IPASJ International Journal of Computer Science (IIJCS)
Web Site: http://www.ipasj.org/IIJCS/IIJCS.htm
A Publisher for Research Motivation ..... Email: editoriijcs@ipasj.org
Volume 2, Issue 8, August 2014 ISSN 2321-5992

Securing Database passwords using a combination of hashing and salting techniques

[1]Nicholas Oluwole Ogini , [2]Noah Oghenefego Ogwara
Department of Mathematics and Computer Science
Delta State University, Abraka, Delta State

ABSTRACT

Recently, in most organizations, there have been a number of high profile security breaches and prominent among this is the theft of database of users' passwords. This has become an area of interest for crackers whose interest has shifted to popular websites. Unfortunately, many consumer database available online, store their passwords either as plain text or in an encrypted form to which crackers have found ways around due to the proliferation of passwords decrypters online. In this paper, we focus on database passwords security, using a strong hashing algorithm and salting. This provides a platform to eliminate the need to ever persist user password in plain text or easy to know any users' password. This system is implemented using the message digest 5 (MD5) algorithms for hashing and a salt pattern with the dot Net (.NET)framework using C#.Net 2010 and the Microsoft SQL Server 2008.

Keywords:- hashing, salting, crackers, decrypters

1. INTRODUCTION

Passwords have no strong reputation among security professionals, but they are nevertheless the most common means of user authentication on the Web. There are alternatives to passwords that are deemed more secure, such as hardware tokens that generate time-dependent pass codes, or SSL/TLS(Secure Socket Layer/Transport layer secure) client certificates [5]. These alternatives, however, require the deployment and maintenance of cumbersome infrastructures. Tokens have to be distributed to users and sometimes resynchronized. Client certificates have to be signed by a certificate authority (CA), and certificate revocation lists have to be maintained. No alternative can compete with passwords in terms of convenience, especially for short-lived on-demand applications. With the advent of computer technology, it became more productive to store information in databases instead of storing in paper documents. Web applications, needing user authentication, typically validate the input passwords by comparing them to the real passwords, which are commonly stored in the company's private databases. If the database and hence these passwords were to become compromised, the attackers would have unlimited access to these users' personal data [2]. A hash function is a one-way encryption function that takes a variable-size input plaintext m and generates a fixed size hash output. It is computationally hard to decipher the hash and any attempt to crack it is practically infeasible. A "secure" hash function should be able to resist pre-image attacks and collision attacks. Due to the pigeonhole principle and birthday paradox, there will be some inputs that will produce the same hash result. The output length is of fixed size 128 bits, making a total of 2128 possible output hash values. This value, as big as it may seem, is not infinite, hence resulting in collisions [2]. Computer applications may require random numbers in many contexts. Random numbers can be used to simulate natural or artificial phenomena in computer simulations, many algorithms that require randomness have been developed that outperform deterministic algorithms for the same problem, and random numbers can be used to generate or verify passwords for cryptography-based computer security systems. The present invention relates to the use of random numbers in such security systems, called as cryptographic applications. Specifically, the present invention pertains to generating a random number in a secure manner for such cryptographic applications. In the context of cryptographic applications, there may be a hostile trespasser or agent, who desires to infiltrate the security of cryptographic security system in order to gain access to sensitive, confidential, or valuable information contained therein. For example, banks often encrypt their transactions and accounts [3]. One common method for circumventing a cryptographic application is to guess at potential passwords or cryptographic key, which are then submitted on a trial basis. This process is repeated until, by happenstance, the valid password is chanced upon. Fortunately, this process is extremely time consuming and inefficient. Also, preventative action can be taken to render this type of attack highly ineffective. However, if the random number generator used in generating valid passwords is somehow flawed in any way, the hostile agent can potentially take advantage of this flaw to circumvent the security of the system. For instance, a security system based on English text passwords, are susceptible to a dictionary attack. Thus, in order to ensure the utmost security, it is essential that the security system implements a method for generating a random number that appears completely random. In this manner, a completely random password or cryptographic key presents no opening or prior knowledge that can be exploited by an hostile agent [1]

