Three Reversible Data Encoding Algorithms based on DNA and Amino Acids' Structure

Mona Sabry
Computer Science dept., Faculty of Computer Science and Information Systems, Ain Shams University, Cairo, Egypt.

Mohamed Hashem
Information Systems dept., Faculty of Computer Science and Information Systems, Ain Shams University,Cairo, Egypt.

Taymoor Nazmy
Computer Science dept., Faculty of Computer Science and Information Systems, Ain Shams University, Cairo, Egypt.

ABSTRACT

The field of using biology in cryptography is a new and very promising direction in cryptographic research. Although in its primitive stage, DNA cryptography is shown to be very effective. Currently, several DNA computing algorithms are proposed for quite some cryptography, cryptanalysis and steganography problems, and they are very powerful in these areas.

In this paper, we introduce three methods of encoding inspired from the DNA (or RNA) structure and its relation to the amino acids in the standard genetic code table. The paper explains three techniques to convert data from binary form to DNA (or RNA) form then to amino acids' form and the reverse. We proved they are applicable and correctly reversible.

The algorithms can serve in DNA computers and biological experiments by representing data in the form of amino acids. They also can be viewed as a simple algorithm to convert data from a form to another completely different form with the ability to convert it back to the initial form. Although they don't include the use of secret key but they can also be used as an auxiliary factor in data integrity and digital signature applications.

Keywords

DNA, RNA, Amino acids, codons, encryption, digital signature, algorithm, encoding, transformation, cryptography, authentication, confidentiality.

1. INTRODUCTION

DNA cryptography is a new born cryptographic field emerged with the research of DNA computing, in which DNA is used as information carrier and the modern biological technology is used as implementation tool. The vast parallelism [1] and extraordinary information density inherent in DNA molecules are explored for cryptographic purposes such as encryption, authentication, signature, and so on [2]. The new born DNA cryptography [3], [4] is far from mature both in theory and realization. However, researchers in DNA cryptography are still looking at much more theory than practicality. The constraints of its high tech lab requirements and computational limitations, combined with the labor intensive extrapolation means, all prevent DNA computing from being of efficient use in today's security world.

There is not a general theory about applying DNA molecules in cryptography [5], [6]. Some key technologies in DNA research, such as Polymerase Chain Reaction (PCR), DNA synthesis, and DNA digital coding, have only been developed and well accepted in recent years [7]. In 1999, Clelland et al. [7] achieved an approach to steganography by hiding secret messages encoded as DNA strands among a multitude of random DNA. In 2000, Prof. Gehani [6] presented one-time-pads mechanism based on DNA to design two encryption methods of one-time-pads of DNA sequence. One method is to translate the fixed length DNA plain code sequence cell to DNA cryptograph sequence according to the defined mapping graph (substitution). The other is called Exclusive-OR method, which uses biological molecular techniques to carry through Exclusive-OR operation of DNA plain code and cipher key sequence.

Another approach is lead by Ning Kang [8] in which he did not use real DNA computing, but just used the principle ideas in central dogma of molecular biology to develop his cryptography method. The method only simulates the transcription, splicing, and translation process of the central dogma; thus, it is a pseudo DNA cryptography method. There is another investigation conducted by [9] which is based on a conventional symmetric encryption algorithm called ―Yet Another Encryption Algorithm‖ (YAEA) developed by Saeb and Baith. In this study, he introduces the concept of using DNA computing in the field of cryptography in order to enhance the security of cryptographic algorithms. He combined the mathematical model of the algorithm with the DNA strands as the secret key to perform a more secured cryptographic algorithm. DNA cryptography does not absolutely repulse traditional cryptography and it is possible to construct hybrid cryptography of them. In 2009, O. Tornea and M.E. Borda [10] and Xing Wang, Qiang Zhang [11] proposed this idea by combining DNA computing with RSA.

In our work, we applied the conversion of character or binary form of data to the DNA form and then to amino acid form in three different methods. The importance of such transformation lies mainly in representing data in a biological form that can make data be able to go through biological experiments and processes, especially related to Amino Acids and DNA. It is also a way of viewing data moving through biological processes and representing it in a binary form which can be used in many computer applications. In the field of cryptography, the encoding techniques cannot provide security by their own as they don't include the use of a secret key. But they can be embedded into another encryption algorithm to enhance confusion and therefore

enhance security. This concept is suitable for applying data integrity, digital signature and confidentiality.

