# A DATABASE ENCRYPTION SOLUTION THAT IS PROTECTING AGAINST EXTERNAL AND INTERNAL THREATS, AND MEETING REGULATORY REQUIREMENTS

## *A practical implementation of field level privacy*

Keywords: Isolation, Intrusion Tolerance, Database Security, Encryption, Privacy, VISA CISP, GLBA, HIPAA.

Abstract: Security is becoming one of the most urgent challenges in database research and industry, and there has also been increasing interest in the problem of building accurate data mining models over aggregate data, while protecting privacy at the level of individual records. Instead of building walls around servers or hard drives, a protective layer of encryption is provided around specific sensitive data-items or objects. This prevents outside attacks as well as infiltration from within the server itself. This also allows the security administrator to define which data stored in databases are sensitive and thereby focusing the protection only on the sensitive data, which in turn minimizes the delays or burdens on the system that may occur from other bulk encryption methods. Encryption can provide strong security for data at rest, but developing a database encryption strategy must take many factors into consideration. This paper presents a practical implementation of field level encryption in enterprise database systems, based on research and practical experience from many years of commercial use of cryptography in database security. We presents how this column-level database encryption is the only solution that is capable of protecting against external and internal threats, and at the same time meeting all regulatory requirements. We use the key concepts of security dictionary, type transparent cryptography and propose solutions on how to transparently store and search encrypted database fields. In this paper we will outline the different strategies for encrypting stored data so you can make the decision that is best to use in each different situation, for each individual field in your database to be able to practically handle different security and operating requirements. Application code and database schemas are sensitive to changes in the data type and data length. The papers presents a policy driven solution that allows transparent data level encryption that does not change the data field type or length. We focus on how to integrate modern cryptography technology into a relational database management system to solve some major security problems.

## 1 INTRODUCTION

Critical business data in databases is an obvious target for attackers. Although access control has been deployed as a security mechanism almost since the birth of large database systems, for a long time security of a DB was considered an additional problem to be addressed when the need arose, and after threats to the secrecy and integrity of data had occurred [3]. Now many major database companies are adopting the loose coupling approach and adding optional security support to their products. You can use the encryption features of your Database Management System (DBMS), or perform encryption and decryption outside the database. Each of these approaches has its advantages and disadvantages. The approach of adding security support as an optional feature is not very satisfactory, since it would always penalize the system performance, and more importantly, it is likely to open new security holes. Database security is a wide research area [6, 3] and includes topics such as statistical database security [1], intrusion detection [14], and most recently privacy preserving data mining [2], and related papers in designing information systems that protect the privacy and ownership of individual information while not impeding the flow of information, include [2, 3, 4, 5].

