ISSN: 0975-833X

Available online at http://www.journalcra.com

SPECIAL ISSUE

International Journal of Current Research
Vol. 3, Issue, 6, pp.291-294, June, 2011

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

RESEARCH ARTICLE

DATA PROTECTION ALGORITHM USING AES

Anitha, P. and *Palanisamy, V

Department of Computer Science and Engineering, Alagappa University, Karaikudi-630003

ARTICLE INFO

Article History:
Received 13th March, 2011
Received in revised form
15th April, 2011
Accepted 20th May, 2011
Published online 14th June 2011

Key words:
Cryptography,
Encryption,
Decryption,
AES algorithm,
Stream cipher,
Hash function.

ABSTRACT

The paper aims at providing a solution for secure storage of the records in a database. The solution should be prone to less security attacks and should take the optimal time for storage. The algorithm used is AES of stream cipher category. The input and the key can be of variable length. Regarding the key for the algorithm, it is the hashed value of the original key along with the Salt value. Since stream cipher, we will encrypt byte by byte using the key. We have a Permutation box (p-box) and Substitution box (s-box) logic to make the algorithm quite complex and to avoid the security breaches. The Key will be in the rotation mode based on a simple logic being implemented in the algorithm thus making it quite complex for attacks. Thus, the records are stored in the encrypted format in the database.

INTRODUCTION

Network Security is becoming more and more important as people spend more and more time connected in a network. Security attacks include unauthorized reading a message of a file or making any modifications of messages of a file etc., one of the primary reasons that intruders can be successful is that most of the information they acquire from a system is in a form that they can read and comprehend. One solution to this problem is, through the use of Cryptography. Cryptography ensures that the messages cannot be intercepted or read by anyone other than the authorized recipient. It prevents intruders from being able to use the information that they capture. Cryptography secures information by protecting its confidentiality and can also be used to protect information about the integrity and authenticity of data. To protect our concern's database assets, the security should be taken today. These include encrypting data as it moves across the enterprise networks and as it sits at rest, in storage on database systems. Extra steps and precautions should be taken to carefully control access this data. This paper will focus on how to protect data at rest.

Hackers Are Not the Only Threat or Even the Most Dangerous

Threats to the databases can come from hackers, attackers external to our network. Without extra precautions taken to secure the confidential data in databases, the concern's privacy is at risk. Here, taking the right security approach enables to protect the critical data infrastructure.

Protecting Data with Encryption

While laws and regulations interpret "protecting privacy" in a number of ways, any enterprise solution for protecting data — especially data at rest-must involve two things: secure encryption technology to protect confidential data and careful management of access to the cryptography keys that unlock the encrypted data.

Encryption Basics: What You Need to Know

To give sensitive data the highest level of security, it should be stored in encrypted form. The goal of encryption is to make data unintelligible to unauthorized readers and extremely difficult to decipher when attacked. Encryption operations are performed by using random encryption keys. The randomness of keys makes encrypted data harder to attack. Keys are used to encrypt data, but they also perform decryption. This paper focuses on a security solution for protection of data at rest, specifically protection of data that resides in databases. The key used in this algorithm is obtained by having the hashed value of the original key generated along with the salt value. Also, the key will be in the rotation mode making quite complex for attacks.

Related works

The first open encryption algorithm, Data Encryption Standard (DES) was adopted by the National Institute of Standards and Technology (NIST) to protect the sensitive information as

***Corresponding author:** anithajoy81@gmail.com*

Federal Information Processing Standard 46 (FIPS PUB 46) in 1977. The shorter length of the key reduces the security of DES. So, Double DES and Triple DES may arise and they also not suitable for long term use because of its slow process. The Rijndael algorithm was adopted as an encryption standard, the Advanced Encryption System (AES) by the NIST as FIPS PUB 197 (FIPS 197) on November 2001 [1]. Computer Security Objects Register (CSOR), the remarkable thing about this entire process has been the openness as well as the international nature of the "competition." NIST maintained an excellent Web site devoted to keeping the public fully informed, at http:// csrc.nist.gov/archive/aes/ [2], which is now available as an archive site. Their Overview of the AES Development effort has full details of the process and algorithms. Daemen and Rijmen, AES Proposal, specifies the Rijndael algorithm [3] and [4], a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192 and 256 bits. Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

