Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 46 (2015) 820 – 832

Procedia Computer Science

International Conference on Information and Communication Technologies (ICICT 2014)

A Novel String Matrix Data Structure for DNA Encoding Algorithm

Noorul Hussain UbaidurRahman[a]*,Chithralekha Balamurugan[b],Rajapandian Mariappan[ab]

[a]*Dept. of Banking Technology, Pondicherry University, Puducherry – 605014, India*
[b]*Dept. of Computer Science, Pondicherry University, Puducherry – 605014, India*
[ab]*Dept. of Mathematics & Computer Science, KMCPGS, Puducherry – 605008, India*

Abstract

DNA Cryptography is a rapidly emerging field of DNA Computing research to provide invincible cryptographic technique for the modern cyber world of today and the futuristic DNA computers. Several DNA based cryptographic algorithms are proposed for encryption, decryption and authentication, etc. The first and foremost step of DNA based encryption is DNA encoding of plaintext. The main limitation of DNA encoding is the absence of effective, randomized, dynamic, secure DNA Encoding technique for DNA encoding of plaintext. To overcome this limitation, this paper describes a novel DNA encoding algorithm. This encoding algorithm is based on a novel string matrix data structure, for generating the unique DNA sequences that can be used to encode plain text (comprising of alphabets, numbers and special characters) as DNA sequences. The experimental results and comparison results have proved that the proposed DNA encoding algorithm is more effective than the existing DNA encoding algorithms.

Keywords: Datastructure; String Matrix; String Matrix Operations, Cryptography; DNA Computing; DNA Cryptography.

1. Introduction

Secure communication is vital to facilitate confidential exchange of information between sender and receiver. With the internet becoming the media for all banking and electronic commerce transactions, it is essential that the

* Corresponding author. Tel.: +91-9944981398
E-mail address: unoorulhussain@gmail.com

communication is made highly secure. The field of cryptography has rendered many cryptographic algorithms for the secure communication of data across the open internetwork. A lot of techniques and systems has developed in the mathematical cryptography for encoding and decoding the plaintext. However, these techniques are broken using DNA Cryptography Techniques and methods. The DNA Cryptography is an emerging field in the area of DNA Computing research. It plays major role in the next generation security. The first and foremost step in DNA cryptography is encoding the plain text to DNA alphabets following which the cryptography techniques actually convert the DNA alphabets to cipher text. The methods available for encoding of plaintext available in DNA cryptography are not as much effective as they are manually computed (which are very easy to break) and not inclusive of the complete character set. In this paper, a novel DNA encoding algorithm which is based on a novel string matrix data structure is described. The paper also provides experimental results that endorses the performance of the algorithm.

The rest of this paper is organized as follows Section 2 describes the related works Section 3 describes the requirements to be fulfilled by DNA encoding algorithm. Section 4 describes the proposed algorithm. In Section 5, the experimental results are presented. Section 6 shows how the proposed algorithm fulfills the required security features. Section 7 gives the conclusion of this work.

2. Related Works

The various DNA encoding techniques and their limitations are addressed in the following literature. These works are compared against the properties required to be fulfilled by the encoding algorithms. This would help to understand how well these existing algorithms fulfill the required properties – and thereby understand the limitations.

Encoding of plaintext to DNA sequence is found to be achieved in two ways:

2.1. Binary to DNA Encoding of Plaintext

In this approach the plaintext is converted into binary form. The binary form of the plain text is transformed into DNA Sequence using an example encoding scheme as follows A-00 T-01 C-10 G-11. Different works use different encoding schemes for converting binary into DNA sequence as shown in Table1.

2.2. Nucleotide based Encoding of Plain Text

In this approach, the plain text alphabets are converted into an 'n' alphabet DNA sequence using a predefined encoding table. This table has predefined nucleotide sequences consisting of 'n' number of DNA alphabets for every plain text alphabet. The encoding table is manually defined and there is no systematic methodology followed for generating this table. The existing works are as shown Table 1.

Table 1 Existing works in the different approaches of DNA Encoding of Plain Text and their Limitations

Sl.No	Authors	Method	Limitation
		BINARY TO DNA ENCODING OF PLAINTEXT	
1	Guangzhao Cui et.al[3]	Plain text to Hexadecimal to Binary to DNA sequence	
2	Li Xin-she[5]	Plaintext to ASCII to Binary to DNA sequence	
3	Qiang Zhang et.al[10]	Original Image to Binary to DNA sequence	
4	Souhila Sadeg et. al[11]	Plaintext to ASCII to Binary to DNA sequence	In these works, the plain text is converted to binary (ascii to binary) and these binary values are encoded using the following encoding rule or a variation of the same
5	Sherif T.Amin et. al[12]	Plaintext to ASCII to Binary to DNA sequence	

6	O.Tornea & M.E.Borda[8]	Plaintext to ASCII to Binary to DNA sequence	A-00 , T – 10, C – 01, G - 11
7	Zhang, Qiang et. al[16]	Plaintext to ASCII to Binary to DNA sequence	
8	Kang Ning[4]	Plaintext to ASCII to Binary to DNA sequence	
9	Mona Sabry et. al[6]	Plaintext to ASCII to Binary to DNA sequence	
		NUCLEOTIDE BASED ENCODING OF PLAINTEXT	
1	Padma Bh et. al[9]	Predefined Manual Table to DNA Sequence	
2	Xing Wang & Qiang Zhang[14]	Predefined Manual Table to DNA Sequence	The DNA Encoding table is prepared manually, there is no systematic methodology followed for generating the DNA encoding table.
3	Akanksha Agarwal et.al[1]	Predefined Manual Table to DNA Sequence	

3. Requirements to be fulfilled by DNA Encoding Algorithm

Every DNA encoding algorithm should fulfill a set of functional attributes. These attributes are listed in Table2 and Table3. The functional attributes have been identified in this paper based on the observed limitations in the existing encoding algorithms. The fulfillment of existing encoding algorithms as shown in Table 3.