The next sections are organized as follows: section 2 explains biological background information that helps us understand the biological concepts involved in our algorithms. Section 3 contains the design of table then the details of the encoding algorithms, their inverse and complexity calculation. Section 4 involves discussions and analysis about the algorithms and their applications.

2. BIOLOGICAL BACKGROUND

DNA is a nucleic acid that contains the genetic instructions used in the development and functioning of all known living organisms and some viruses. The DNA double helix is stabilized by hydrogen bonds between the bases attached to the two strands. The four bases found in DNA are adenine (abbreviated A), cytosine (C), guanine (G) and thymine (T).

A gene is a sequence of DNA that contains genetic information. The genetic code consists of three-letter 'words' called codons formed from a sequence of three nucleotides (e.g. ACT, CAG, TTT). Since there are 4 bases in 3-letter combinations, there are 64 possible codons (4^3 combinations). These encode the twenty standard amino acids, giving most amino acids more than one possible codon. There are also three 'stop' or 'nonsense' codons signifying the end of the coding region; these are the TAA, TGA and TAG codons.

3. THE ENCODING TO AMINO ACIDS

Any form of data can be represented in a binary form (message, image or signal). This form can be transferred to DNA or RNA form according to Table 1. Note that the only difference between DNA and RNA is that letter 'T' in DNA is the same as letter 'U' in RNA. The RNA form is transferred to the Amino acids form according to Table 2 which is a standard universal table of Amino acids and their codons representation in the form of RNA.

Note that each amino acid has a name, abbreviation (3-letter form), and a single character symbol (1-letter form). This character symbol is what we will use in our algorithm.

Table 1. DNA and RNA Representation of bits.

Bit 1	Bit 2	RNA	DNA
0	0	A	A
0	1	C	C
1	0	G	G
1	1	U	T

Note that the Table 2 and Table 3 are referenced from wikipedia: http://en.wikipedia.org/wiki/Genetic_code

Table 2. The RNA codon table

nonpolar	polar	Basic	acidic	(stop codon)

Standard genetic code

1st base	2nd base U		2nd base C		2nd base A		2nd base G		3rd base
U	UUU	(Phe/F) Phenylalanine	UCU	(Ser/S) Serine	UAU	(Tyr/Y) Tyrosine	UGU	(Cys/C) Cysteine	U
	UUC		UCC		UAC		UGC		C
	UUA	(Leu/L) Leucine	UCA		UAA	Stop (Ochre)	UGA	Stop (*Opal*)	A
	UUG		UCG		UAG	Stop (*Amber*)	UGG	(Trp/W) Tryptophan	G
C	CUU		CCU	(Pro/P) Proline	CAU	(His/H) Histidine	CGU	(Arg/R) Arginine	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	(Gln/Q) Glutamine	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	(Ile/I) Isoleucine	ACU	(Thr/T) Threonine	AAU	(Asn/N) Asparagine	AGU	(Ser/S) Serine	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	A
	AUG	(Met/M) Methionine	ACG		AAG		AGG		G
G	GUU	(Val/V) Valine	GCU	(Ala/A) Alanine	GAU	(Asp/D) Aspartic acid	GGU	(Gly/G) Glycine	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	(Glu/E) Glutamic acid	GGA		A
	GUG		GCG		GAG		GGG		G

Table 3. The inverse RNA codon table

Amino Acid	Codons	Amino Acid	Codons
Ala/A	GCU, GCC, GCA, GCG	**Leu/L**	UUA, UUG, CUU, CUC, CUA, CUG
Arg/R	CGU, CGC, CGA, CGG, AGA, AGG	**Lys/K**	AAA, AAG
Asn/N	AAU, AAC	**Met/M**	AUG
Asp/D	GAU, GAC	**Phe/F**	UUU, UUC
Cys/C	UGU, UGC	**Pro/P**	CCU, CCC, CCA, CCG
Gln/Q	CAA, CAG	**Ser/S**	UCU, UCC, UCA, UCG, AGU, AGC
Glu/E	GAA, GAG	**Thr/T**	ACU, ACC, ACA, ACG
Gly/G	GGU, GGC, GGA, GGG	**Trp/W**	UGG
His/H	CAU, CAC	**Tyr/Y**	UAU, UAC
Ile/I	AUU, AUC, AUA	**Val/V**	GUU, GUC, GUA, GUG
START	AUG	**STOP**	UAA, UGA, UAG

3.1. Constructing the English alphabet table:

In the standard genetic code table (Table 2 or its inverse (Table 3)), we have only 20 amino acids in addition to 1 start and 1 stop codons. Each amino acid is abbreviated with one unique character (English letter). In order to construct a complete set of alphabetical English letters, we need 26 letters with their transformation-encoding- to DNA.