## 2 CHOOSING THE POINT OF ENCRYPTION

Implementing a data privacy solution can be done at multiple places within the enterprise. There are implementation decisions to be made as well. Where will you perform the data encryption — inside or outside of the database? Your answer can affect the data's security. How do you create a system that minimizes the number of people who have access to the keys? Storing the encryption keys separately from the data they encrypt renders information useless if an attacker found a way into the database through a backdoor in an application. In addition, separating the ability of administers to access or manage encryption keys builds higher layers of trust and control over your confidential information infrastructure. There should be limited access to the means to decrypt sensitive information – and this access should be locked down and monitored with suspicious activity logged. Choosing the point of implementation not only dictates the work that needs to be done from an integration perspective but also significantly affects the overall security model. The sooner the encryption of data occurs, the more secure the environment—however, due to distributed business logic in application and database environments, it is not always practical to encrypt data as soon as it enters the network. Encryption performed by the DBMS can protect data at rest, but you must decide if you also require protection for data while it's moving between the applications and the database. How about while being processed in the application itself, particularly if the application may cache the data for some period? Sending sensitive information over the Internet or within your corporate network clear text, defeats the point of encrypting the text in the database to provide data privacy. Good security practice is to protect sensitive data in both cases – as it is transferred over the network (including internal networks) and at rest. Once the secure communication points are terminated, typically at the network perimeter, secure transports are seldom used within the enterprise. Consequently, information that has been transmitted is in the clear and critical data is left unprotected. One option to solve this problem and deliver a secure data privacy solution is to selectively parse data after the secure communication is terminated and encrypt sensitive data elements at the SSL/Web layer. Doing so allows enterprises to choose at a very granular level (usernames, passwords, etc.) sensitive data and secure it throughout the enterprise. Application-level encryption allows enterprises to selectively encrypt granular data within application logic. This solution also provides a strong security framework and, if designed correctly, will leverage standard application cryptographic APIs such as JCE (Java-based applications), MS-CAPI (Microsoft-based applications), and other interfaces. Because this solution interfaces with the application, it provides a flexible framework that allows an enterprise to decide where in the business logic the encryption/decryption should occur. Some of these applications include CRM, ERP, and Internet-based applications. This type of solution is well suited for data elements (e.g. credit cards, email addresses, critical health records, etc.) that are processed, authorized, and manipulated at the application tier. If deployed correctly, application-level encryption protects data against storage attacks, theft of storage media, and application-level compromises, and database attacks, for example from malicious DBAs. Although it is secure, application encryption also poses some challenges. If data is encrypted at the application, then all applications that access the encrypted data must be changed to support the encryption/decryption model. Clearly, during the planning phase, an enterprise must determine which applications will need to access the data that is being encrypted. Additionally, if an enterprise leverages business logic in the database in the form of stored procedures and triggers, then the encrypted data can break a stored procedure. As a result application-level encryption may need to be deployed in conjunction with database encryption so that the DBMS can decrypt the data to run a specific function. Finally, while leveraging cryptographic APIs is useful, the implementation does require application code changes as well as some database migration tasks to address field width and type changes as a result of encryption, if not type-preserving encryption is used. And while homegrown applications can be retrofitted, off the shelf applications do not ship with the source and often do not provide a mechanism to explicitly make a cryptographic function call in the logic.

### 2.1 Database-layer Encryption

Database-level encryption allows enterprises to secure data as it is written to and read from a database. This type of deployment is typically done at the column level within a database table and, if coupled with database security and access controls, can prevent theft of critical data. Database-level encryption protects the data within the DBMS and also protects against a wide range of threats, including storage media theft, well known storage attacks, database-level attacks, and malicious DBAs. Database-level encryption eliminates all application

changes required in the application-level model, and also addresses a growing trend towards embedding business logic within a DBMS through the use of stored procedures and triggers. Since the encryption/decryption only occurs within the database, this solution does not require an enterprise to understand or discover the access characteristics of applications to the data that is encrypted. While this type of solution can certainly secure data, it does require some integration work at the database level, including modifications of existing database schemas and the use of triggers and stored procedures to undertake encrypt and decrypt functions. Additionally, careful consideration has to be given to the performance impact of implementing a database encryption solution, particularly if support for accelerated index-search on encrypted data is not used. First, enterprises must adopt an approach to encrypting only sensitive fields. Second, this level of encryption must consider leveraging hardware to increase the level of security and potentially to offload the cryptographic process in order to minimize any performance impact. The primary vulnerability of this type of encryption is that it does not protect against application-level attacks as the encryption function is strictly implemented within the DBMS.

## 2.2 Storage-Layer Encryption

Storage-level encryption enables enterprises to encrypt data at the storage subsystem, either at the file level (NAS/DAS) or at the block level SAN. This type of encryption is well suited for encrypting files, directories, storage blocks, and tape media. In today's large storage environments, storage-level encryption addresses a requirement to secure data without using LUN masking or zoning. While this solution does provide the ability to segment workgroups and provides some security, it presents a couple limitations. First, it only protects against a narrow range of threats, namely media theft and storage system attacks. However, storage-level encryption does not protect against most application- or database-level attacks, which tend to be the most prominent type of threats to sensitive data. Second, current storage security mechanisms only provide block-level encryption; they do not give the enterprise the ability to encrypt data within an application or database at the field level. Consequently, one can encrypt an entire database, but not specific information housed within the database.