Table 2 Definition of Functional Attributes to be fulfilled by Encoding Algorithm

Sl.No	Features	Definition
1	Robustness	The Encoding Technique is not breakable.
2	Confidentiality	The DNA encoding technique should help to encode the plain text in such a way that it is not possible to be deciphered.
3	Randomness	The concept of randomness suggest that a non-order or non-coherence in sequence of steps is required to achieve more randomness in each and every step of DNA encoding table generation so that the encoding table cannot be deciphered.
4	Dynamicity	The DNA encoding table generated should be dynamic across different sessions whereby every session between a sender and receives generates and uses a new encoding table.
5	Complete Character Set Fulfillment	The DNA encoding table should provide for DNA encoding sequences for the complete character set, which contains 96 elements.
6	Uniqueness	The encoding of plaintext into DNA sequence is unique in every generation of encoding table.

Table 3 Fulfilment of Functional Characteristics of the existing encoding algorithms

Authors	Robustness	Confidentiality	Randomness	Dynamicity	Complete Character set Fulfillment	Uniqueness
			BINARY TO DNA ENCODING OF PLAINTEXT			
Guangzhao Cui et.al	×	✓	×	×	×	×
Li Xin-she	×	×	×	×	×	×
Qiang Zhang et.al	×	×	×	×	NA	×
Souhila Sadeg et. Al	×	×	×	×	×	×
Sherif T.Amin et. Al	×	×	×	×	✓	×
O.Tornea & M.E.Borda	×	×	×	×	✓	×
Zhang, Qiang et. Al	×	×	×	×	NA	×

Kang Ning et. Al	×	×	×	×	×	×
Mona Sabry et. Al	×	×	×	×	×	×
			NUCLEOTIDE BASED ENCODING OF PLAINTEXT			
Padma Bh et. Al	×	×	×	×	×	×
Xing Wang & Qiang Zhang	×	×	×	×	×	×
Akanksha Agarwal et.al	×	×	×	×	✓	×

×- Indication of minimum level of supporting. ✓ - Indication of Acceptable Level of Supporting

4. Proposed Algorithm

DNA encoding of plain text is the initial step of any cryptographic algorithm. DNA alphabets ‘ATCG’ are used to encode the plaintext into DNA sequence. This would enable the given message sequence to be encoded as a DNA sequence which would be acted upon by the DNA based encryption algorithm. In order to generate this table, string matrix data structures along with its operations are defined. The use of this data structure to generate the DNA encoding table is described subsequently.

4.1. Definitions of String Matrix data structure

Definition 1
A **StringMatrix**, denoted by $A_p^{m,n}$, is defined as a matrix of m rows and n columns in which each cell shall hold a string of length p.
The Order of the StringMatrix is defined as mxn and the Strength of the StringMatrix is defined as p.

Notation:
We may denote it as simply A_p, explicitly stating only the common string-length of contents of the StringMatrix.
We may also denote the element stored in the ith row jth column of the StringMatrix $A_p^{m,n}$, by $a_{i,j}$.

Definition 2
A StringMatrix having the same number of columns as rows is called a Square StringMatrix.

4.1.1. Operations and Functions on String Matrices

Definition 3
The operation **ADD**: performed only on StringMatrices of the same order.

The function **ADD** ($\boldsymbol{C_{p+q}^{m,n}}$; $\boldsymbol{A_p^{m,n}}, \boldsymbol{B_q^{m,n}}$) yields $C_{p+q}^{m,n}$, where ∀ i,j $c_{i,j} = a_{i,j} + b_{i,j}$ = Concatenation of the strings $a_{i,j}$ and $b_{i,j}$.

Example:
If $A_1^{2,3} = \begin{vmatrix} A & A & A \\ T & T & T \end{vmatrix}$ and $B_1^{2,3} = \begin{vmatrix} C & C & C \\ G & G & G \end{vmatrix}$ then, ADD($C_2^{2,3}$; $A_1^{2,3} + B_1^{2,3}$)= $\begin{vmatrix} AC & AC & AC \\ TG & TG & TG \end{vmatrix}$.

Definition 4
The operation **MULT**: performed on a StringMatrix A_p of n columns and a StringMatrix B_q of n rows.

The function **MULT** ($\boldsymbol{C_{n\times(p+q)}^{m,r}}$; $\boldsymbol{A_p^{m,n}}$,$\boldsymbol{B_q^{n,r}}$), yields $C_{n\times(p+q)}^{m,r}$ where ∀ i,j, $c_{i,j} = \sum_{k=0}^{n-1}(a_{i,k} + b_{k,j})$, summation denoting the concatenation (i.e. the ith row jth column element of the resulting StringMatrix will be the concatenation of the strings which are the results of the concatenation of the corresponding elements in the ith row of $A_p^{m,n}$ and jth column of $B_q^{n,r}$).
Example:

If $A_1^{2,3} = \begin{vmatrix} A & A & A \\ T & T & T \end{vmatrix}$ and $B_1^{3,3} = \begin{vmatrix} C & G & A \\ C & G & A \\ C & G & A \end{vmatrix}$ then, $C_6^{2,3} = \begin{vmatrix} ACACAC & AGAGAG & AAAAAA \\ TCTCTC & TGTGTG & TATATA \end{vmatrix}$

Definition 5

The operation **JOINCOLUMNS:** performed only on StringMatrices having the same number of rows and having the same strength

The function ***Joincolumns*** $(\boldsymbol{C_p^{m,n+r}};\ \boldsymbol{A_p^{m,n}}, \boldsymbol{B_p^{m,r}})$, performed on StringMatrices $A_p^{m,n}$ and $B_p^{m,r}$, yields $C_p^{m,n+r}$, where∀i, $c_{i,j} = a_{i,j}$ for j < n and $c_{i,j} = b_{i,j-n}$ for j ≥ n.

