Hindawi
Scientific Programming
Volume 2022, Article ID 1846788, 14 pages
https://doi.org/10.1155/2022/1846788

Research Article

Protein Motifs to Hide GA-Based Encrypted Data

Noura A. Mawla and Hussein K. Khafaji

Department of Computer Science, AL-Rafidain University College, Baghdad, Iraq

Correspondence should be addressed to Noura A. Mawla; noora.ahmed.elc@ruc.edu.iq

Received 13 February 2022; Revised 14 July 2022; Accepted 12 August 2022; Published 25 September 2022

Academic Editor: Roberto Natella

The arms of the Internet octopus have reached the ends of the planet. As it has become indispensable in our daily lives, huge amounts of information are transmitted through this network, and it is growing momentarily, which has led to an increase in the number of attacks on this information. Keeping the security of this information has become a necessity today. Therefore, the scientists of cryptography and steganography have seen a great and rapid development in the previous years to the present day, where various security and protection techniques have been used in these two technologies. In this research, it was emphasized to secure the confidentiality and security of the transmitted data between the sending and receiving parties by using both techniques of encryption and steganography. In contrast, where genetic algorithms and logic gates are exploited in an encryption process, in an unprecedented approach, protein motifs are used to mask the encoded message, gaining more dispersion because there are 20 bases used to represent the protein. The real payload gained ranges between 0.8 and 2.666, which outperforms the algorithms that depend on DNA sequences.

1. Introduction

In the era of the current technology revolution, and with the increasing growth of multimedia applications and the wide spread of Internet networks, it has become difficult to maintain the security and confidentiality of the information of individuals and institutions in this digital world. Therefore, providing security and maintaining the confidentiality of information has become very important in our day, so encryption processes are always required [1, 2].

Cryptography has been known since ancient times, as it was used in the military field. It was mentioned that the first encryption process for messaging between the army sectors was accomplished by the Pharaohs. It was also mentioned that the Arabs had made serious attempts in the field of encryption. The Chinese used many methods of cryptography to transmit messages during wars. Their intention was to use encryption to hide the true form of messages, even if they fell into the hands of the enemy, as it would be difficult for them to understand.

Cryptography is the transformation of information from a readable state to a completely opaque state, such that it becomes not useful and does not add information to the reader.

The idea of any encryption system is to hide confidential information in such a way that its meaning becomes incomprehensible to any unauthorized person. The two most common uses of encryption are to securely store data in a computer file or to transmit it over an insecure channel such as the Internet. In either case, the fact that the document is encrypted does not prevent unauthorized people from accessing it, but it does ensure that they cannot understand what they see. In addition to many encryption algorithms, the genetic algorithm GA, or its genetic operations, is used in a variety of aspects of data security [1, 3–7].

In addition to encryption processes, to ensure greater protection of information, another technology is used, that is, hiding information or steganography.

Steganography is the process of hiding information within different media such as video, audio, and image, where these media are considered as an envelope surrounding valuable information. This type of protection increases the security and confidentiality of data. Often, the process of data encryption is confused with the process of

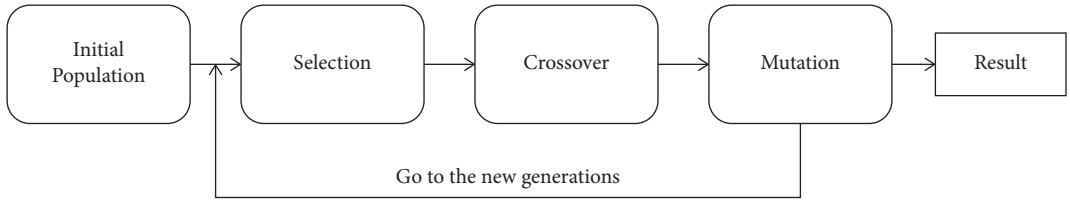

Figure 1: General scheme of a GA.

data hiding. In the process of encryption, the form of data is completely changed and modified, and this change is visible. As for the process of concealment, its results are not visible [8–12]. The encryption and steganography will be used in this paper to increase the security of transmitted data.

Recently, DNA and RNA have been exploited to hide data through their four-letter structures, bases, used as a cover to hide data inside. In this paper, protein motifs are used to hide the encrypted message instead of DNA to obtain more scattering because there are 20 bases used to represent a protein. The next sections present some concepts that are used in this paper, such as GA and bioinformatics [13–16].