# 3 USER MANAGEMENT ISSUES

To access database resources, a user must have an account with the database. User account management is the basis for the overall database system security. A DBA has the responsibility to create and maintain all DB user accounts, which Is a large portion of her/his system administration effort. At the account creation time, the DBA species how the newly created user will be authenticated, and what system resources the user can use. When a user wants to connect to a database, she/he must identify her-self/himself to the server and the server will verify her/his identity using the pre-specied authentication method. Current commercial RDBMSs support many dierent kinds of identication and authentication methods, among them are password-based authentication [12], host-based authentication [4, 12, 11], PKI (Public Key Infrastructure) based authentication [19], and other third party-based authentications such as Kerberos [17], DCE (Distributed Computing Environment [23]) and smart card [20]. Essentially, all methods rely on a secret known only to the connecting user. It is vital that a user should have total control over her/his own secret. For example, only she/he should be able to change her/his password. Other people can change a user's password only if they are authorized to do so. In a DB system, a DBA can reset a user's password upon the user's request, probably because the user might have forgotten her/his password. However, as we have noticed before, the DBA can temporarily change a user's password without being detected and caught by the user, because the DBA has the capability to update (directly or indirectly) the system catalogs. Design Issues in Encryption and Key Management. The most important problem in using encryption/decryption is key management implementation across all database platforms in an enterprise. When we consider incorporating encryption in a database server, there are two design issues:

1. There should be a way for a user to indicate that some data should be encrypted before storage, and an option to send encrypted data to the application tier.

2. There should be a way for a user to specify (explicitly or implicitly) a key that will be used for data encryption, and optional use of HSM (Hardware Security Module) support.

## 3.1 A separated Security Directory

A traditional data directory stores all of the information that is used to manage the objects in a database. A data directory consists of many catalog tables and views. It is generally recommended that users (including DBAs) do not change the contents of a catalog table manually. Instead, those catalogs will be maintained by the DB server and updated only through the execution of system commands. However, a DBA can still make changes in a catalog table if she/he wants to do so. To prevent unauthorized access to important security-related information, we introduce the concept of security catalog. A security catalog is like a traditional system catalog but with two security properties: It can never be updated manually by anyone, and its access is controlled by a strict authentication and authorization policy.

# 4 COMPLETE ACCOUNTABILITY

From an administration point of view, a DBA (Database Administrator) is playing an important and positive role. However, when security and privacy become a big issue, we cannot simply trust particular individuals to have total control over other people's secrecy. This is not just a problem of trustiness, it is a principle. Technically, if we allow a DBA to control security without any restriction, the whole system becomes vulnerable because if the DBA is compromised, the security of the whole system is compromised, which would be a disaster. On the other hand, if we have a mechanism in which each user could have control over his/her own secrecy, the security of the system is maintained even if some individuals do not manage their security properly. Access control is the major security mechanism deployed in all RDBMSs. It is based upon the concept of privilege. A subject (i.e., a user, an application, etc.) can access a database object if the subject has been assigned the corresponding privilege. Access control is the basis for many security features. Special views and stored procedures can be created to limit users' access to table contents. However, a DBA has all the system privileges. Because of her/his ultimate power, a DBA can manage the whole system and make it work in the most efficient way. In the mean time, she/he also has the capability to do the most damage to the system. With a separated security directory the security administrator is responsible for setting the user permissions. Thus, for a commercial database, the security administrator (SA) operates through a separate middle-ware, the access control system (ACS), which serve for authentication verification, authorization, audit, encryption and decryption. The ACS is tightly coupled to the database management system (DBMS) of the database. The ACS controls access in real-time to the protected fields of the database. Such a security solution provides separation of the duties of a security administrator from a database administrator (DBA). The DBA's role could for example be to perform usual DBA tasks, such as extending tablespaces etc, without being able to see (decrypt) sensitive data. The SA could then administer privileges and permissions, for instance add or delete users. For most commercial databases, the database administrator has privileges to access the database and perform most functions, such as changing password of the database users, independent of the settings by the system administrator. An administrator with root privileges could also have full access to the database. This is an opening for an attack where the DBA can steal all the protected data without any knowledge of the protection system above. The attack is in this case based on that the DBA impersonates another user by manipulating that users password, even though the user's password is enciphered by a hash algorithm. An attack could proceed as follows. First the DBA logs in as himself, then the DBA reads the hash value of the users password and stores this separately. Preferably the DBA also copies all other relevant user data. By these actions the DBA has created a snapshot of the user before any altering. Then the DBA executes the command "ALTER USER username IDENTIFIED BY newpassword". The next step is to log in under the user name "username" with the password "newpassword" in a new session. The DBA then resets the user's password and other relevant user data with the previously stored hash value. Thus, it is important to further separate the DBA's and the SA's privileges. For instance, if services are outsourced, the owner of the database contents may trust a vendor to administer the database. Then the role of the DBA belongs to an external person, while the important SA role is kept within the company, often at a high management level. Thus, there is a need for preventing a DBA to impersonate a user in a attempt to gain access to the contents of the database. The method comprises the steps of: adding a trigger to the table, the trigger at least triggering an action when an administrator alters the table through the database management system (DBMS) of the database; calculating a new password hash value differing from the stored password hash value when the trigger is triggered; replacing the stored password hash value with the new password hash value. A similar authentication verification may also

be implemented if VPN based connection and authentication is used.