Example:

If $A_1^{2,3} = \begin{vmatrix} A & A & A \\ T & T & T \end{vmatrix}$ and $B_1^{2,3} = \begin{vmatrix} C & C & C \\ G & G & G \end{vmatrix}$ then,

Joincolumns $(C_1^{2,6};\ A_1^{2,3}, B_1^{2,3})$ yields $C_1^{2,6} = \begin{vmatrix} A & A & A & C & C & C \\ T & T & T & G & G & G \end{vmatrix}$

Definition 6

The operation **JOINROWS**: performed only on StringMatrices having the same number of columns and having the same strength

The function ***Joinrows*** $(\boldsymbol{C_p^{m+r,n}};\ \boldsymbol{A_p^{m,n}}, \boldsymbol{B_p^{r,n}})$, performed on StringMatrices $A_p^{m,n}$ and $B_p^{r,n}$ yields $C_p^{m+r,n}$, where∀j, $c_{i,j} = a_{i,j}$ for i < m and $c_{i,j} = b_{i-n,j}$ for i ≥ m.

Example:

If $A_1^{2,3} = \begin{vmatrix} A & A & A \\ T & T & T \end{vmatrix}$ and $B_1^{2,3} = \begin{vmatrix} C & C & C \\ G & G & G \end{vmatrix}$ then,

Joinrows $(C_1^{4,3};\ A_1^{2,3}, B_1^{2,3})$ yields $C_1^{4,3} = \begin{vmatrix} A & A & A \\ T & T & T \\ C & C & C \\ G & G & G \end{vmatrix}$

Definition 7

The operation **SPLITCOLUMNS**: Splits a StringMarix vertically into two StringMatrices.

The function **Splitcolumns** $(\boldsymbol{B_p^{m,q}}, \boldsymbol{C_p^{m,n-q}}; \boldsymbol{A_p^{m,n}})$, for 0 <q < n, performed on StringMatrix $A_p^{m,n}$, yields $B_p^{m,q}$ and $C_p^{m,n-q}$, where∀i, $b_{i,j} = a_{i,j}$ for 0 ≤ j < q and $c_{i,j} = a_{i,j+q}$ for 0 < j < n-q.

Example:

If $A_1^{2,6} = \begin{vmatrix} A & A & A & C & C & C \\ T & T & T & G & G & G \end{vmatrix}$, then Splitcolumns $(B_1^{2,3}, C_1^{2,3}; A_1^{2,6})$ yields $B_1^{2,3} = \begin{vmatrix} A & A & A \\ T & T & T \end{vmatrix}$ and $C_1^{2,3} = \begin{vmatrix} C & C & C \\ G & G & G \end{vmatrix}$.

Definition 8

The operation **SPLITROWS**: Splits a StringMarix horizontally into two StringMatrices.

The function **Splitrows** $(\boldsymbol{B_p^{q,n}}, \boldsymbol{C_p^{m-q,n}}; \boldsymbol{A_p^{m,n}})$, for 0 <q < n, performed on StringMatrix $A_p^{m,n}$, yields $B_p^{q,n}$ and $C_p^{m-q,n}$, where∀j, $b_{i,j} = a_{i,j}$ for 0 ≤ i < q and $c_{i,j} = a_{i+q,j}$ for 0 < i < n-q.

Example:

If, $A_1^{4,3} = \begin{vmatrix} A & A & A \\ T & T & T \\ C & C & C \\ G & G & G \end{vmatrix}$, then Splitrows $(B_1^{2,3}, C_1^{2,3};\ A_1^{4,3})$ yields $B_1^{2,3} = \begin{vmatrix} A & A & A \\ T & T & T \end{vmatrix}$ and $C_1^{2,3} = \begin{vmatrix} C & C & C \\ G & G & G \end{vmatrix}$

Definition 9
The operation **ExtractRow**: Extracts a particular row of a StringMarix.

The function ***ExtractRow*** $(\boldsymbol{B_p^{1,n}};\ \boldsymbol{A_p^{m,n}, r})$ fills up the cells of $B_p^{1,n}$ with the values in the corresponding cells extracted from the rth row of $A_p^{m,n}$. i.e. ∀ j, $b_{1,j} = a_{r,j}$.

Example:

If, $A_1^{3,3} = \begin{vmatrix} A & T & C \\ T & C & G \\ C & G & A \end{vmatrix}$, then *ExtractRow* $(B_1^{1,3};\ A_1^{3,3}, 3)$ yields $B_1^{1,3} = |C \quad G \quad A|$.

Definition 10
The operation **RFLCTROWS:** gets the horizontal reflexion of a StringMatrix.

The function **RflctRows** $(\boldsymbol{B_p^{m,n}};\ \boldsymbol{A_p^{m,n}})$ yields $B_p^{m,n}$, the horizontal reflection of $A_p^{m,n}$, such that ∀ i,j $b_{i,j} = a_{n-1-i,j}$.

Example:

If, $A_1^{3,3} = \begin{vmatrix} A & T & C \\ T & C & G \\ C & G & A \end{vmatrix}$, then RflctRows $(B_1^{3,3};\ A_1^{3,3})$ yields $B_1^{3,3} = \begin{vmatrix} C & G & A \\ T & C & G \\ A & T & C \end{vmatrix}$.

Definition 11
The operation **RFLCTCOLS:** gets the vertical reflexion of a StringMatrix.

The function **Rflctcols** $(\boldsymbol{B_p^{m,n}};\ \boldsymbol{A_p^{m,n}})$ yields $B_p^{m,n}$, the vertical reflection of $A_p^{m,n}$, such that ∀ i,j $b_{i,j} = a_{i,n-1-j}$.

