ISSN No. 0976-5697

Volume 6, No. 2, March-April 2015

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# HASCII Encryption Algorithm

Akshat Gandhi
Computer Science Department,
K.J.Somaiya College of Engineering,
Mumbai, Maharashtra, India

Chintan Doshi
Computer Science Department,
K.J.Somaiya College of Engineering,
Mumbai, Maharashtra, India

Hemang Tailor
Computer Science Department,
K.J.Somaiya College of Engineering,
Mumbai, Maharashtra, India

Sahil Ajmera
Computer Science Department,
K.J.Somaiya College of Engineering,
Mumbai, Maharashtra, India

***Abstract:*** In this paper we propose a new encryption algorithm, which is called as HASCII Algorithm. This encryption algorithm is simple and fast. It provides security and limits the added time cost for encryption and decryption to as to not degrade the performance of a database system which at last also degrades the performance of the whole system. Encryption in database systems is an important topic for research, as secure and efficient algorithms are needed that provide the ability to query over encrypted database and allow optimized encryption and decryption of data. Clearly, there is a compromise between the degree of security provided by encryption and the efficient querying of the database, because the operations of encryption and decryption greatly degrade query performance. Results of a set of experiments validate the functionality and usability of the proposed algorithm.

***Keywords:*** Encryption, Database Security, Decryption, passwords, 2’s Complement, ASCII

## I. INTRODUCTION

Database security concerns the use of a broad range of information security controls to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network links) against compromises of their confidentiality, integrity and availability. It involves various types or categories of controls, such as technical, procedural/administrative and physical.

Databases can be treasure troves of sensitive information. They can contain customers' personal data, confidential competitive information, and intellectual property. Lost or stolen data, especially customer data, can result in brand damage, competitive disadvantage, and serious fines—even lawsuits. Many of today’s privacy mandates require protecting data at rest, and the database is an obvious place where data accumulates and is potentially accessible to range of business systems and users. Organizations can choose to encrypt data at the application level, the database level, or the storage level. Encryption at the lowest of these levels, the storage level—on the disk or tape—guards against risk in the case where storage media are lost, but it does little to protect against malicious insiders or systems infected by malware . Application-level encryption on the other hand represents the other extreme by providing the highest degree of control, but it may not always be a viable approach. Because of these tradeoffs, many organizations are increasingly turning to database encryption as offering the best of both worlds when it comes to protecting data at rest—the protection goes further than storage level encryption and also avoids widespread changes in the application layer. [1]

As corporate networks become more and more open to the outside to accommodate suppliers, customers and partners, network perimeter security is no longer sufficient to protect data. Industry experts have long recommended a “defense in depth” approach by adding layers of security around the data. With the network being regarded as inherently insecure, encrypting the data it is the best option, often cited as the “last line of defense”. In terms of database security, encryption secures the actual data within the database and protects backups. That means data remains protected even in the event of a data breach. [2]

Encryption mechanism can prevent users from obtaining data in an unauthorized manner. Encryption mechanism can verify the authentic origin of a data item. Encryption mechanism also prevents from leaking information in a database when storage mediums, such as disks, CD-ROM, and tapes, are lost. When data is stored in the form of cipher, we have to decrypt all the encrypted data before querying them. For this purpose, we put forward the innovative encryption algorithm, known as “HASCII Algorithm”. Our new encryption algorithm is efficient and reliable. It has accomplished security requirements and is fast enough for most widely used software. This encryption algorithm limits the added time cost for encryption and decryption and at the same time improves the performance of the query over encrypted database. We also provide a thorough description of the proposed encryption algorithm and its processes.[3]

## II. ALGORITHM

. HASCII Algorithm is specially designed for storing passwords in encrypted format. In this encryption algorithm we require username and password for encrypting password. To Increase confusion Username is appended into password in a specific manner so that attacker can’t get to password easily Figure-1 shows how Encryption and Decryption takes place in the HASCII Algorithm.[4]

## A. *Encryption :*

Figure-2 Shows steps followed in encryption. Steps are explained below.