2. Genetic Algorithm

In the 1850s, the British naturalist Charles Darwin wrote his book "On the Origin of Species," in which he claimed that organisms evolve to adapt to the environment and compete for the acquisition of nature's resources to achieve their survival. He also established the rule of survival for the strongest beings, and this principle constitutes a natural selection for the most fit organisms, whose survival is expected more than the weak ones. Genetic traits are enhanced in the coming generations, generation after generation. The theory of evolution inspired John Holland to introduce the genetic algorithm [17–19]. A genetic algorithm (GA) is an instance of the evolutionary algorithms (EA) that draws its inspiration from the idea of natural selection. GA is a metaheuristic search algorithm which is frequently employed to provide optimal or nearly optimal solutions to optimization and search problems via biologically inspired operators such as mutation, crossover, and selection. Figure 1 shows some important steps in the GA.

After this elementary abstraction and the presentation of Figure 1, the general structure of a GA can be written as follows:

```
t = 0;
Construct the initial population Pt from randomly generated individuals;
WHILE stopping condition is not satisfied DO.
{
Select fittest individuals for reproduction, Pt;
Create offsprings by crossing individuals;
Eventually mutate some individuals;
t++;
Compute new generation, Pt;
}
```

The algorithm above makes clear that the change from one generation to the next is accomplished based on four genetic operations:

(i) Selection is a technique for choosing which individuals (strings/chromosomes) to be used for reproduction based on their values of fitness function (objective function value).

(ii) Crossover: this technique combines the genetic information of two individuals; if the coding is chosen appropriately, two parents with prominent traits will result in offspring with prominent features.

(iii) During the evolution of the creatures, the genetic information may change at random due to inefficient reproduction or other gene alterations, such as those caused by gamma radiation. In GAs, mutation may be implemented as a probabilistically random distortion of the strings. Maintenance of genetic variety is an advantage of the mutation, and as a result, that local maxima can be avoided.

(iv) Sampling is a procedure which computes a new generation from the previous one and the generated offsprings.

In addition to the previously mentioned GA operations, designers should think about the fitness function, which determines the legibility of the individuals in the population, in addition to the chromosomes' representation. Usually, binary representation is used, but many other representation methods are available, which vary in complexity and efficiency according to the problem to be solved. This paper deals with data security; therefore, binary representation is used to represent the chromosome, that is, plain data, keys, etc.

The performance of genetic algorithms largely depends on crossover and mutation of individuals and, of course, the method of representation and calculation of fitness function. In this research, three types of crossover are used and described as follows [7]:

(i) Single-point crossover: in this type, a single crossover point is initially determined, and then the genes for the first child are copied from the beginning of the chromosome of one of the parents to the crossover point, and the rest is copied from the second parent, and the second child is produced from the remaining genes of the selected two parents. Figure 2 depicts the single-point crossover for chromosomes represented as binary representation.

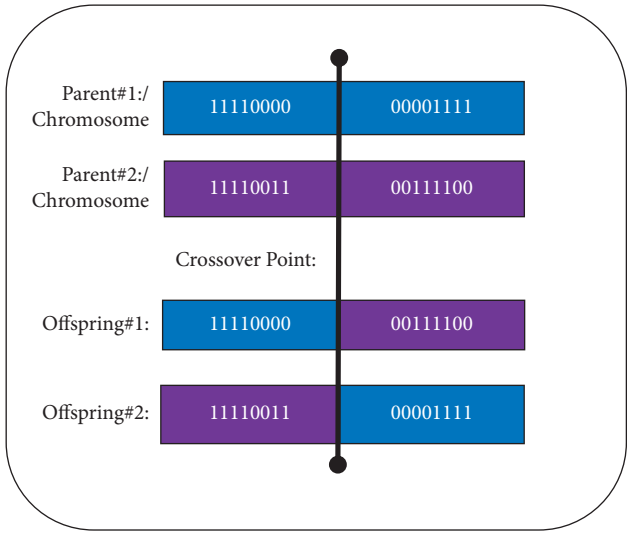

Figure 2: Single-point crossover.