Example:

If, $A_1^{3,3} = \begin{vmatrix} A & T & C \\ A & T & C \\ A & T & C \end{vmatrix}$, then Rflctcols $(B_1^{3,3};\ A_1^{3,3})$ yields $B_1^{3,3} = \begin{vmatrix} C & T & A \\ C & T & A \\ C & T & A \end{vmatrix}$.

Definition 12
The operation **TRANSPOSE:** interchanges columns and rows of StringMatrix.

The function **Transpose** $(\boldsymbol{B_p^{n,m}};\ \boldsymbol{A_p^{m,n}})$ yields $B_p^{n,m}$, where ∀ i,j $b_{i,j}=a_{j,i}$.

Example:

If, $A_1^{4,3} = \begin{vmatrix} A & A & A \\ T & T & T \\ C & C & C \\ G & G & G \end{vmatrix}$, then Transpose $(B_p^{3,4};\ A_1^{4,3})$ yields $B_p^{3,4} = \begin{vmatrix} A & T & C & G \\ A & T & C & G \\ A & T & C & G \end{vmatrix}$

For DNA encoding over the alphabets A, T, C, G the Watson-complements are T, A, G, C respectively.
The operation

Definition 13
The operation **WCOMP:** (available only for DNA StringMatrices) replaces the values in all the cells with the corresponding Watson Complements.

The function **WComp** $(\boldsymbol{B_p^{m,n}};\ \boldsymbol{A_p^{m,n}})$, (available only for DNA encoded StringMatrices) yields $B_p^{m,n}$, the Watson complement of $A_p^{m,n}$, such that ∀ i,j $b_{i,j}$ = WC($a_{i,j}$) where WC("A") = "T", WC("T") = "A", WC("C") = "G", and WC("G") = "C".

Example:

If, $A_1^{4,3} = \begin{vmatrix} A & T & C \\ T & C & G \\ C & G & A \\ G & A & T \end{vmatrix}$, then WComp ($B_1^{4,3}$; $A_1^{4,3}$) yields $B_1^{4,3} = \begin{vmatrix} T & A & G \\ A & G & C \\ G & C & T \\ C & T & A \end{vmatrix}$,

4.1.2. Storage in a String matrix

The following operations and functions are defined on Strings of known lengths.

Definition 1

The operation **FILLSMAT:** fills up all the cells of a StringMatrix with some specified character.

The function **FillSMat ($\boldsymbol{A_1^{m,n}}$, "$\boldsymbol{X}$")** fills up the StringMatrix $A_1^{m,n}$, with the character "X" such that ∀ i,j $a_{i,j}$ = "X".

Example:

FillSMat ($A_1^{3,3}$, "A") yields $A_1^{3,3} = \begin{vmatrix} A & A & A \\ A & A & A \\ A & A & A \end{vmatrix}$.

Definition 2

The operation **STOROWSMAT:** fills up a StringMatrix, row-wise, with a character in each cell from some specified string in order

The function **StoRowSMat ($\boldsymbol{A_1^{m,n}, Seq}$)** yields $A_1^{m,n}$, filled up, row-wise, with a character in each cell from the string Seq in order. If the string is of length < mn, the string is repeated. If the string is of length > mn, the string is truncated. i.e. If strlen(Seq) < mn, ∀ i,j $a_{i,j}$ = Seq[k], where k = mod(i*n + j; strlen(Seq)) else ∀ i,j $a_{i,j}$ = Seq[k], where k = i*n + j.

Example:

If, Seq = "ATCGTCGACGATATCG" then StoRowSMat ($A_1^{4,3}, Seq$) yields $A_1^{4,3} = \begin{vmatrix} A & T & C \\ T & C & G \\ C & G & A \\ G & A & T \end{vmatrix}$.

If, Seq = "ATCGTCGA" then StoRowSMat ($A_1^{4,3}, Seq$) yields $A_1^{4,3} = \begin{vmatrix} A & T & A \\ T & C & T \\ C & G & C \\ G & A & G \end{vmatrix}$.

Definition 3

The operation **STOCOLSMAT:** fills up a StringMatrix, column-wise, with a character in each cell from some specified string in order

The function **StoColSMat ($\boldsymbol{A_1^{m,n}, Seq}$)** yields $A_1^{m,n}$, filled up, column-wise, with a character in each cell from the string Seq in order. If the string is of length < mn, the string is repeated. If the string is of length > mn, the string is truncated. i.e. If strlen(Seq) < mn, ∀ i,j $a_{i,j}$ = Seq[k], where k = mod(j*m + i; strlen(Seq)) else ∀ i,j $a_{i,j}$ = Seq[k], where k = j*m + i.

Example:

If, Seq = "ATCGTCGACGATATCG" then StoColSMat $(A_1^{4,3}, Seq)$ yields $A_1^{4,3} = \begin{vmatrix} A & T & C \\ G & T & C \\ G & A & C \\ G & A & T \end{vmatrix}$.

If, Seq = "ATCGTCGA" then StoColSMat $(A_1^{4,3}, Seq)$ yields $A_1^{4,3} = \begin{vmatrix} A & T & C \\ G & T & C \\ G & A & A \\ T & C & G \end{vmatrix}$.

4.2. Algorithm for DNA Encodings

Using the above defined string matrix data structure and its operations, the algorithm for generating the DNA encoding table is described below.

