykleuton, E & Narasimha, M & Tsudik, G. (2003a), *Providing Authentication and Integrity in Outsourced Databases using Merkley Hash Trees*, UCI_SCONCE Technical Report, from http://sconce.ics.uci.edu/das/MerkleODB.pdf

Mykleuton, E & Narasimha, M & Tsudik, G. (2003b), *Authentication and Integrity in Outsourced Databases,* University of California, Irvine.

Richard, G. (1984), *The Integrity-Lock Approach to Secure Database Management,* The Mitre Corporation, Bedford, MA.

Sarbanes Oxley Section 404, A Toolkit for Management and Auditors. (n.d.). Retrieved August 8, 2004, from www.pwc.com/ca/eng/about/svcs/sox_404_v2.pdf.

Secure Hash Standard, NIST. (n.d.). Retrieved August 8, 2004, from http://csrc.nist.gov/cryptval/shs.html

Secure hash Standard (1995), FIPS Publications 180-1. Retrieved August 8, 2004, from http://www.itl.nist.gov/fipspubs/fip180-1.htm

White Papers from Application Security Inc. (n.d.). Retrieved August 8, 2004, from http://www.appsecinc.com/whitepapers/

White Papers from nCipher. (n.d.). Retrieved August 8, 2004, from http://active.ncipher.com/index.php

XP_Crypt Product Details. (n.d.). Retrieved August 8, 2004, from http://www.act