The first security-related component in an RDBMS (and actually in most systems) is user management. A user account needs to be created for anyone who wants to access database resources. However, how to maintain and manage user accounts is not a trivial task. User management includes user account creation, maintenance, and user authentication. A DBA (DataBase Administrator) is responsible for creating and managing user accounts. When a DBA creates an account for user Alice, she/he also species how Alice is going to be authenticated, for example, by using a database password. The accounts and the related authentication information are stored and maintained in system catalog tables. When a user logs in, she/he must be authenticated in the exact way as specified in her/his account record. However, there is a security hole in this process. A DBA can impersonate any other user by changing (implicitly or explicitly) the system catalogs and she/he can do things on a user's behalf without being authorized/detected by the user, which is a security hole. A DBA's capability to impersonate other users would allow her/him to access other users' confidential data even if the data are encrypted.

# 5 CHOSING THE STORAGE FORMAT OF ENCRYPTED INFORMATION

Application code and database schemas are sensitive to changes in the data type and data length. If data is to be managed in binary format, varbinary can be used as the data type to store encrypted information. On the other hand, if a binary format is not desirable, the encrypted data can be encoded and stored in a varchar field. There are size and performance penalties when using an encoded format, but this may be necessary in environments that do not interface well with binary formats, if support for transparent data level encryption is not used. In environments where it is unnecessary to encrypt all data within a database, a solution with granular capabilities is ideal. Even if only a small subset of sensitive information needs to be encrypted, additional space will still be required if transparent data level encryption is not used. The secure data level encryption for data at rest can be based on block ciphers. The proposed solution is based on transparent data level encryption with Data Type Preservation that Does Not Change ASCII Data Field Type or length. The solution provides a cost effective implementation, avoiding changes of Millions of Lines of Business Code in larger enterprise information systems. The solution also provides an effective last line of defence: selective column-level data item encryption, cryptographically enforced authorization; key management based on hardware or software, secure audit and reporting facility, and enforced separation of duties. The method is Cryptographically Strong, Work With Any DBMS and OS, Work With Different Character Sets, No Application or Database Changes, No Programming Language Dependence, Fail Safe, Requires no DBA intervention. Loader Functions Normally and Queries Function Normally. Accelerated search capabilities based on partial encryption of data and accelerated search index can also be utilized.

## 5.1 Searching on encrypted data

Searching for an exact match of an encrypted value within a column is possible, provided that the same initialization vector is used for the entire column. On the other hand, searching for partial matches on encrypted data within a database can be challenging and can result in full table scans if support for accelerated index-search on encrypted data is not used. One approach to performing partial searches, without prohibitive performance constraints and without revealing too much sensitive information, is to apply an HMAC to part of the sensitive data and store it in another column in the same row, if support for accelerated index-search on encrypted data is not used. For example, a table that stores encrypted customer email addresses could also store the HMAC of the first four characters of each email address. This approach can be used to find exact matches on the beginning or end of a field. One drawback to this approach is that a new column needs to be added for each unique type of search criteria. So if the database needs to allow for searching based on the first four characters as well as the last five characters, two new columns would need to be added to the table. However, in order to save space, the HMAC hash values can be truncated to ten bytes without compromising security in order to save space. This approach can prove to be a reasonable compromise especially when combined with non-sensitive search criteria such as zip code, city, etc. and can significantly improve search performance if support for accelerated index-search on encrypted data is not used.