Guideline for Implementing Cryptography in the Federal Government [4],[5],[6] determines the implementations of the algorithm that are tested by an accredited laboratory and validated will be considered as complying with this standard. Since cryptographic security depends on many factors besides the correct implementation of an encryption algorithm, Federal Government employees, and others, should also refer to NIST Special Publication 800-21, is available at http://csrc.nist.gov/publication/. Database Encryption Solution using Empress RDBMS [10] white paper by Srdjan Holovac, Empress Software Inc., January 2006. This paper focuses on a security solution for protection of data at rest, specifically protection of data that resides in database. Encryption, the process of disguising data in such a way to hide its substance, is a very effective way to achieve security for data at rest [8],[9]. The recommended cryptographic algorithm is AES, the larger the key length, the more computation it requires and the greater security it provides.

The FIPS PUB specifies AES, a cryptographic algorithm, used to protect data's, which is a symmetric block cipher that can encrypt and decrypt information and Capable of using keys of 128,192,256 bits to encrypt and decrypt data. For AES-128, AES-192 and AES-256 it performs 10,12 and 14 number of rounds respectively, and these leads to a SLOW process for encryption and decryption. Disadvantage is, the encryption of any plaintext with a block cipher will result in the same cipher text, when the SAME KEY is used. Nowadays, the usage of Stream ciphers has evolved significantly over the last 5 to 10 years only**.** The main proposal is, if we maintain the keys properly, we can achieve a secure algorithm using stream ciphers.

Proposed work

We proposed here a solution for secure storage of records in the database which will be in the encrypted format so that the messages cannot be intercepted or read by anyone other than the authorized recipient. The main modules are encryption module and decryption module and the sub-modules are permutation module, substitution module and change key module.

Key Generation using Hash function

A hash function is not an encryption. Encrypted items are always meant for eventual decryption. A hash function, on the other hand, is meant only as a one-way operation. The whole idea of a hash function is that it should be very difficult to calculate the original input value, given the hash value.
If y = hash(x) is a hash function, and y and x both represent text strings, then

- given the output y, it is very hard to deduce the original input x.
- similar inputs will give markedly different outputs.
- input x can have *arbitrary* length.
- output y will have *fixed* length.
- there is only a trivial chance of "collisions", where different inputs give the same output.

The clear text version of the password should be the end user's secret. It should never be directly repeated or stored by an application. A user inputs a password in its original, unaltered form - that is, in "clear text". However, a database should never store passwords in clear text. Instead, the value stored by the database should be calculated as,

Key = hashed (clear-text- password + salt-value)

The intent here is to store text which cannot be easily reverse-engineered back into the original password. The salt-value is a random string added to the password. It's added to prevent simple dictionary-style reverse engineering of the hashed value of the plain text password. The "salt" or "salt value" is a random or pseudo-random sequence of bytes that is combined with the encryption password to create encryption and authentication keys. The salt is generated by the encrypting application and is stored unencrypted with the file data. The addition of salt values to passwords provides a number of security benefits and makes dictionary attacks based on precomputed keys much more difficult. For example, The size of the salt value depends on the length of the encryption key, as follows:

Key size	Salt size
128 bits	8 bytes
192 bits	12 bytes
256 bits	16 bytes

In this paper, the clear-text password is the user's password and it will be hashed along with the salt value, which is the random sequence of bytes and the resulting value can be used as the Key for our AES algorithm. Since Stream cipher category, we will encrypt byte by byte using the key. The key will be in the rotation mode making quite complex for attacks and the records are stored in the encrypted format in the database.

ALGORITHM

Key

Key (k) = Hash (original key + salt value)

- Consider the above key (k) value
- i=0;
- Next key=key(k) [i++];
  1. Current key = next key;
  2. Read a byte from source file.
  3. XOR->(Byte ^ Current key) -> result 1;

4. Pass result1 to pbox( ) -> result 2;
5. if (i==length_of_currentkey)
   { i=0; change Key() }
6. Next key = key(k)[i++];
7. XOR->(result 2^next key)->result 3;
8. Pass result 3 to sbox( )->result (Encrypted value).