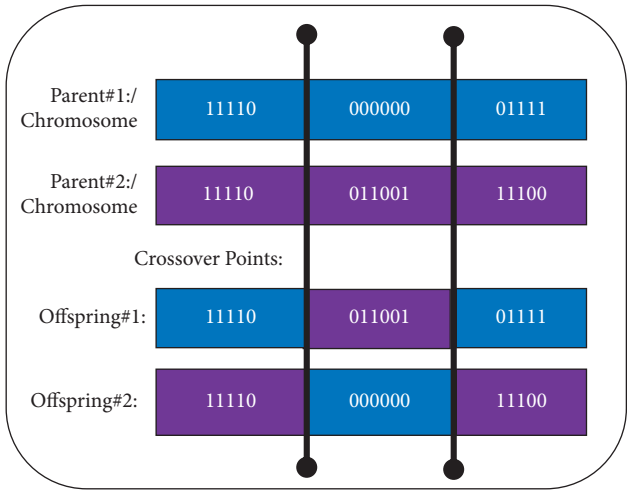

Figure 3: Two-point crossover.

Binary representation is used for illustration because this research deals with streams of byte data.

(ii) Two-point crossover: in this type, two random crossover points are determined for the parental chromosomes and the offspring are formed by exchanging chromosome segments as shown in Figure 3.

(iii) Uniform crossover: in this type of crossover, bits are randomly selected and copied from the first parent or the second parent of the child. Figure 4 illustrates Uniform crossover.

As in crossover, there are many methods of genetic mutation according to the way the chromosome is represented (see Figures 3 and 4). In binary representation, bits are inverted from zero to one and vice versa in predetermined locations as shown in Figure 5.

The randomness of the genetic algorithms included in crossover and genetic mutation operations in the process of encoding and hiding information was exploited by the characteristics of the genetic algorithms, including crossover and mutation, to produce a new generation (cipher text) of information that differs from the old generation (the plaintext).

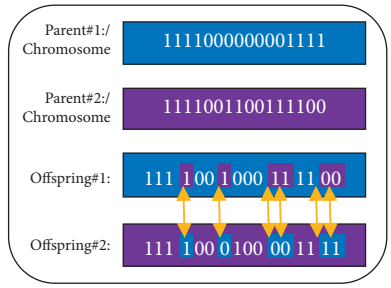

Figure 4: Uniform crossover.

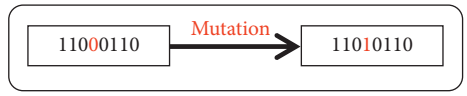

Figure 5: Binary mutation.

3. Protein in Bioinformatics

Bioinformatics, sometimes known as "computational biology," is a growing, developing field that blends biology, information technology, and mathematics to assist in solving biological problems. In reality, computational biology mostly works with modeling biological processes. The design of software tools and algorithms and the analysis and interpretation of biological data utilizing a variety of software tools and specific algorithms are the two primary pillars of bioinformatics. It usually involves genes, DNA, RNA, or protein chains and is particularly useful for comparing genes with other protein chains and with other chains within or between organisms, looking at the evolutionary relationships between organisms, and using patterns in DNA and protein sequences to figure out their functions [20].

While DNA chemically consists of phosphate, sugar, and one of four nucleotides of (G) guanine, (C) cytosine, (A) adenine, and (T) thymine, the protein sequences consist of 20 different kinds of chemical compounds, known as amino acids (bases), and they serve as building blocks of proteins. The typical three- and one-base representations of the protein sequences are used. Typically, single-base representation is used to code databases. These bases are adenine (A), thymine (T), cytosine (C), guanine (G), isoleucine (I), phenylalanine (F), serine (S), glutamine (Q), histidine (H), asparagine (N), aspartic acid (D), arginine (R), glutamic acid(E), lysine (K), leucine (L), valine (V), tryptophan (W), tyrosine (Y), methionine (M), and proline (P) [21, 22]. In this paper, the bases of protein motifs will be used to hide data in specified bits of binary representation of a base.