## 5.2 Encryption of Primary and Foreign Keys

Encrypted columns can be a primary key or part of a primary key, since the encryption of a piece of data is stable (i.e., it always produces the same result), and no two distinct pieces of data will produce the same cipher text, provided that the key and initialization vector used are consistent. However, when encrypting entire columns of an existing database, depending on the data migration method, database administrators might have to drop existing primary keys, as well as any other associated reference keys, and re-create them after the data is encrypted. For this reason, encrypting a column that is part of a primary key constraint is not recommended if support for accelerated index-search on encrypted data is not used. Since primary keys are automatically indexed there are also performance considerations, particularly if support for accelerated index-search on encrypted data is not used. A foreign key constraint can be created on an encrypted column. However, special care must be given during migration. In order to convert an existing table to one that holds encrypted data, all the tables with which it has constraints must first be identified. All referenced tables have to be converted accordingly. In certain cases, the referential constraints have to be temporarily disabled or dropped to allow proper migration of existing data. They can be re-enabled or recreated once the data for all the associated tables is encrypted. Due to this complexity, encrypting a column that is part of a foreign key constraint is not recommended, if automated deployment tools are not used. Unlike indexes and primary keys, though, encrypting foreign keys generally does not present a performance impact.

## 5.3 Indexing of encrypted columns

Indexes are created to facilitate the search of a particular record or a set of records from a database table. Carefully plan before encrypting information in indexed fields. Look-ups and searches in large databases may be seriously degraded by the computational overhead of decrypting the field contents each time searches are conducted if accelerated database indexes are not used. This can prove frustrating at first because most often administrators index the fields that must be encrypted – social security numbers or credit card numbers. New planning considerations will need to be made when determining what fields to index if accelerated database indexes are not used. Indexes are created on a specific column or a set of columns. When the database table is selected, and WHERE conditions are provided, the database will typically use the indexes to locate the records, avoiding the need to do a full table scan. In many cases searching on an encrypted column will require the database to perform a full table scan regardless of whether an index exists. For this reason, encrypting a column that is part of an index is not recommended, if support for accelerated index-search on encrypted data is not used.

## 5.4 Use of Initialization Vectors

When using CBC mode of a block encryption algorithm, a randomly generated initialization vector is used and must be stored for future use when the data is decrypted. Since the IV does not need to be kept secret it can be stored in the database. If the application requires having an IV per column, which can be necessary to allow for searching within that column, the value can be stored in a separate table. For a more secure deployment, but with limited searching capabilities if support for accelerated index-search on encrypted data is not used, an IV can be generated per row and stored with the data. In the case where multiple columns are encrypted, but the table has space limitations, the same IV can be reused for each encrypted value in the row, even if the encryption keys for each column are different, provided the encryption algorithm and key size are the same.

# 6 IMPLEMENTING ENCRYPTION KEY MANAGEMENT

One of the essential components of encryption that is often overlooked is key management, which refers to the way cryptographic keys are generated and managed throughout their life. Because cryptography is based on keys that encrypt and decrypt data, your database protection solution is only as good as the protection of your keys. Security depends on two factors: where the keys are stored and who has access to them. When evaluating a data privacy solution, it is essential to include the ability to securely generate and manage keys. This can often be achieved by centralizing all of the tasks of key management on a single platform and effectively automating administrative key management tasks, which will lead to both operational efficiency and reduced cost of management. Data privacy solutions should also include an automated and secure mechanism for key

rotation, replication, and backup. Today's complex and performance sensitive environments require the use of a combination of software cryptography and specialized cryptographic chipsets, HSM, to balance security, cost, and performance needs. One easy solution is to store the keys in a restricted database table or file. But, all administrators with privileged access could also access these keys, decrypt any data within your system and then cover their tracks. Your database security in such a situation is based not on industry best practice, but based on an honour code with your employees. If your human resources department locks employee records in file cabinets where one person is ultimately responsible for the keys, shouldn't similar precautions be taken to protect this same information in its electronic format? All fields in a database do not need the same level of security. With tamper-proof hardware and software implemented, the encryption being provided by different encryption processes utilizing at least one process key in each of the categories master keys, key encryption keys, and data encryption keys, the process keys of different categories being held in the encryption devices;

wherein the encryption processes are of at least two different security levels, where a process of a higher security level utilizes the tamper-proof hardware device to a higher degree than a process of a lower security level; wherein each data element which is to be protected is assigned an attribute indicating the level of encryption needed, the encryption level corresponding to an encryption process of a certain security level. With such a system it becomes possible to combine the benefits from hardware and software based encryption. The software-implemented device could be any data processing and storage device, such as a personal computer. The tamper-proof hardware device provides strong encryption without exposing any of the keys outside the device, but lacks the performance needed in some applications. On the other hand the software-implemented device provides higher performance in executing the encryption for short blocks, in most implementations [26], but exposes the keys resulting in a lower level of security.