1. Select two different DNA sequence patterns Seq_1, Seq_2, reflecting the biological characteristics of n nucleotide bases.
   a. Store seq_1 in $A_1^{1,4}$ row-wise. StoRowSMat($A_1^{1,4}$, seq_1).
   b. Create a 16x4 StringMatrix by repeating the rows 16 times. $B_1^{16,4}$ = joinrows $(16*A_1^{1,4})$
2. For each one of the alphabets in Seq_2, create StringMatrix of order 4x4 each cell holding the same alphabet. Name them $C1_1^{4,4}$, $C2_1^{4,4}$, $C3_1^{4,4}$, $C4_1^{4,4}$ in order.
   Join them vertically in the order of the alphabets in Seq_2.
   $C1_1^{4,4}$ = FillMat($C1_1^{4,4}$,X_1); $C2_1^{4,4}$ = FillMat($C2_1^{4,4}$,X_2);
   $C3_1^{4,4}$ = FillMat($C3_1^{4,4}$,X_3); $C4_1^{4,4}$ = FillMat($C4_1^{4,4}$,X_4), where Seq_2 = (X_1,X_2,X_3,X_4).
   $C_1^{16,4}$ = Joinrows ($C1_1^{4,4}$, $C2_1^{4,4}$, $C3_1^{4,4}$, $C4_1^{4,4}$).
3. D = ADD(B,C). $D_2^{16,4} = B_1^{16,4} + C_1^{16,4}$.
4. Store seq_2 in $E_1^{1,4}$ column-wise. Join 4 identical $E_1^{4,1}$ horizontally or to obtain $E_1^{4,4}$. Join 4 identical $E_1^{4,4}$ vertically to obtain $F_1^{16,4}$
   a. StoColSMat($E_1^{1,4}$, seq_2); JOINCOLUMNS($E_1^{4,4}$,4*$E_1^{1,4}$).
   b. Join 4 identical $E_1^{4,4}$ vertically to obtain $F_1^{16,4}$ = JOINROWS $(4 * E_1^{4,4})$.
5. $G_3^{16,4} = D_2^{16,4} + F_1^{16,4}$.
6. Choose an option (u,v), where $1 \le u < v \le 4$. (To choose two different rows from $C1_1^{4,4}$). Extract from $G_3^{16,4}$ the eight rows numbered u, v, u+4, v+4, u+8, v+8, u+12, v+12 and Join them vertically in order.
   $G1_3^{1,4}$ =SUBMAT$\left(G_3^{16,4}, \mathrm{u}, 1, \mathrm{u}, 4\right)$; $G2_2^{1,4}$ = SUBMAT$\left(G_3^{16,4}, \mathrm{v}, 1, \mathrm{v}, 4\right)$.
   $G3_3^{1,4}$ =SUBMAT$\left(G_3^{16,4}, \mathrm{u}+4,1, \mathrm{u}+4,4\right)$; $G4_2^{1,4}$ = SUBMAT$\left(G_3^{16,4}, \mathrm{v}+4,1, \mathrm{v}+4,4\right)$.
   $G5_3^{1,4}$ =SUBMAT$\left(G_3^{16,4}, \mathrm{u}+8,1, \mathrm{u}+8,4\right)$; $G6_2^{1,4}$ = SUBMAT$\left(G_3^{16,4}, \mathrm{v}+8,1, \mathrm{v}+8,4\right)$.
   $G7_3^{1,4}$ =SUBMAT$\left(G_3^{16,4}, \mathrm{u}+12,1, \mathrm{u}+12,4\right)$; $G8_2^{1,4}$ = SUBMAT$\left(G_3^{16,4}, \mathrm{v}+12,1, \mathrm{v}+12,4\right)$.
   $H_3^{8,4}$ = JOINROWS ($G1_3^{1,4}$, $G2_3^{1,4}$, $G3_3^{1,4}$, $G4_3^{1,4}$, $G5_3^{1,4}$, $G6_3^{1,4}$, $G7_3^{1,4}$, $G8_3^{1,4}$)
7. Generate an Intron Sequence of length 20. Let C_i be the ith character of the Intron Sequence. Define a string Seq_3 of length 64 such that if Y_i be the ith character of Seq_3, then $Y_i = C_{i \bmod 20}$. Store Seq_3 in $J_1^{16,4}$ row-wise or column-wise. (24x23x? x2 possibilities). Let C_i be the ith character of the Intron Sequence. Define a string Seq_3 of length 64 such that if Y_i be the ith character of Seq_3, then $Y_i = C_{i \bmod 20}$.
   a. Store Seq_3 in $J_1^{16,4}$ row-wise. StoRowSMat($J_1^{16,4}$, Seq_3)
   b. Store Seq_3 in $J_1^{16,4}$ column-wise. StoColSMat($J_1^{16,4}$, Seq_3)

8. Extract from $J_1^{16,4}$ the eight rows numbered u, v, u+4, v+4, u+8, v+8, u+12, v+12 and Join them vertically in order to obtain $M_1^{8,4}$. Obtain $N_1^{8,4}$, the complement of $M_1^{8,4}$.
   $J1_1^{1,4}$ =SUBMAT$\left(J_1^{16,4}, \mathrm{u}, 1, \mathrm{u}, 4\right)$; $j2_1^{1,4}$ = SUBMAT$\left(J_1^{16,4}, \mathrm{v}, 1, \mathrm{v}, 4\right)$.
   $J3_1^{1,4}$ =SUBMAT$\left(J_1^{16,4}, \mathrm{u}+4,1, \mathrm{u}+4,4\right)$; $J4_1^{1,4}$ = SUBMAT$\left(J_1^{16,4}, \mathrm{v}+4,1, \mathrm{v}+4,4\right)$.
   $J5_1^{1,4}$ =SUBMAT$\left(J_1^{16,4}, \mathrm{u}+8,1, \mathrm{u}+8,4\right)$; $J5_1^{1,4}$ = SUBMAT$\left(J_1^{16,4}, \mathrm{v}+8,1, \mathrm{v}+8,4\right)$.
   $J3_1^{1,4}$ =SUBMAT$\left(J_1^{16,4}, \mathrm{u}+16,1, \mathrm{u}+16,4\right)$; $J4_1^{1,4}$ = SUBMAT$\left(J_1^{16,4}, \mathrm{v}+16,1, \mathrm{v}+16,4\right)$
   $M_1^{8,4}$ = JOINROWS($J1_1^{1,4}$,$J2_1^{1,4}$,$J3_1^{1,4}$,$J4_1^{1,4}$,$J5_1^{1,4}$,$J6_1^{1,4}$,$J7_1^{1,4}$,$J8_1^{1,4}$).