IPASJ International Journal of Computer Science (IIJCS)
Web Site: http://www.ipasj.org/IIJCS/IIJCS.htm
A Publisher for Research Motivation ..... Email: editoriijcs@ipasj.org
Volume 2, Issue 8, August 2014 ISSN 2321-5992

Full Name	Email Address/Username	Password
Ogwara Noah Oghenefego	Fegwara@yahoo.com	k468dD8F
Ogini Nicholas Oluwole	ognino@yahoo.com	8Fk4lVQ0
Peter okafor	Peter203@gmail.com	8FkDDQ0
Samson Ojo	Ojo45@yahoo.com	56lkV#p6

SOME CHALLENGES OF THE EXISTING SYSTEM

Storing of Passwords in a Plain Text

Most database applications usually store their password in the database as plain text which is not a good practice, especially for those application that hold sensitive data of user. Plain text password, offer the flexibility for the application for easy Authentication (checking that the username and password pair matches the pair in the table). This is very simple – just compare the strings. Forgotten passwords can be retrieved – the password is easily accessible, given the user name.

But the Major challenges of this include the following.

- **First**, anyone with access to that file, (or able to SELECT from the table) gains immediate access to all passwords. An employee with legitimate access to the file might print the file or email out the information. Therefore, all the passwords are compromised. Most SQL software support encryption, however SQL encryption is not strong enough. The built-in encryption only protects the information on disk. If a user is allowed to access the data (perform a SELECT), SQL will automatically decrypt the information. If your web application is allowed to access the data, then a hacker hacking your application can access the data as well, gaining the same access as your web application.
- **The second** problem is that during the authentication exchange, the password is visible on the network. Unless secure communication is used throughout, the password can be seen while traveling on the network. For example, even if the web application uses SSL to submit the password, the password is still visible when the web application server SELECTs the information from the remote database. The results of the query are transferred unencrypted over the network.

Table 1: a sample of Passwords stored as Plain text

Full Name	Email Address/Username	Password
Ogwara Noah Oghenefego	Fegwara@yahoo.com	Fegi2013
Ogini Nicholas Oluwole	ognino@yahoo.com	Nigeria
Peter okafor	Peter203@gmail.com	Psquare
Samson Ojo	Ojo45@yahoo.com	Ojo3663

Encrypting passwords – a little more secure

A better approach for storing passwords (and the only viable alternative if users need to be able to recover passwords) is scrambling the passwords before storing them. This approach relies on having a secret. The secret is either the scrambling algorithm, or the key used in conjunction with a modern encryption algorithm. Scrambling (or encrypting) passwords is a reversible operation. The secret is used to garble the password, and the same secret can be used to retrieve the original password. When a user supplies a password, the stored password is de-scrambled using the secret, and the passwords are compared. An alternative approach is to scramble the provided password using the secret and compare the two garbled versions – a match indicates the provided password was good. If a user needs to retrieve a password, the stored password is de-scrambled and provided to the user (usually via email). Encryption as a means of securing password in the database is better than plain text password such that a forgotten or lost password can be retrieved. Only a single secret needs to be stored securely (either an algorithm or a key) and for a multi-user distributed applications, when using encryption, either the clear text password needs to be communicated (to be checked), or the secret must be communicated to perform the authentication on the front end. Although encryption a means of passwords security, it has some critical issues especially if the secret is compromised, all passwords are compromised. If somebody has access to the secret and to the password store, all passwords can be unscrambled, and access to the password store alone is sufficient to provide information about the passwords. Since all passwords are scrambled using the same algorithm, if two users share the same scrambled password, they must have the same password as well. Crafty hackers with access to the password store can create users with known passwords and check for other users with the same password. This type of attack is a variation of the 'known plaintext' attack. Attacks such as this can be stopped using a SALT. When using a block cipher, the password length must be stored as part of the scrambled password. The length must be stored because block ciphers always produce a fixed-size block of scrambled text. If the password length

IPASJ International Journal of Computer Science (IIJCS)
Web Site: http://www.ipasj.org/IIJCS/IIJCS.htm
A Publisher for Research Motivation ..... Email: editoriijcs@ipasj.org
Volume 2, Issue 8, August 2014 ISSN 2321-5992

is not scrambled (for example, if stored as a column in a table), the information is very useful to password crackers. Knowing the exact length of the password makes it much easier to try to guess a password.