The letters we need to fill are (B, J, O, U, X, Z). So we will make these characters share some amino acids their codons. The three stop codons have 2 of one family type (UAA,UAG) to be assigned to letter B and one of other type (UGA) to be assigned to letter J. We have 3 amino acids (L, R, S) having 6 codons. By noticing the sequence of RNA of each, we can figure out that each has 4 codons of the same type and 2 of another type.

Those 2 of the other type are shifted to the letters (O, U, X) respectively. Letter (Z) will take one codon from (Y), so that Y: UAU, Z: UAC. Now the new distribution of codons is illustrated in Table 4.

The table illustrates letters from A-Z with the associated amino acids as explained before. It explains the modification process by colors; the orange codons are the newly modified ones to be distributed among acids under the non-yellow letters.

Counting the number of codons of each letter, we will find the number varies between 1 and 4 codons per letter. We will call this number 'Ambiguity' of the character. The table shows also the ambiguity for each letter after modifications. Table 4 also shows each family from which each group of codons derived (last row).

Table 4. The final distribution of Amino upon English letters.

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ambiguity	4	2	2	2	2	2	4	2	3	1	2	4	1	2	2	4	2	4	4	4	2	4	1	2	1	1
A	GCU	UAA	UGU	GAU	GAA	UUU	GGU	CAU	AUU	UGA	AAA	CUU	AUG	AAU	UUA	CCU	CAA	CGU	UCU	ACU	AGA	GUU	UGG	AGU	UAU	UAC
C	GCC	UAG	UGC	GAC	GAG	UUC	GGC	CAC	AUC		AAG	CUC		AAC	UUG	CCC	CAG	CGC	UCC	ACC	AGG	GUC		AGC		
G	GCA						GGA		AUA			CUA				CCA		CGA	UCA	ACA		GUA				
U	GCG						GGG					CUG				CCG		CGG	UCG	ACG		GUG				
Family	GC	UA	UG	GA	GA	UU	GG	CA	AU	UG	AA	CU	AU	AA	UU	CC	CA	CG	UC	AC	AG	GU	UG	AG	UA	UA

3.2.Convert from DNA or RNA to Amino Acids

We have three different ways to use this table to convert from **RNA form** to **Amino acids form**:

- Discrete Encoding.
- Overlapping Encoding.
- Embedded DNA.

1- Discrete Encoding:

Each letter in Table 4 is derived from a family. The family can have more than one letter as illustrated in Table 5. Having the message to be encoded in the binary form, we can use table 1 to convert it to RNA form. We can use table 5 to convert the RNA form to amino acids form by taking each two RNA characters as input to the table then choosing randomly one of the corresponding letters which represent the amino acids.

The **inverse encoding** is the same using table 5 but in reverse order.

Table 5. The RNA families and their letters (inverse of table 4).

Hexa-decimal value	Amino Acid Family	Letter 1	Letter 2	Letter 3
0	AA	K	N	
1	AC	T		
2	AG	U	X	
3	AU	I	M	
4	CA	H	Q	
5	CC	P		
6	CG	R		
7	CU	L		
8	GA	D	E	
9	GC	A		
a	GG	G		
b	GU	V		
c	UA	B	Y	Z
d	UC	S		
e	UG	C	J	W
f	UU	F	O	

Algorithm complexity:

Complexity of discrete encoding has two main parts:

- Search table 5 which in worst case and on average will cost constant N operations.
- Processing the input of size (N) which is the RNA message:
  This loop is LOG (N). LOG is to the base 2 as we take two RNA to substitute in the table.

  So total algorithm complexity is: $O(\log_2 N)$.