$N_1^{8,4}$ = COMPLEMENT($M_1^{8,4}$).

9. K = ADD (G, J); K1 = ADD (H, N); L = Join (K, K1) vertically. State is S(8) = {$L_4^{24,4}$}.
   a. $K_4^{16,4} = G_3^{16,4} + J_1^{16,4}$;
   b. $K1_4^{8,4} = H_3^{8,4} + N_1^{8,4}$;
   c. $L_4^{24,4}$ = Joinrows($K_4^{16,4}$, $K1_4^{8,4}$)

   Thus the encoding table that has been formed will have 96 unique DNA code sequences.
10. In a similar manner we can create a StringMatrix $L_4^{4,24}$ that will have 96 unique DNA code sequences.
11. $L_4^{24,4}$ / $L_4^{4,24}$ is the codomain of Encoding Mapping.
12. Select any choice for the alphanumeric characters from the Collating Sequence Table (Opt_6). Concatenate the sequences together in order. The formed sequence may be termed Coll_seq.

Table 4: Collating Sequence Table (48 Choices)

Choice	CodeSeq	Choice	CodeSeq	Choice	CodeSeq	Choice	CodeSeq
0	aAN	12	zNA	24	ZaN	36	NAa
1	aAn	13	zNZ	25	Zan	37	NAz
2	aZN	14	znA	26	ZzN	38	NZa
3	aZn	15	znZ	27	Zzn	39	NZz
4	aNA	16	AaN	28	ZNa	40	naN
5	aNZ	17	Aan	29	ZNz	41	nan
6	anA	18	AzN	30	Zna	42	nzN
7	anZ	19	Azn	31	Znz	43	nzn
8	zAN	20	ANa	32	NaN	44	nAa
9	zAn	21	ANz	33	Nan	45	nAz
10	zZN	22	Ana	34	NzN	46	nZa
11	zZn	23	Anz	35	Nzn	47	nZz

a denotes the sequence a, b, … z	26 in number
z denotes the sequence a in reverse order	
A denotes the sequence A, B, … Z	26 in number
Z denotes the sequence A in reverse order	
N denotes the sequence 0, 1,...9, <, >,...	10 + 34 in number
n denotes the sequence N in reverse order.	

If we need to provide for more characters, we can increase the number of rows/columns in the domain and range of Encoding Mapping at Stage 8.

For example the choice of 22 (Ana) in the collating sequence table, the order of the alphanumeric characters is the sequence of Upper case Alphabets followed by the Numbers and the chosen special characters followed by Lower case Alphabets.

For example, the choice 3 (aZn) will lead to Coll_seq of length

10.1.1.For Row-wise (Opt_7=”R”) storing up of Coll_seq in a StringMatrix of $L1_4^{24,4}$ of order 24x4: Form a StingMatrix $L1_4^{24,4}$, such that the ith row jth column element equals the mth character in the Coll_seq where m = i*4 + j. $L1_4^{24,4}$ is the domain of Encoding Mapping.

10.1.2,For Column-wise (Opt_7=”C”) storing up of Coll_seq in a StringMatrix $L1_4^{24,4}$ of order 24x4: Form a StingMatrix $L1_4^{24,4}$, such that the ith row jth column element equals the mth character in the Coll_seq where m = j*24 + i. $L1_4^{24,4}$ is the domain of Encoding Mapping.

10.2.1.For Row-wise (Opt_7=”R”) storing up of Coll_seq in a StringMatrix of $L1_4^{4,24}$ of order 4x24: Form a StingMatrix $L1_4^{4,24}$, such that the ith row jth column element equals the mth character in the Coll_seq where m = i*24 + j. $L1_4^{4,24}$ is the domain of Encoding Mapping.

10.2,2.For Column-wise (Opt_7=”C”) storing up of Coll_seq in a StringMatrix $L1_4^{4,24}$ of order 4x24: Form a StingMatrix $L1_4^{4,24}$, such that the ith row jth column element equals the mth character in the Coll_seq where m = j*4 + i. . $L1_4^{4,24}$ is the domain of Encoding Mapping.

13. $L1_4^{24,4}$ / $L1_4^{4,24}$ is the domain of Encoding Mapping.

Encoding

Map $L1_1^{24,4} \rightarrow L_4^{24,4}$, $\forall$ i,j $l1_{i,j} \rightarrow l_{i,j}$, so that any alphabet or number of special character will be encoded as a sequence of 4 DNA alphabets. The DNA encoding table that would be generated using the above string matrix data structure definition and operations is given in table 5 & 6 below.

Example of $L1_1^{24,4}$

Table 5: Domain for the DNA encoding Mapping.

a	Y	W	–
b	Z	X	–
c	A	Y	}
d	B	Z	]
e	C	0	|
f	D	1	\
g	E	2	+
h	F	3	=
i	G	4	_
j	H	5	-
k	I	6	)
l	J	7	(
m	K	8	*
n	L	9	&
o	M	<	^
p	N	>	%
q	O	,	$
r	P	.	#
S	Q	?	@
T	R	/	!
U	S	:	~
V	T	;	`
W	U	“	€
X	V	‘	£

Example of $L_4^{24,4}$

Table 6: Co Domain for DNA Encoding Mapping.