## CONCLUSION

This paper presents experience from many years of research and practical use of cryptography in database security. We use the key concepts of security dictionary, type transparent cryptography and propose solutions on how to transparently store and search encrypted database fields. Database attacks are on the rise even as the risks of data disclosure are increasing. Several industries must deal with legislation and regulation on data privacy. In this environment, your security planning must include a strategy for protecting sensitive databases against attack or misuse by encrypting key data elements. Whether you decide to implement encryption inside or outside the database we recommend that encrypted information should be stored separately from encryption keys, strong authentication should be used to identify users before they decrypt sensitive information, access to keys should be monitored, audited and logged, Sensitive data should be encrypted end-to-end, while in transit in the application and while in storage in enterprise databases. A forthcoming paper will discuss performance aspects, transparent storage and search of encrypted database fields in more detail.

## REFERENCES


[1] N. R. Adam and J. C. Wortman. Security-control methods for statistical databases. ACMComputing Surveys, 21(4):515– 556, Dec. 1989.

[2] R. Agrawal and J. Kiernan. Watermarking relational databases. In 28th Int'l Conference on Very Large Databases, Hong Kong, China, August 2002.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In Proc. of the 28th Int'l Conference on Very Large Databases, Hong Kong, China, August 2002.

[4] Agrawal,J.Kiernan,R.Srikant,andY.Xu.Implementing P3P using database technology. In Proc. of the 19th Int'l Con-ference on Data Engineering, Bangalore, India, March 2003.

[5]R.Agrawal,J.Kiernan,R.Srikant,andY.Xu.AnXPath-based preference language for P3P. In Proc. of the 12th Int'l World Wide Web Conference, Budapest, Hungary, May 2003.

[6] D. E. Denning. Cryptography and Data Security. Addison-Wesley Publishing Company, Inc., 1982.

[7] T. Dierks and C. Allen. The TLS Protocol - Version 1.0, Internet -Draft. November 1997.

[8] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol Version 3.0, Internet-Draft. November 1996.

[9] S. Garnkel and G. Spa ord. Web Security & Commerce. O'Reilly & Associates, Inc., 1997.

[10] S. B. Guthery and T. M. Jurgensen. Smart Card Developer's Kit. Macmillan Technical Publishing, 1998.

[11] Informix. Informix-Online Dynamic Server Administrator's Guide, Version 7.1. INFORMIX Software, Inc., 1994.

[12] G. Koch and K. Loney. Oracle8: The Complete Reference. Osborne/McGraw-Hill, 1997.

[13] J. C. Lagarias. Pseudo-random number generators in cryptography and number theory. InCryptology and Computational Number Theory, pages 115{143. American Mathematical Society, 1990.
[14] T. F. Lunt. A survey of intrusion detection techniques. Computer & Security, 12(4), 1993.
[15] National Bureau of Standards FIPS Publication 180. Secure Hash Standard, 1993.

[16] National Bureau of Standards FIPS Publication 46. Data Encryption Standard (DES), 1977.
[17] B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. IEEE Communications, 32(9):33{38, 1994.
[18] San Jose Mercury News. Web site hacked; cards being canceled, Jan. 20, 2000.
[19] Oracle Technical White Paper. Database Security in Oracle8i, November 1999.
[20] W. Rankl and W. E_ng. Smart Card Handbook. John Wiley & Sons Ltd, 1997.
[21] R. Rivest. The MD5 Message-Digest Algorithm, RFC1321 (I). April 1992.
[22] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signature and public key cryptosystems. Communications of the ACM, 21:120{126, February 1978.
[23] W. Rosenberry, D.Kenney, and G. Fisher. Understanding DCE. O'Reilly & Associates, Inc.,1992.
[24] A. Shamir. How to share a secret. Communication of the ACM, 22(11):612{613, 1979.
[25] D. R. Stinson. Cryptography; Theory and Practice. CRC Press, Inc., 1995.
[26] M. Lindemann and SW Smith, Improving DES Hardware Throughput for Short Operations, IBM Research Report, 2001.