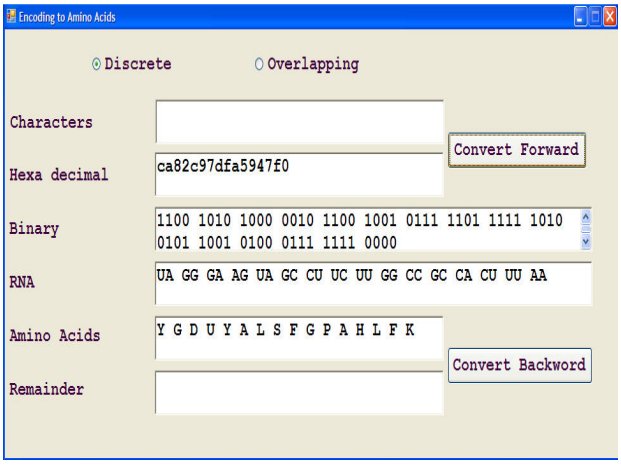

Fig 1: Example of Discrete Encoding.

In the previous example, each two binary digits are used as input to table 1 to convert them to one RNA character. Then each two RNA characters representing one family are used in table 5 to select any of the amino acids in this family.

2- Overlapping Encoding:

In this method, we will use Table 4 with the Ambiguity cells, and ignore the family cells. Having the input in the form of RNA, Each 3 RNA are used as input to substitute in table 4. The output is one amino acid character (column number) and one ambiguity number (row number).the ambiguity is a number that varies from 0 to 3. The ambiguity number is to be converted to RNA form through Table 1 and then added to the end of the input to take its turn in processing.

Steps are summarized through the following algorithm:

```
1- Input is the RNA form of the
   message (Input).
2- Output is the Amino acids form
   (Output) which is initially an
   empty string.
3- Count number of characters in
   Input (Count).
4- If Count <3: go to step 10
5- Else: Take 3 RNA characters from
   Input and find their position in
   table 4. The position is defined
   by an amino acid letter (A) and
   ambiguity number (N) ranging from
   0-3.
6- Add A to Output.
7- Using table 1, the ambiguity (N)
   can be encoded in the form of RNA
   (R).
8- Append R to the end of Input.
9- Go to step 3.
10-Make the final Remainder = Input.
```

	input	F	C	I	S			
	HEXA	46	43	49	53			
	BINARY	0X01000110	0X01000011	0X01001001	0X01010011			
Input	RNA	CACG	CAAU	CAGC	CCAU			
Triples	3 RNA codons	CAC	GCA	AUC	AGC	CCA	U	
A	AMINO	H	A	I	X	P		
R	AMBIG	C	G	C	C	G		
	collect ambig	UCG	CCG					
A	Amino	S	P					
R	REMAINDER	U	U					
Output	FINAL	H	A	I	X	P	S	P
Remainder	REMAINDER	UU						

Fig 2: Detailed Example of overlapping encoding (paper and pencil).

In the previous example, each two binary digits are used in table 1 to get the corresponding RNA character. Each triple of RNA are used in Table 4 to get the corresponding row and column. The column represents the amino acid and row represents the ambiguity. Collecting all the message ambiguity and reprocess them again as we explained, until the number of ambiguity characters is less than 3. Here we call the left ambiguity "the remainder".

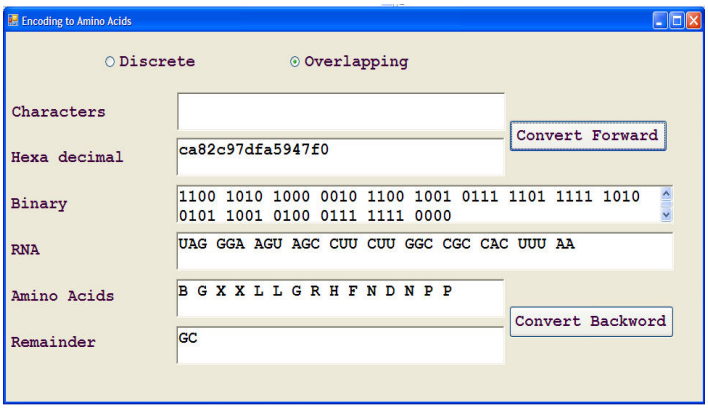

Fig 3: Example of overlapping encoding (program implementation).