ACAT	AAAA	ATAA	AGAG
ACTG	AATT	ATTT	AGTA
ACCC	AACC	ATCG	AGCG
ACGA	AAGG	ATGC	AGGG
TCAT	TAAT	TTAA	TGAA
TCTG	TATG	TTTT	TGTT
TCCG	TACC	TTCC	TGCG
TCGT	TAGA	TTGG	TGGC
CCAG	CAAT	CTAT	CGAA
CCTA	CATG	CTTG	CGTT
CCCG	CACG	CTCC	CGCC
CCGG	CAGT	CTGA	CGGG
GCAA	GAAG	GTAT	GGAT
GCTT	GATA	GTTG	GGTG
GCCG	GACG	GTCG	GGCC
GCGC	GAGG	GTGT	GGGA
ACTC	AATA	ATTA	AGTT
ACCG	AACG	ATCC	AGCC
TCTC	TATC	TTTA	TGTA
TCCC	TACG	TTCG	TGCC
CCTT	CATC	CTTC	CGTA
CCCC	CACC	CTCG	CGCG
GCTA	GATT	GTTC	GGTC
GCCC	GACC	GTCC	GGCG

4.3. Novel features of the DNA Encoding Algorithm

The proposed algorithm has some novel features as follows,

- Is based on a novel string matrix data structure
- To be the first complete DNA Encoding of Plaintext Table in the field of DNA Cryptography.
- The DNA Encoding of Plaintext using the proposed encoding scheme itself helps to achieve first level of encryption.
- The encoding table is generated using random DNA sequences. The procedure for generating the table also has randomized steps. Hence the table is difficult to mimic.
- The table is dynamically changed for every session between the sender and receiver. So, even if the table is cryptanalyzed, it cannot be used for the next session to tap the communication between the sender and receiver.
- The Table has unique values for all characters in the character set and does not have duplicate DNA sequences across multiple table generation iterations.

5. Experimental Analysis

5.1. Time Required For Encoding Table Generation

The Figure (a) shows the total time utilized for DNA Encoding table generation. The table generation has been tested across 1000 sample encoding table generations. The chart below shows the time required for generating the tables taken in buckets / groups of 100 tables each. The chart evidently shows that the time to generate the encoding table is in the order of nanoseconds only and the minimum time required is 1.95 ns and an average of 3 ns and a maximum of less than 4.5 ns is required in each of the 1000 table groups.

5.2. Test for Duplicate DNA Sequence Generation across Multiple Encoding Table Generations

Using this Duplicates Testing, we try to show that the encoding table that is generated always has a collection of unique DNA sequences for assigning to the character set. The test has been taken for randomly generated 1000 tables. The probability result of duplicate is 0 as shown in Figure (b).

5.3. Experimental Results

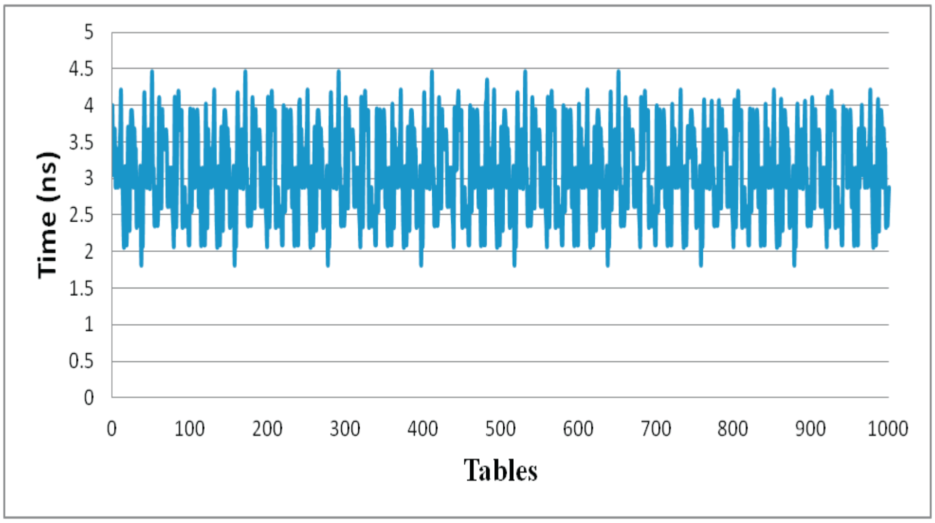

Figure (a) Time Utilization for Encoding Table Generation

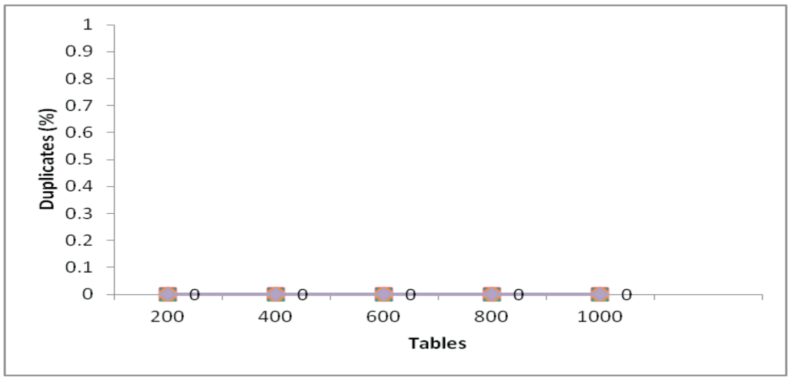

Figure (b) Test Result for Duplicate DNA Sequence Generation

6. Fulfilment of security features

We consider the following features and compare the fulfilment of features against the existing technique and the novel technique described above. This shows how the DNA encoding algorithm described above also possesses the required qualitative security features as shown in Table7.