MATERIALS AND METHODS

A cryptographic hash is also known as a 'one-way function'. A hash function takes input of any length, and produces a unique output of constant length. For example, if we hash a password (of any length) with the MD5 cryptographic hash, the result would be a 128 bit number that uniquely corresponds to the password. Cryptographic hashes work on more than passwords, if the cryptographic hash of two files is identical, then the two files are identical as well. When storing hashed passwords, the password is hashed (run through the hashing algorithm) and the resulting hash is stored instead of the password. To compare passwords, just hash the given password using the same hash function and compare the results. If the hashes are the same, the passwords match. The beauty of a one way function is that there is no way to compute the password based on the hash. Hashed passwords are not immune to brute force attacks – given a dictionary and the password hash, a hacker can compute the hashes of all the words in the dictionary, compare the words with the password hash, and discover the password. This is where strong passwords (containing letters, numbers, and special characters) help us defend against brute-force attacks. The application was implemented using .net framework with C#.net as the language of implementation

Methodology

We present a system that will secure database password using hashing algorithm and salting techniques. We create a simple application in .net framework 4.0 using Visual C# as the language of Implementation of securing database passwords. Password hashing still possess some weaknesses, this paper discusses the processes involved in making a hash password more secure by using the salting pattern. The salting technique method use their username as the salt which is different for each user, the salt can also be in the form of a random number generate for each user when storing their password in the database. We used the MD5 (Message Digest) Algorithm for our hashing by introducing a salt to the password which will make it different from every user hashed passwords even though they are using the same password. The hashed salted password cannot be rehashed to get the password which makes it more secure than that of encryption which is a 2 way process while hashing and salting is a one way approach.

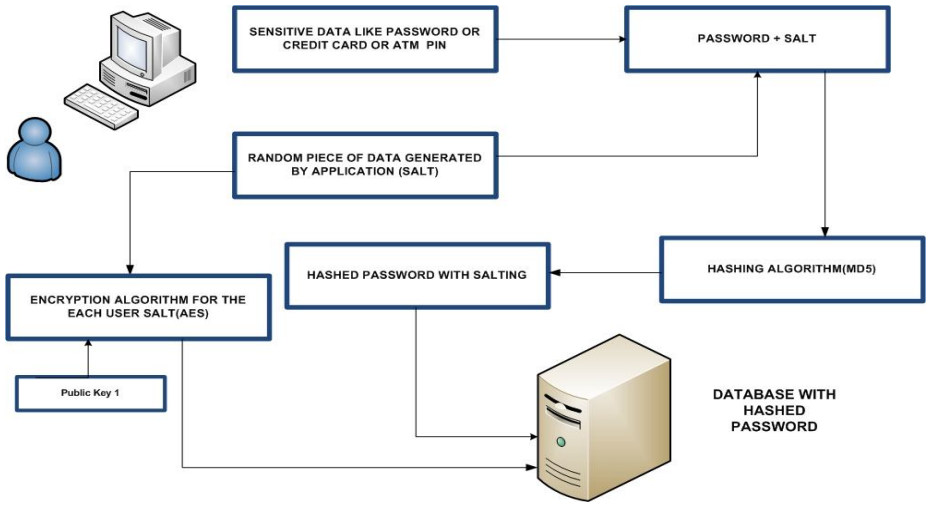

Figure 1: Architecture of the proposed System using Hashing and Salting techniques for Password Storage.

MD5 Algorithm

According to [4], supposing that we have a b-bit message as input, and that we wish to find its message digest. Here, b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1}

The following five steps are performed to compute the message digest of the message.