Algorithm complexity:

The algorithm contains four main parts:

- Load of the amino acids genetic table:

  It is (64x2) assignment operations to record the amino acid character and ambiguity of 64 combinations of the possible RNA triples.

- Search the amino acid genetic table (substitution): in the worst case and average case, we will have constant number of operations to reach the result.

- Convert from integer value of ambiguity to RNA: it is also considered a search in a static table. The worst and average case will cost constant number of operations.

- Encoding of DNA message to Amino acids:
  Suppose (N) is the size of input RNA string. The loop (log (N+log N)) represents processing of N inputs and successive additional results of ambiguity. LOG is to the base 3 as we process triples of RNA per iteration. Inside the loop we have: (3) constant operations + search in genetic table + conversion from integer to RNA. In the worst case we have (71) operations and average (37) operations per iteration.
  So we have total algorithm complexity **O ($\log_3$ (N + $\log_3$ N)).**

The **inverse** of overlapping encoding is implemented as follows:

```
1- Inputs: the  remainder (R) and the
   amino acids form (Input)
2- (Output) is the RNA form
3- Define len_R = length of R.
4- Define len_Inp = length of Input
5- While (len_Inp>0), do the following
1. If (len_R< len_Inp), Then
i- (Amino)  =Input.substring(  len_Inp-
   len_R)
ii-      Input =Input- Amino
iii-     (Ambig) =R
2. Else
i- (Ambig) =R.substring(len_R-len_Inp)
ii-      R= R-Ambig
iii-     Amino=Input
iv-      Input =""
3. Define  (E): Substitute in table 4
   using each character in (Amino) and
   its corresponding in (Ambig) to get
   their decode to RNA.
4. Let R=E and go to step5.
6- Output =R.
```

In the previous algorithm fig.2, we will start with the remainder and process the amino acid sequence from its end till the start.of the remainder is placed in reverse order in adverse to end of the amino acid sequence. One character from the remainder (row number) in addition to one amino acid (column number) are used to substitute in table 4 to get the corresponding codons. These codons are used as remainder with the rest of the amino acid sequence till they are finished. In the last step the remaining codons are themselves the RNA form of data.

Handling the Remainder:

Here we will introduce a way of embedding the remainder in the output message so as to simplify and unify the result. The remainder is defined to be a small portion of data that is used as a key to start decoding. Size of remainder is one or two DNA characters.

To unify this size we can add DNA character 'A' if the size equals to one. The couple of characters can be decoded using table 5 to get an amino acid character without a remainder. What is left is to give the information of the original size of the remainder to be used in decoding to extract the exact remainder and continue decoding. This is done by additional two DNA: "AC" represents the 'one' size and "AG" to represent the 'two' size. These two DNA can be also decoded to Amino Acid using table 5. So, the remainder is identified by two Amino acid characters at the end of the message.

In the inverse algorithm, we can firstly extract the last two Amino Acid characters to define the remainder. They should be decoded using table 5. The last amino acid represents the size. The other represents the remainder itself. If size = one, skip the last 'A' and use one character only of the remainder. Then we can continue decoding as previously explained given the remainder and the rest of the message to be decoded.

From the previous example, the final output is "HAIXPSP" and remainder is "UU". We will treat the remainder from Table 5 as 'F' for 'UU' and 'X' for 'AG' representing that the remainder is two characters. So the final output is "HAIXPSPFX".

3- Embedded DNA Encoding:

In this method, we will use Table 4 with the Ambiguity cells, and ignore the family cells like the overlapping Encoding but the difference is in the way of placing the Amino acid and ambiguity. Instead of collecting the ambiguity at the end of the message and re-encoding them as the original DNA, we embed each ambiguity in the form of DNA after each Amino acid. Then the whole result can be transformed to binary form to have the result in the same form.

The algorithm is as follows:

```
1- Input is the DNA or RNA form of the
   message (Input).
2- Output is the Amino acids form with
   embedded DNA representing ambiguity
   (Output).
3- If the Input count <3 and !=0
a. Then add 'A' character to Input
b. Go to step 3
4- If Input count ==0 then exit.
5- Else: Take 3 RNA characters from Input
   and find their position in table 4.
   The position is defined by (row) an
   amino acid letter (A) and (column)
   ambiguity number (N) ranging from 0-3.
6- Add A + N to Output.
7- If input count > 0, go to step 3.
```

As we see the output is in two forms Amino Acid and DNA. We can unify the form of output by putting all parts in binary form. The DNA part is encoded to binary using Table 1. the amino acid part is now acting in the form of characters so they are transformed to binary form of each character (ASCII code). Each 6 bits of input is converted to 3 DNA, they then are converted to one Amino Acid + one DNA. Each couple of output (Amino Acid + DNA) takes (8 bits +2 bits) =10 bits. This means that the ratio between input and output is 3/5.