Table 7: Fulfilment of Functional characteristics of the proposed encoding algorithm

Sl.No	Features	Fulfillment
1	Robustness	Is based on a formally defined string matrix data structure and its operations
2	Confidentiality	Since the encoding technique is random and dynamic, the confidentiality of the encoded plain text can be never compromised.
3	Randomness	Every step of the encoding algorithm is constituted of random steps (for e.g., the DNA alphabets used for generating the first two table elements, the choice of the DNA alphabet to be included as the third element, the choice of the inclusion of the third element either row-wise or column-wise, etc.)
4	Dynamicity	The DNA encoding table is newly generated for every session between the sender and the receiver. Thus, the encoding is dynamic and helps to circumvent the Man in the Middle attack.
5	Complete Character Set Fulfillment	The encoding table provides for encoding sequences for the complete character set.
6	Uniqueness	The experiment analysis shows that the DNA encoding sequences generated are unique across multiple iterations.

Table 8: Time Taken for Encoding

Words (in count)	Encoding Time (ns)
500	66
1000	209
2000	868
4000	3872
6000	10531
8000	23422
10000	38940

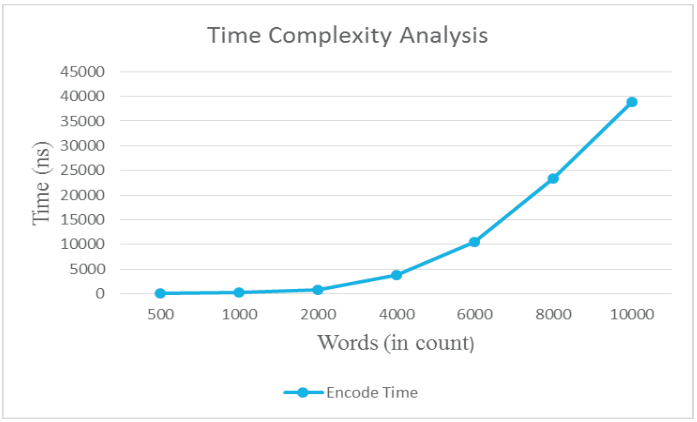

Figure (c) Time Complexity Chart

7. Conclusion

DNA encoding of plain text is the first step towards DNA cryptography. At present, this step has not received enough focus and very simple encoding techniques which fulfil only a subset of the required functional and non-functional attributes of encoding algorithms are available. Also, most of them do not support for the encoding of the complete character set. In addition, the encoding technique does not have a systematic methodology to be followed. This paper proposed a novel and unique technique for DNA encoding of plain text which fulfills all of the functional and non-functional attributes that should be characteristic of an encoding algorithm so that it could constitute for a systematic technique for DNA encoding of plain text.

References

1. Akanksha Agrawal, Akansha Bhopale, Jaya Sharma, Meer Shizan Ali, and Divya Gautam. *Implementation of DNA algorithm for secure voice communication*. International Journal of Scientific & Engineering Research; 2012.
2. G.Cui; L. Cuiling, L. Haobin, and L. Xiaoguang. *DNA Computing and its application to information security field.* IEEE Fifth International Conference on Natural Computation- Tianjian, China; 2009.
3. Guangzhao Cui; Limin Qin; Yanfeng Wang and Xuncai Zhang. *An encryption scheme using DNA technology.* Third International Conference on Bio-Inspired Computing: Theories and Applications; 2008. p. 37- 42.
4. Kang Ning. *A Pseudo DNA Cryptography Method.* http://arxiv.org/abs/0903.2693; 2009.
5. Li Xin-she; Zhang Lei and Hu Yu-pu. *A Novel Generation Key Scheme Based on DNA.* International Conference on Computational Intelligence and Security; 2008. p. 264-266.
6. Mona Sabry, Mohamed Hashem, and Taymoor Nazmy. *Three Reversible Data Encoding Algorithms based on DNA and Amino Acids Structure.* International Journal of Computer Applications; 2012.
7. NRDC, Govt. of India, http://www.nrdcindia.com/Patent%20Asistance%20 (in%20India) %20Form%202011.pdf
8. O.Tornea and M.E.Borda. *DNA Cryptographic Algorithms.* International Conference on Advancements of Medicine and Health Care through Technology IFMBE Proceedings, Springer; 2009.p. 223-226.
9. Padma Bt. *DNA computing theory with ECC*. http://www.scribd.com/doc/55154238/Report; 2010.
10. Qiang Zhang; Ling Guo; Xianglian Xue; Xiaopeng Wei. *An image encryption algorithm based on DNA sequence addition operation.* Fourth International Conference on Bio-Inspired Computing; 2009, p.1-5, 16-19.
11. Sadeg, S.; Gougache, M.; Mansouri, N. and Drias, H. *An encryption algorithm inspired from DNA.* International Conference on Machine and Web Intelligence; 2010. p.344 -349, 3-5.
12. Sherif T. Amin, MagdySaeb, Salah El-Gindi. *A DNA-based Implementation of YAEA Encryption Algorithm.* IASTED International Conference on Computational Intelligence; 2006. p.120-125.
13. X.Guozhen, L.Mingxin, Q.Lei, and L.Xuejia. *New field of cryptography: DNA Cryptography.* Chinese Science Bulletin, Springer Verlag, Germany; 2006. p.1413-1420.
14. Xing Wang , Qiang Zhang. *DNA computing based Cryptography*. IEEE proceeding of Fourth International Conference on Bio-Inspired Computing; 2009. p.1-3.
15. Xiutang Geng, Linqiang Pan, and Jin xu. *A DNA Sticker algorithm for bit substitution in a block cipher.* Journal of Parallel and Distributed Computing; 2008.
16. Zhang, Qiang, Wang, Qian, Wei, Xiaopeng. *A Novel Image Encryption Scheme based on DNA Coding and Multi-Chaotic Maps.* Advanced Science Letters; 2010. p. 447-451.