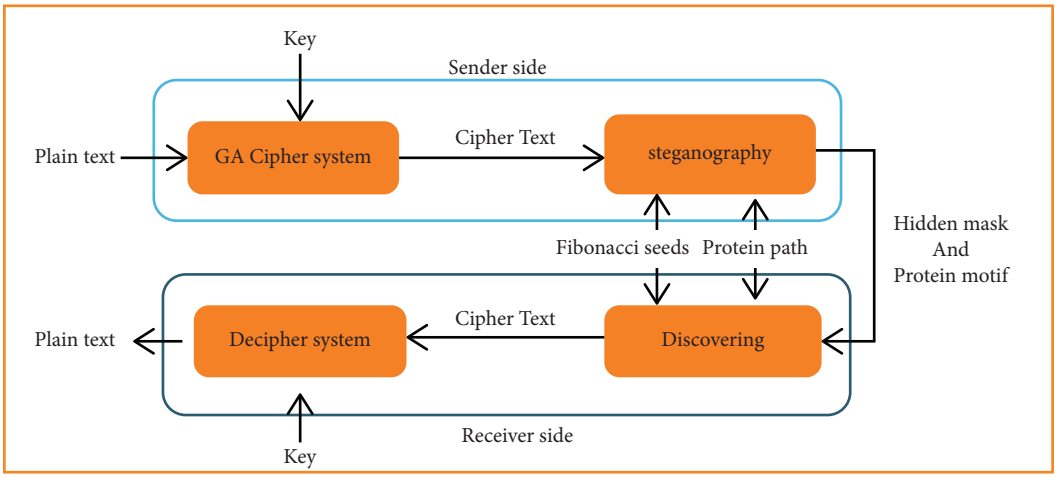

Figure 6: The proposed approach architecture.

4. The Proposed Approach to Hide GA-Based Encrypted Data

The proposed approach can manipulate text, images, audio, video, etc. The data are encrypted on the sender side using a GA-based algorithm and an agreed-upon key, and then the encrypted data are hidden in a protein motif based on the Fibonacci series and a number of protein base combinations. At the receiver side, the hidden data should be uncovered and then it will be decrypted. Figure 6 shows the block diagram of the proposed approach.

The following sections concentrate on the components of the proposed system, which consists of two divisions: the first division is related to data encryption that can be accomplished by any encryption algorithm rather than the presented algorithm, while the second division is related to a new algorithm to hide the encrypted data in protein sequences.

4.1. GA-Based Encryption Algorithm (GABEA). The encryption algorithm converts plain text to cipher text. It depends on logic gates and some characteristics of GA. In addition to the plain text, the input to this algorithm includes a symmetric key, K, agreed upon by the sender and recipient. Algorithm 1 presents the pseudocode of the GA-based encryption algorithm (GABEA).

The first step is responsible for generating a new key by calling the Generated_Key algorithm. The Generated_Key algorithm will be explained in Section 4.2. The second and third steps find the length of the plaintext string and the length of the key. The next steps are responsible for encoding the plaintext using logic gates and some properties of the genetic algorithm, beginning with step 4. This step begins with converting the plaintext to binary code and putting the result in a one-dimensional matrix called Binary_PlainText. Then, the process of XORing the bytes of Binary_PlainText and the bytes of the KEY starts. The result of this logic gate is placed in the BPT_XOR matrix.

The results of the XORing operation in the previous step will be used to accomplish the crossover operation for every two consecutive bytes stored in BPT_XOR. The type of crossover is determined according to (1):

$$\text{TypeOfCrossover} = \text{number of itertion} \mod 4. \quad (1)$$

Four crossover schemes are used: one-point, two-point, multipoint, and uniform crossover according to the values of TypeOfCrossover, 0, 1, 2, and 3, respectively. After the crossover genetic operation, the mutation operation will be done by NOT gate.

4.2. Key-Generating Algorithm. The Key-Generating Algorithm is responsible for generating a key. This algorithm uses the properties of the genetic algorithms and the logic gates in its processes. The pseudocode of this algorithm is presented in Algorithm 2.

The first step divides the entered key, K, into two keys k1 and k2. The length of k1 and k2 will be calculated in the second step and then each character of these keys will be converted to decimal by using ASCII code table. The statement (new_key1[L-(j-1)] = dec_key1[j] – 1) will be applied to all the characters of the first key and the results are stored in horizontal matrix called new_key1 in reverse order. When the second key is entered in the next iteration, (new_key2[j] = dec_key1[j] –(2* j)) will be applied to the parts of the second key and in vertical matrix called new_key2.

The third step of this algorithm is to multiply the elements of the two matrices (new_key1 and new_key2), where an element is taken from new_key1 matrix and this element is multiplied by all the elements of new_key2 matrix. Then, each result is tested; if the value of this product (M) is greater than 256, the mod 256 is calculated for this value.

The final result (M) will be converted into binary codes that are stored in the Key. For more explanation, consider the following example.