The algorithm complexity is O(log N). The log is to base 3 as we process each time 3 RNA characters in constant steps of the loop.

	input	F	C	I	S		
	HEXA	46	43	49	53		
	BINARY	0X01000110	0X01000011	0X01001001	0X01010011		
Input	**RNA**	CACG	CAAU	CAGC	CCAU		
Triples	**3 RNA codons**	CAC	GCA	AUC	AGC	CCA	UAA
A	**AMINO +DNA**	H 'C'	A 'G'	I'C'	X 'C'	P 'G'	B 'A'
Output	**FINAL**	0X0100100001	0X0100000110	0X0100100101	0X0101100001	0X0101000010	0X0100001000
	DNA	**CAGAC**	**CAACG**	**CAGCC**	**CCGAC**	**CCAAG**	**CAAGA**

Fig 4: Example of the Embedded DNA encoding algorithm

In the previous example, each two binary digits are substituted in table 1 to get the corresponding RNA. Each triple of RNA is used in table 4 to get its corresponding row (ambiguity) and column (the amino acid). If the last RNA's don't complete 3 characters, we add additional 'A' characters till they reach 3 RNA's. Then each amino acid character is converted to binary (ASCII code). And each ambiguity in the form of RNA is converted to binary but through table 1. The result is 10 binary digits each. The whole result is converted to DNA or RNA form using table 1 so that it is represented in the form of DNA or RNA.

The inverse algorithm is to convert the RNA form of output to binary form. Then we take each 10 bits for processing. The first 8 bits are converted to hexadecimal then to a character. The last 2 bits are used to define ambiguity. Using each character with each ambiguity, we can substitute in table 4 and get the suitable codons (RNA). The codons are then converted to binary to get the original form of data.

4. ANALYSIS AND DISCUSSIONS

We have introduced three methods of data encoding to the form of Amino acids, or we can say our semi-artificially built distribution of amino acids. "Artificial" because of characters we added and switching's of codons we introduced to the genetic code table and "Semi" because we kept most of the standards in the genetic code table.

The introduced algorithms proved to meet the main characteristics of an algorithm. Definiteness: the algorithm is clearly specified and implemented through a computer program. Effectiveness: steps are sufficiently simple, basic and easily reversible. Input is defined to be any sequence of binary data. Output is defined to be a sequence of English characters representing the Amino acids in additional to remainder in the form of RNA or DNA. The outputs can be represented in a binary form. Finiteness: the algorithm terminates after a finite number of steps which is proved in the algorithm complexity.

We have proved that the encoding algorithms are reversible and applicable. The three algorithms can be implemented with one or many rounds. The idea of representing the amino acid form of data in English characters makes this form to be used as input to additional cycles. This is implemented by calculation of the hexadecimal of each letter. Then we can convert it to the binary then DNA forms which act as input to a new round.

The importance of such transformation lies mainly in representing data in a biological form that can make data be able to go through biological experiments and processes, especially related to Amino Acids and DNA. It is also a way of viewing data moving through biological processes and representing it in a binary form which can be used in many computer applications.

In the field of cryptography, the encoding techniques cannot provide security by their own as they don't include the use of a secret key [12]. But they can be embedded into another encryption algorithm to enhance confusion specially that the output can be again represented in a binary form which is completely different from the binary input. This was successfully implemented in a previous paper as a hybrid system with a cryptographic algorithm [13].

Although they don't depend on secret keys [12], they cannot be used as hash functions as they are reversible and size of output is directly proportional related to the size of input and not a fixed size.

It is obvious that the Discrete Encoding is much simpler to understand, implement and reverse than Overlapping encoding. But on the contrary to Overlapping and Embedded DNA, output of discrete encoding can vary from one implementation to other but gives the same decoding value. This depends mainly on the random function used in selection. This is a critical advantage against known plaintext attacks. But the difficulty of overlapping and embedded DNA encodings is also an advantage in the field of cryptography making the attacker's mission more difficult.