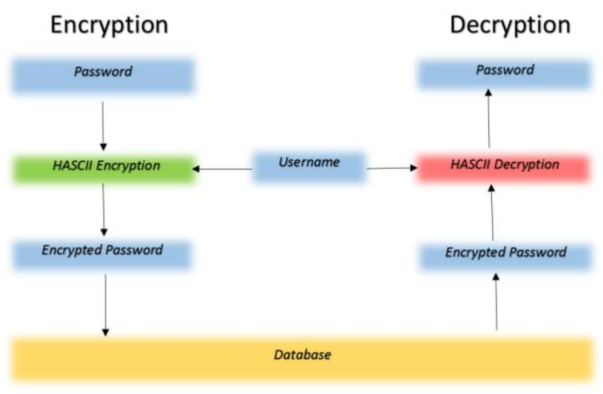

Figure 1. Encryption and Decryption in HASCII algorithm

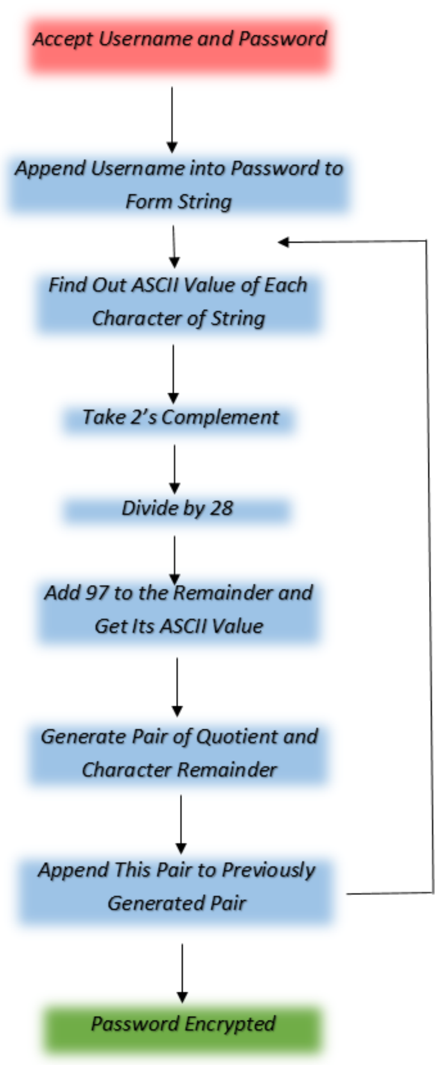
Accept Username and Password
Append Username into Password to Form String
Find Out ASCII Value of Each Character of String
Take 2's Complement
Divide by 28
Add 97 to the Remainder and Get Its ASCII Value
Generate Pair of Quotient and Character Remainder
Append This Pair to Previously Generated Pair
Password Encrypted

Figure 2. Encryption in HASCII algorithm

- Accept username and password from the user
- Generate a String with a Combination of username and password.
  - Accept username
  - Append Password to the Username only till length equal to that of the username.
  - If length of the appended password is less than that of the username, make it equal by inserting blank spaces.
  - If all characters of the password are not appended, repeat 1,2 and 3 else repeat 1
- From the String Generated in Step 2 take a character.
- Find out ASCII Value of that Character.
- Take 2's complement.
- Divide that ASCII Value with 28.
- Add 97 and convert that integer value to a Character according to ASCII.
- Form pair of quotient and generated character.
- Append generated pair to String.
- Repeat from Step 3 to Step 8 till all characters are read.
- Exit, as encrypted password is obtained and stored in the database.

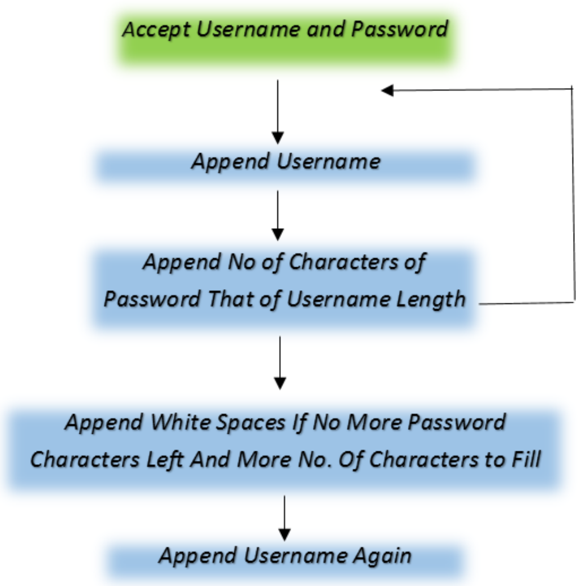
Accept Username and Password
Append Username
Append No of Characters of Password That of Username Length
Append White Spaces If No More Password Characters Left And More No. Of Characters to Fill
Append Username Again

Figure 3. String Formation in HASCII algorithm

## B. *Decryption :*

Figure-4 Shows decryption algorithm and steps followed is explained below.

- Accept encrypted password from the database.
- Accept a pair of characters i.e. quotient and remainder which are not yet read.
- Multiply quotient with 28 and add remainder.
- Calculate 2's complement and convert that integer value to Character according to ASCII.

- Repeat from Step 2 to Step 4 till all pairs of quotient and remainder are accepted or read. And a String is obtained
- As the username is known, accept those characters from the string excluding username and blank spaces from left to right.
- Exit as password is decrypted.