Example 1. In this example, the encryption algorithm will be shown in detail. The inputs for this algorithm are shown in Table 1.

```
    Input: PlainText, K.
    Output: En_text.//cipher text.
    Begin.
(1) key = Generated_Key(K);
(2) L_P = length (PlainText);
(3) L_K = length(key);
(4) for i = 1 to L_P do
    {//byte by byte.
    Binary_PlainText[i] = char_to_binary(ascii(PlainText[i]);
    BPT_XOR[i] = XOR_gate(Binary_PlainText[i],key[i]);
    }
(5) i = 1; x = 0;
(6) While(i ≤ L_P)
    {
    Cut_type = x mod 4; //crossover type.
    if(Cut_type = = 0).
    then one_point_crossover(BPT_XOR[i], BPT_XOR[i+1]);
    else if(Cut_type = = 1).
    then two_point_crossover(BPT_XOR[i], BPT_XOR[i+1]);
    else if(Cut_type = = 2).
    then multi_point_crossover(BPT_XOR[i], BPT_XOR[i+1]);
    else uniform _crossover(BPT_XOR[i], BPT_XOR[i+1]);
    for(j = 0; j < 2; j++).
    En_text[j + i] = Not_gate(BPT_XOR[j + i]);//Mutation.
    i = i+2;
    x = x+1;
    }//while.
    Return En_text;
    End.//Algorithm.
```

Algorithm 1: GABEA.

The first step in this algorithm is to create the key that will be used in the encryption process by calling the Generated_Key Algorithm. The letters of key1 are converted to decimal ASCII code and the first equation is applied to each letter. The results are stored in inverse order in a horizontal matrix called new_key1.

The same process is done when entering the second key, but the second equation is used instead of the first equation, and the storage is done by natural arrangement in a vertical matrix called new_key2 as shown in Table 2.

The next step will be a multiplication between the two matrices where each value in the new_key1 matrix will be multiplied by all the values in the new_key2 matrix as the following:

$84^{*}113 = 9492 \quad 51^{*}113 = 5763.$

$84^{*}101 = 8484 \quad 51^{*}101 = 5151.$

$84^{*}109 = 9156 \quad 51^{*}109 = 5559.$

Each product will be tested; if the result is greater than 256, the mod operation will be done for this product and the result will be put in M variable. Each element in M will be transformed from decimal to binary codes and the result will be put in the Key matrix. This process is shown in Table 3.

Here, the first stage of the text encryption process ended, as a new key was created using the Generated_Key Algorithm. Now the second stage of the text encryption process will start using the new key. The first step in this stage is to convert the plaintext (May-21) into binary code. Each character converted to binary code is passed to an XOR gate with a byte chosen from the key matrix. The result will be stored in BPT_XOR. All these steps are shown in Table 4.

The last stage of encoding is limited to crossover operations, which occur between the two successive bytes selected from the BPT_XOR matrix and determine the type of cut for them, where the type of cut depends on the sequence of two bytes in the BPT_XOR matrix, as we explained earlier when explaining the encryption algorithm in section (4.1). These operations are declared in Table 5.

The resulting chromosomes are passed to NOT gate as shown in Table 6 and the product through this gate is stored in the En_text matrix.

Finally, the text is encrypted and is now ready to be hidden in protein sequence.

4.3. Steganography Using Protein Motif. In bioinformatics, the protein motif is represented using 20 bases. Table 7 shows sufficient information about the protein representation.

The hiding algorithm divides each byte of the encrypted message into 3 parts: 3 bits, 3 bits, and 2 bits partitions. Therefore, each byte of the encrypted message requires 3 bytes to be covered. Table 8 presents the protein bases matching the possibilities of 3 bits starting from the least significant bit of the base, while Table 9 shows the protein

```
    Input: K.
    Output: Key.//generated key
    Begin
(1) divide Key(K, K1, K2); //divide K into two divisions k1 and k2
(2) For i = 1 to 2 do
    {
    L[i] = length(Ki);
    For j = 1 to L[i] do
    {
    if (i = = 1) then
    {
    dec_key1[j] = char_to_decimal(ascii(K[j]));
    new_key1[ L-(j-1) ] = dec_key1[j] – 1;
    }//if
    else {
    dec_key2[j] = char_to_decimal(ascii(K[j]));
    new_key2[j] = dec_key2[j] –(2* j);
    }//else
    }//for j
    }//for i
    k = 1;
(3) for i = 1 to L [1] do
    for j = 1 to L [2] do
    {
    M = new_key1[i]* new_key2[j];
    If (M > 256) then M = M mod 256;
    key[k] = decimal_to_binary(ascii(M) );
    k = k+1;
    }//for
    Return Key;
    End.//Algorithm
```

Algorithm 2: Key-Generating Algorithm.