Discrete encoding can be viewed as one-to-one conversion; each time we have a certain letter in our message, it is encoded to the same output. This means that each letter is independent and not affected by the surrounding characters. But the random selection to codes of some characters enhances the concept of confusion.

On the contrary, in Overlapping and embedded DNA encoding, the sequence of characters in the input message clearly affects the output. This is because it is based on combining triples of DNA while one character is represented by 4 DNA which allows to interfering of code between successive characters.

This concept is suitable for applying data integrity, digital signature and confidentiality. As the change of a portion of the message will lead to completely another output. The remainder is a very critical member in the process of reverse overlapping encoding that it is considered the key to decode the message. The loss of the remainder for instance will make us completely unable to decode the message. The last portions of the output are related to the entire input message representing the set of ambiguity.

In Embedded DNA encoding, the output size is extended by 5/3 of the input size. This concept is increased with the use of multiple rounds which is suitable for data expansion like key expansion processes.

We also proved that the complexity of discrete encoding is **$O(\log_2 N)$** . Complexity of overlapping encoding is **$O(\log_3 (N + \log_3 N))$.** And complexity of Embedded DNA encoding is **$O(\text{Log}_3 N)$.**

5. CONCLUSION AND FUTURE WORK

We have introduced three methods of data encoding to the form of Amino acids. The encoding algorithms and their decoding proved to meet the main characteristics of an applicable reversible algorithm. They can be implemented with one or many rounds. This concept is suitable for applying data integrity, digital signature and confidentiality. As they don't include a secret key they cannot provide security by their own. But they can be combined with other traditional or biological cryptographic algorithm to create new security systems.

6. REFERENCES

[1] L. Kari, "DNA Computing: Arrival of Biological Mathematics," *The Mathematical Tntelligencer*, vol. 19, pp. 9–22, 1997.

[2] S.V. Kartalopoulos, "DNA-inspired cryptographic method in optical communications," in *authentication and data mimicking Military Communications Conference*, 2005, pp. 774–779.

[3] G. Z. Cui, L. M. Qin, Y. F Wang and X. C. Zhang, "Information Security Technology Based on DNA Computing," *2007 IEEE International Workshop on Anti-counterfeiting Security, Identification.,* 2007, pp. 288–291.

[4] A. Leier, C. Richter and W. Banzhaf, "Cryptography with DNA binary strands," *Biosystems*, vol. 57, pp. 13–22, 2000.

[5] M. X. Lu, "Symmetric-key cryptosystem with DNA technology, "*Science in China Series F: Information Sciences*, vol. 3, pp. 324–333, 2007.

[6] A. Gehani, T. H. LaBean and J. H. Reif, "DNA-based cryptography,"*DNA Based Computers V. Providence: American Mathematical society*, vol. 54, pp. 233–249, 2000.

[7] C. T. Celland, V. Risca and Bancroft C. "Hiding messages in DNA microdots," *Nature*, vol. 399, pp. 533–534, 1999.

[8] KANG Ning, "A Pseudo DNA Cryptography Method", Independent Research Study Project for CS5231, October 2004.

[9] Sherif T. Amin, Magdy Saeb, Salah El-Gindi, "A DNA-based Implementation of YAEA Encryption Algorithm," IASTED International Conference on Computational Intelligence (CI 2006), San Francisco, Nov. 20, 2006.

[10] O. Tornea and M.E. Borda," DNA Cryptographic Algorithms", MEDITECH 2009, IFMBE Proceedings 26, pp. 223–226, 2009.

[11] Guangzhao Cui, Limin Qin, Yanfeng Wang, Xuncai Zhang," An Encryption Scheme Using DNA Technology", 3rd international conference on Bio-Inspired Computing: Theories and Applications, BICTA 2008, pp. 37 – 42, Oct. 1 2008.

[12] William Stallings. "Cryptography and Network Security", Third Edition, Prentice Hall International, 2003.

[13] *Mona Sabry, Mohamed Hashem, Taymoor Nazmy, Mohamed Essam Khalifa, "*A DNA and Amino Acids-Based Implementation of Playfair Cipher*",(IJCSIS) International Journal of Computer Science and Information Security,Vol. 8, No. 3, 2010.*