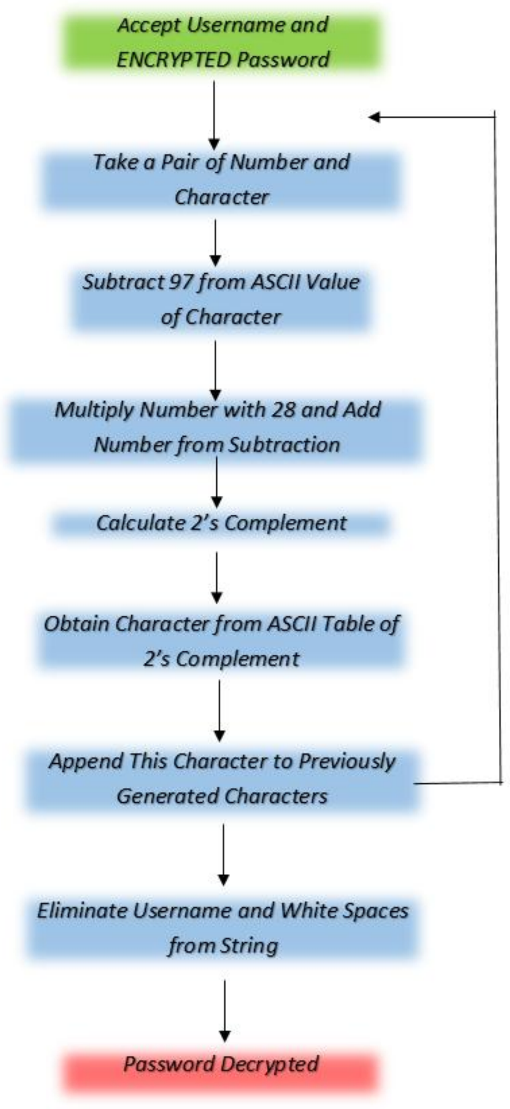
Accept Username and ENCRYPTED Password
Take a Pair of Number and Character
Subtract 97 from ASCII Value of Character
Multiply Number with 28 and Add Number from Subtraction
Calculate 2's Complement
Obtain Character from ASCII Table of 2's Complement
Append This Character to Previously Generated Characters
Eliminate Username and White Spaces from String
Password Decrypted

Figure 4. Decryption in HASCII algorithm

## III. EXAMPLE

Let Username Ram and Password HasciiAlgo in the following way it's encrypted.

### *A. Encryption :*

- Username: Ram
  Password: HasciiAlgo
- String Formation
  - Username Length: 3
  - Password Length:10
  - String:RamHasRamciiRamAlgRamo##Ram
  - # indicated blank spaces
- First Character is R
- ASCII Value of R:82
- 2's Complement:175
- Division by 28
  - Quotient: 6
  - Remainder: 7
- Add 97 i.e. Remainder 104
- Equivalent Character: 'h'
- Generated Pair: 6h
- Repeat above process till last character.
- Encrypted Password:6h5u5i6r5u5c6h5u5i5s5m5m6h5u5i6y5j5o6h5u5i5g8b8b6h5u5i

### *B. Decryption :*

- Username: Ram
- Encrypted Password:6h5u5i6r5u5c6h5u5i5s5m5m6h5u5i6y5j5o6h5u5i5g8b8b6h5u5i
- Take First Pair that is 6h
- ASCII Value of h:104
- Subtract 97 i.e. 7
- Now multiply 28 i.e. (6*28) + 7 = 175
- Get 2's Complement i.e. 82
- Convert to Character using ASCII that is 'R'.
- Repeat above process till last pair
- Remove Username and blank spaces.
- Decrypted Password: HasciiAlgo