1) Step1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

2) Step2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.) At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

3) Step3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10

4) Step4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased. The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs. This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Do the following

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do
/* Copy block i into X. */
 For j = 0 to 15 do
Set X[j] to M[i*16+j].
end /* of loop on j */
/* Save A as AA, B as BB, C as CC, and D as DD. */
AA = A
BB = B
CC = C
DD = D
/* Round 1. */
/* Let [abcd k s i] denote the operation
a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]
/* Round 2. */
/* Let [abcd k s i] denote the operation
a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
```

[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]
/* Round 3. */
/* Let [abcd k s t] denote the operation
a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]
/* Round 4. */
/* Let [abcd k s t] denote the operation
a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]
/* Then perform the following additions. (That is increment each of the four registers by the value it had before this block was started.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD
end /* of loop on i */

Sample Form For Create User Profile

This forma enable us to capture information including their passwords and store only the hash values in the database.

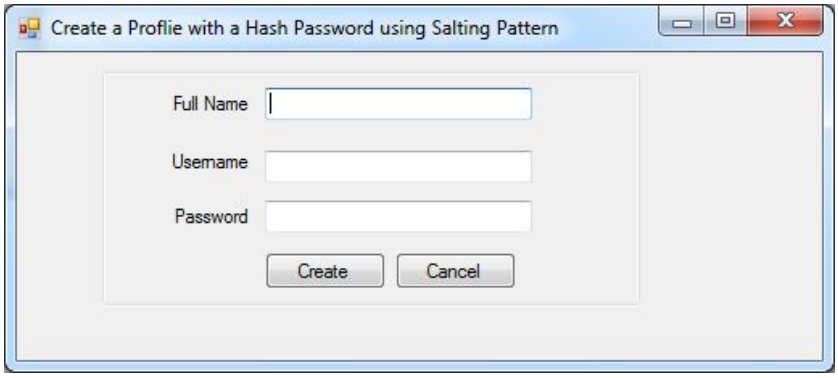

Figure 2: Form that Compute the user password and produce the hashed value.

Sample Source Codes

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Security.Cryptography;
using System.Data.SqlClient;
```

Module 1 To Compute the Hash of Password and its randomly generated salt

```
public static string GetMD5Hash(string plaintext,string salt)
{MD5CryptoServiceProvider MD5provider = new MD5CryptoServiceProvider();
 byte[] hasedvalue = MD5provider.ComputeHash(Encoding.Default.GetBytes(plaintext));
 StringBuilder str = new StringBuilder();
```

```
 for (int counter = 0; counter < hasedvalue.Length; counter++)
 {str.Append(hasedvalue[counter].ToString("x2"));}
return str.ToString();
}
```

MODULE 2 To Compare Hashed Password with the Store Hashes in the database

```
static bool VerifyMD5hash(string PlainText, string prevhashedvalue, string salt)
{string hashedvalue2 = GetMD5Hash(PlainText,salt);
StringComparer strcomparer = StringComparer.OrdinalIgnoreCase;
if (strcomparer.Compare(hashedvalue2, prevhashedvalue).Equals(0))
{
  return true;
        }
else
    {
        return false;
         }
                }
```

MODULE 3: CREATING A USER ACCOUNT WITH HASHED PASSWWORD AND STORE IN THE DATABASE

```
private void btnCreate_Click(object sender, EventArgs e)
{
    string pwd;
    pwd = textBox3.Text;
    string username;
    username = textBox2.Text;
    string salt;
    salt = username.ToLower();
    pwd = pwd + salt;
    string password = pwd;
    SqlConnection cn = new SqlConnection();
    cn.ConnectionString = "Data Source=FEGWARA-PC;Initial Catalog=TestDbase;Persist Security
Info=True;User ID=sa;Password=f1e2g34real";
    cn.Open();
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = cn;
    cmd.CommandText = "Insert into tblLogTest(FullName,Username,Password) values
(@FullName,@Username,@Password)";
     cmd.Parameters.AddWithValue("@Fullname", textBox1.Text);
     cmd.Parameters.AddWithValue("@Username", textBox2.Text);
     cmd.Parameters.AddWithValue("@Password", GetMD5Hash(password,salt));
     cmd.ExecuteNonQuery();
     cmd.Parameters.Clear();
     MessageBox.Show("New user Information Sucessfully added to the database");

  }
```

RESULTS AND DISCUSSION

Below are sample of result of hashed password generated from our system which shows that even if two users have the same passwords those the hashed value shown in the table can never be the same.