Pbox ( )

Pbox (int n)
1. Binary equivalent of n
2. Fill the bits in 2 x 4 matrix by row-wise.
3. Perform clockwise rotation
4. Form a string by taking bits from odd position then even position in matrix.
5. Reverse the digit.
6. Complement the value.

Sbox ( )

```
Sbox (byte by, byte m)
  {
      X=Octal value of m
      Y=(byte ^ x) ^ 10;
  }
```

Change Key ( )

```
Change key ( )
  {
      left shift each char
      In Current key by 'n'
  }
```

The Result of the above operation is the Encrypted text of a single byte in the given input. Repeat the above steps for each and every byte of the given input. Finally, when the encrypted text is obtained, append the salt value at the end, so that while decrypting it, we can easily extract the salt value from the cipher text and perform the hash function on the key with this salt value. For Decryption, consider the cipher text obtained from the above process and perform the reverse of the above process yields the original plaintext. Thus, the decryption process can be achieved.

PERFORMANCE ANALYSIS

Consider the plaintext which may be characters, string or the both and also consider the key stream which is generated. According to the above mentioned algorithm, the encrypted text is obtained. Similarly, by considering the cipher text obtained and by using the same key, we can obtain the Plain text again. The following figure represents the conversion of plaintext to cipher text and back to the plaintext again. By using this algorithm, we can store the data in the database in a well secure manner. Even if the entire database is cracked by the intruders, they can only view the data which is in the encrypted format (i.e.) the cipher text. Unless the intruders know the exact key length, he cannot be able to retrieve the records and even if he give the key length similar to the cipher text length, he may able to get the records which is incorrect. So knowing the exact key is important. At the same time, if the key length is smaller than the given input, the key will be in the rotation mode based on the iteration value and thus making it quite complex for attacks. Thus, the records are stored in the encrypted format in the database.

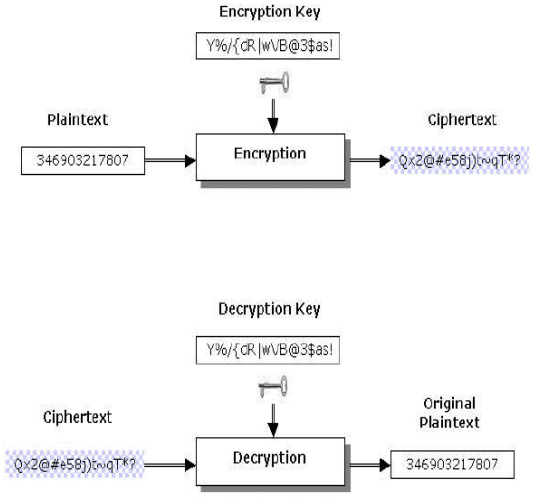

Fig. 1. Encryption and Decryption

This algorithm can be used for the sensitive data storage such as medical records, banking information, business data and other confidential records.

System Parameters

For our experiment, we use laptop of Intel® core ™2 Duo CPU T6600 @ 2.20 GHZ with 3 GB RAM and 32 bit OS.

Experiment Factors

In order to evaluate the performance of the algorithm, the parameters that the algorithms must be tested for must be determined. The chosen factor here to determine the performance is the algorithm's speed to encrypt/decrypt the data blocks of various sizes. By considering the different sizes (0.5 MB to 20 MB), the algorithms were evaluated in terms of the time required to encrypt and decrypt the data block. We are comparing the AES block cipher with our algorithm, which is a stream cipher category. The simulation program is compiled using the default settings in .NET 2003 visual studio for C# windows applications. The experiments will be performed couple of times to assure that the results are consistent and valid to compare these algorithms. This implementation uses managed wrappers for AES (Rijindael Managed) which is available in System. Security. Cryptography that wraps unmanaged implementations available in CryptoAPI. There is only a pure managed implementation of Rijindael available in System, Security, Cryptography which was used in the tests. As we know, the AES block cipher takes much processing time than any other algorithms (DES, 3DES, Blowfish) [16], our algorithm (stream cipher) minimizes the processing time than the block cipher.