## IV. IMPLEMENTATION

The following implementation is done in C#.

```
using System;
using System.Collections.Generic;
using System.Linq;
public class HASCIIAlgo
{
  private String pass, user;
  private String encrypt, decrypt;
 public HASCIIAlgo(String u, String p)
    {
       pass = p;
       user = u;
    }
    public HASCIIAlgo(String u, String e, int n)
    {
       user = u;
       encrypt = e;
    }
// Construction of String-Encryption
private void Form()
    {
       String s = "";
       char[] c = pass.ToCharArray();
       int i = 0;
       for (; i < c.Length; i++)
       {
          if ((i % user.Length) == 0)
          {
             s = s + user;
          }
          s = s + c[i];
       }
       while ((i % user.Length) != 0)
       {
          s = s + " ";
```

```
        i++;
      }
      s = s + user;
      encrypt = s;
   }
//Encryption Algorithm
   private void Encrypt()
   {
      Form();
      String s="";
      char[] c = encrypt.ToCharArray();
      for (int i = 0; i < c.Length; i++)
      {
         int temp = (int)c[i];
        temp = 257 - temp;
         int q = temp / 28;
         int r = temp % 28;
         r = r + 97;
         s = s + q;
         char x = (char)r;
         s = s + x;
      }
      encrypt = s;
   }
//Decryption Algorithm
private void Decrypt()
   {
      char[] c = encrypt.ToCharArray();
      String s = "";
      for (int i = 0; i < c.Length; i++)
      {
         int q = (int)c[i];
         q = q - 48;
         i++;
         int r = (int)c[i];
         r = r - 97;
         int temp = (q * 28) + r;
         temp = 257 - temp;
         char c1 = (char)temp;
         s = s + c1;
      }
      decrypt = s;
      Construct();
   }



//Construction of Decrypted password
   private void Construct()
   {
      String s = "";
      char[] c = decrypt.ToCharArray();
      for (int i = 0; i < c.Length; i++)
      {
         if ((i % user.Length) == 0)
         {
            i++;
            while ((i % user.Length) != 0)
               i++;
         }
         if (i >= c.Length)
            break;
         else if (c[i] == ' ')
            continue;
         else if(i<c.Length)
            s = s + c[i];
      }
      decrypt = s;
   }
// Get Encrypted Password
public String Get_Encrypted_Value()
   {
      Encrypt();
      return encrypt;
   }
//Get Decrypted Password
  public String Get_Decrypted_Value()
   {
      Decrypt();
      return decrypt;
   }
}
```

## V. ADVANTAGES AND DISADVANTAGES

### A. *Advantages :*

- The “HASCII” algorithm is fast and simple for encrypting and decrypting the messages. It provides maximum security and limits the added time cost for encryption and decryption.
- It has accomplished security requirements and is fast enough for most widely used software.
- It has a variable key-length which offers better security.
- The positions where the key would be used is not fixed (dynamic). Hence it offers confusion which in-turn increases the security of the system.
- The encryption algorithm contains simple arithmetic operations which are easy to realize and implement.
- It reduces the time cost for encryption/decryption process as compared to algorithms like DES, AES, blue-fish, etc.
- Even if by some means, the attacker is successful to intrude into the database, the attacker will not be able to get any useful information unless and until the attacker know the key and the process by which the encryption has taken place.

### B. *Disadvantages :*

- The length of the cipher text is not fixed and it directly depends on the length of the plain-text and the key length.
- The cipher text occupies more memory as compared to the memory required for storing the plain text. Hence, it increases the memory requirement of the database.
- The overhead of range queries over encrypted database is much higher than the overhead of range queries over plaintext database.
- Also, if the length of the plaintext is less than the length of the key used, then it results in adding some escape characters in between, which leads to un-necessary increase in the size of the cipher text.
- Also, there is a need for in-between processing for converting the plain text to cipher text or converting the cipher text to plain text. This causes an increased overhead for processing. Though the time for performing this process is much less as compared to

some other existing algorithms, still it requires a sufficient amount of time.

- If careful observation is made on the cipher text, then it can be found that there are some repeating patterns of text which are nothing but the encryption of the keys. If the attacker gets to know the algorithm used then there is a possibility of information prediction. Although it is a very tough task, still it is not impossible to break the algorithm.

## VI. CONCLUSION

HASCII algorithm takes some intermediate processing time for encryption and decryption purpose, but it also offers a very good and efficient security to the data which is needed to be protected. So, it involves a trade-off between the time performance and security performance factor.

## VII. ACKNOWLEDGMENT

Through this acknowledgement, we express our sincere gratitude to all those people who have been associated with this project and helped us with it and made it a worthwhile experience. Firstly we extend our thanks to the various people who have shared their Signatures. Our deepest thanks to Ms. Suchita Patil, the guide of our project for guiding us with attention and care. Also, we would like to express our gratitude to Prof. Swati Mali Deshpande and Prof. Pallavi Kulkarni for helping us. We would also thank our institution and our faculty members.

## VIII. REFERENCES

[1] Bouganim L. and Pucheral P., “Chip-SecuredData Access: Confidential Data on Untrusted Servers,” in Proceedings of the 28th International Conference on Very Large Data Bases, China,pp. 131-142, 2002.

[2] Chen G., Chen K., and Dong J., “A Database Encryption Scheme for Enhanced Security and Easy Sharing,” in Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design, Nanjing pp. 1-6, 2006.

[3] Coppersmith D., “The Data Encryption Standard and Its Strength Against Attacks,” IBM Journal of Research and Development, vol. 38, no. 3, pp. 243-250, 1994.

[4] Daemen J. and Rijmen V., “Rijndael: The Advanced Encryption Standard,” Dr. Dobb's Journal, pp.137-139, 2001.