IPASJ International Journal of Computer Science (IIJCS)
Web Site: http://www.ipasj.org/IIJCS/IIJCS.htm
A Publisher for Research Motivation ..... Email: editoriijcs@ipasj.org
Volume 2, Issue 8, August 2014 ISSN 2321-5992

Table 2: Details of hashed passwords using the hashing and salting pattern

ID	FullName	Username	Password
1	Ogwara Noah Oghenefego	fegwara	383fcabc41db01c4cdd77aabf974e5b6
2	Ogini Nicolas Oluwole	Ogini	99ae023721da9d71c34692ab49a8dcf7
3	Peter Okafor	Okafor	9fcf5794a70b38e371d1bdd8abb3be10
4	Chukwuma	Patricia Nwamaka	1b9ec41d7b2a1b076aa354292cf70aa7
5	Emojorho	Joyce Ogheneseme	2abe4377648155b52272cea92a092603
6	Amos Faith	Faith	98595d11f45eb11767782ea6b1755428
7	Onwuma Grace	Grace	4ab1b464c8ae5262defea33e3d8ebde5
8	Bola Tinibu	Bola2013	aa14bf6b0b31b3a208ea5623a7c41c77
9	Nathaniel Mercy	Mercy	432e6610d3328578723db18ad48255e0
10	Bointa Godwin	Junior	e30d9909f5b923abb1546f85f2a95029

From the result shown above database password security using hashing and salting pattern provide a stronger security such that the original 'clear text' password is never stored. Even if the password store is compromised, only the hashes become public. The password length is not stored and cannot be estimated, making password cracking that much harder. There is no need for a **secret** as none is used to hash the password. For multi-user distributed applications, the password hash can be used for authentication. When using encryption, either the clear text password needs to be communicated (to be checked), or the secret must be communicated to perform the authentication on the front end.

2. CONCLUSION

Database Password security is one important aspect of data security as most systems require an authentication method using passwords. Hashing algorithms such as MD5 used for encrypting plaintext passwords into strings that theoretically cannot be deciphered by hackers due to their one-way encryption feature. However, in order to prevent hackers that used password guess attacks, We introduce the salting pattern such that each password is hashed with a salt which can be in the form of random numbers generated during the process of calculating the hashed function or using some parts of their logon details which is the salt that is not know even to the user password to reduce the incident of being cracked and made stronger compared with that of encryption which is a two way process. An implementation of the proposed solution is carried out using C# as programming language and Microsoft SQL Server 2008 as database.

REFERENCES

[1] J. A. Halderman, B.Waters, and E. Felten "A convenient method for securely managing passwords" To appear in Proceedings of the 14th International World Wide Web Conference (WWW 2005), 2005.
[2] Mary Cindy Ah Kioon, ZhaoShun Wang and Shubra Deb Das "Security Analysis of MD5 algorithm in Password Storage" Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13), 2013
[3] N. Chou, R. Ledesma, Y. Teraguchi, and J. Mitchell, "Client-side defense against web based identity theft ", In Proceedings of Network and Distributed Systems Security (NDSS), 2004.
[4] Rivest, R. The MD5 message-digest algorithm. RFC 1321, 37 (April 1992).
[5] Zhang Shaolan, Xing Guobo, Yang Yixian, Improvement and Security Analysis on MD5 [J]. Computer Application, 2009, vol. 29(4):947-949.

AUTHOR

Ogini Oluwole Nicholas, Diploma in Electrical/ Electronic Engineering, Auchi Polytechnic, and B.Sc, M.Sc. and Ph.D. in Computer Science from University of Benin. Now a lecturer at Delta State niversity, Abraka, Nigeria.

Ogwara Noah Oghenefego OND (Computer Science) Delta State Polytechnic, Ozoro B.Sc(Computer Science) Delta State University,Abraka. Msc In View (Computer Science) Nnamadi Azikiwe University, Awka. Now a lecturer at Delta State University, Abraka, Nigeria.