Table 1. Comparison of processing time between block cipher and stream cipher

Data Size (MB)	Time (AES block) secs	Time (AES stream) secs
0.5	0.6	0.3
1	1.2	0.9
2	1.9	1.4
5	4.2	2.4
10	7.8	3.9
15	12.4	7.7
20	15.3	12.1

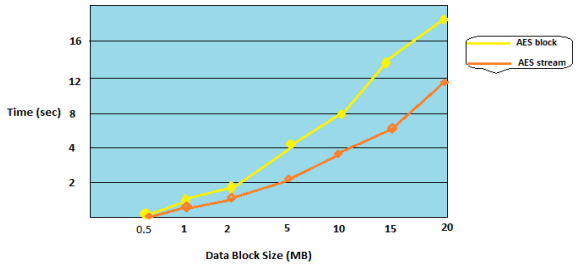

Fig. 2. Graph representing processing time

Conclusion

Enforcing data security is a top priority for both governments and businesses worldwide and for protecting customer's information such as securing the medical records, financial records and other personal information etc. and our project focuses on a security solution for the protection of data that resides in the databases. Most databases are deployed and stored in some kind of storage device such as disks etc., and have the need to backup the data could end up in storage device in plaintext itself. So Encryption, the process of disguising data in such a way to hide its substance, is a very effective way to achieve security. Thus, we use the Advanced Encryption Standard (AES) algorithm which uses the Stream cipher category; encrypt individual characters of a plaintext. The Key will be in the rotation mode based on a simple logic using in this algorithm. Hence, even if the key's length is less than that of the input, it will be in rotation mode and thus making it quite complex for attacks. Thus, the records are stored in the encrypted format in the database. Even if the entire database is cracked by the intruders, they can only view the data which is in the encrypted format (i.e.) the cipher text. Unless the intruders know the exact key length, he cannot be able to retrieve the records. The Current security strategy is to use a single cipher key for each database. A future solution will be implemented where a key per database column can be specified. This will allow the users to tighten their overall security policy when encrypting additional database.

REFERENCES

[1] Federal Information Processing Standards Publications (FIPS PUBS 197) pp. 100-235 issued by NIST after approval by security of commerce to section 5131.

[2] Computer Security Objects Register (CSOR): http://csrc.nist.gov/csor/., January 2001.

[3] Daemen, J. and V. Rijmen, AES Proposal: Rijndael, AES Algorithm Submission, September 3, 2005, available at [1].

[4] Daemen, J. and V. Rijmen, The block cipher Rijndael, Smart Card research and Applications, LNCS 1820, Springer-Verlag ,pp.288-296.

[5] Gladman's, B. AES related home page http://fp.gladman.plus.com/cryptography_technology/.

[6] Guideline for Implementing Cryptography in the Federal Government, should also refer to NIST Special Publication 800-21, is available at http://csrc.nist.gov/publication/.

[7] Garfinkel SL, Shelat A. Remembrance of Data Passed: A Study of Disk Sanitization Practices, *IEEE Security & Privacy*, 1(1), pp. 17-28, 2003.

[8] Garfinkel T, Pfaff B, Chow J, Rosenblum M. Data Lifetime is a Systems Problem, *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop: Beyond the PC*, September 2004.

[9] White paper by Srdjan Holovac, Empre Chow J, Pfaff B, Garfinkel T, Rosenblum M. Shredding your garbage: Reducing data lifetime through secure deallocation, *Proceedings of the USENIX Security Symposium*, August 2005.

[10] Securing data at rest: Database Encryption Solution using Empress RDBMS-white paper by Srdjan Holovac, January 2006.

[11] Mason J, Watkins K, Eisner J, Stubblefield A. A natural language approach to automated cryptanalysis of two-time pads. *Proceedings of the 13th ACM Conference on Computer and Communications Security*, October 2006.

[12] Griffing A. Solving XOR Plaintext Strings with the Viterbi Algorithm. Cryptologia, 30(3), pp. 258-265, 2006.

[13] Halcrow M. eCryptfs: a Stacked Cryptographic Filesystem. *Linux Journal.* April 2007.

[14] Debian Source Repository, http://ftp.de.debian.org /debian, 2008.

[15] The GNU Privacy Guard, http://gnupg.org, 2008.

[16] Performance Analysis of Data Encryption Algorithms, Abdel-Karim Al Tamimi, aa7@wustl.edu, Retrieved October 1, 2008.

[17] Ferguson, N., Schneier, B., and Kohno T., "Cryptography Engineering: Design Principles and Practical Applications". New York: John Wiley and Sons, 2010.

****