Table 1: Plain text and used key.

Input of encryption algorithm		
	Key = 4Usis	
Plaintext	Key1	Key2
May-21	4U	Sis

Table 2: Key construction.

	Symbol	dec_key	Equations	New_key
Key1	4	52	new_key1[L-(j-1)] = dec_key1[j]– 1	Horizontal_M
	U	85		84 51
Key2	S	115	new_key2[j] = dec_key1[j] –(2* j)	Vertical _M
				113
	I	105		101
	S	115		109

Table 3: Binary coding of the key.

	M					
Multiplication results	9492	8484	9156	5763	5151	5559
M % 256	20	36	196	131	31	183
Key	00010100	00100100	11000100	10000011	00011111	10110111

Table 4: XOR_gate results.

PlainText	M	a	y	-	2	1
Binary PlainText	01001101	01100001	01111001	00101101	00110010	00110001
Key	00010100	00100100	11000100	10000011	00011111	10110111
BPT_XOR = BinaryPT (XOR_get)key	01011001	01000101	10111101	10101110	00101101	10000110

Table 5: Crossover operations.

Types of cut	One-point cut	Two-point cut	Multipoint cut (three points or more)
Sequence of chromosome	Crossover (1 and 2)	Crossover (3 and 4)	Crossover (5 and 6)
Crossover operation (A&B)	01011001 01000101	10111101 10101110	00101101 10000110
Result of crossover (C&D)	01010101 01001001	10101101 10111110	10001110 00100101

Table 6: Mutation operations.

Crossover result	01010101	01001001	10101101	10111110	10001110	00100101
Not_gate result	10101010	10110110	01010010	01000001	01110001	11011010

Table 7: Protein representation in bioinformatics.

Iupac amino acid code	Binary	Three-letter code	Amino acid
A	01000001	Ala	Alanine
C	01000011	Cys	Cysteine
D	01000100	Asp	Aspartic acid
E	01000101	Glu	Glutamic acid
F	01000110	Phe	Phenylalanine
G	01000111	Gly	Glycine
H	01001000	His	Histidine
I	01001001	Ile	Isoleucine
K	01001011	Lys	Lysine
L	01001100	Leu	Leucine
M	01001101	Met	Methionine
N	01001110	Asn	Asparagine
P	01010000	Pro	Proline
Q	01010001	Gln	Glutamine
R	01010010	Arg	Arginine
S	01010011	Ser	Serine
T	01010100	Thr	Threonine
V	01010110	Val	Valine
W	01010111	Trp	Tryptophan
Y	01011001	Tyr	Tyrosine

Table 8: The protein bases matching 3 bits.

3 bit pattern	Base
000	P, H
001	Q, A, I, Y
010	R
011	C, K, S
100	D, L, T
101	E, M
110	F, N, V
111	G, W

Table 9: The protein bases matching 2 bits starting from 4th bit.

bit pattern	Base
00	A, C, D, E, F, G
01	H, I, K, L, M, N
10	P, Q, R, S, T, V, W
11	Y

bases matching 2-bit patterns starting from the 4th bit of the base.

Accordingly, if the sender and receiver agreed upon a base for each 2-bit or 3-bit pattern, this would lead to 864 possible paths for 3-bit patterns and 210 paths for 2-bit patterns, as shown in Figure 7, which represents part of the tree of possible combinations of the protein bases matching the 3-bit patterns.

Each possible path is assigned a unique number. For example, the combination P, Q, R, C, D, E, F, and G is assigned path number 1, indicating that it will be used to conceal 000, 001, ..., 111. The path number must be agreed upon by the sender and receiver exactly as the encryption key. At the same time, they should agree upon the path number of 2-bit patterns. Algorithm 3. shows the details of the hiding algorithm.

The locations of the hidden data in the protein motif are determined according to the Fibonacci sequence. Therefore, the first and second "seed" should be known to the sender and the receiver; therefore, they are given as input in Algorithm 3. To avoid the rapid growth of Fibonacci terms, the following formula is suggested to determine the number of amino acids generated for each byte of encrypted data:

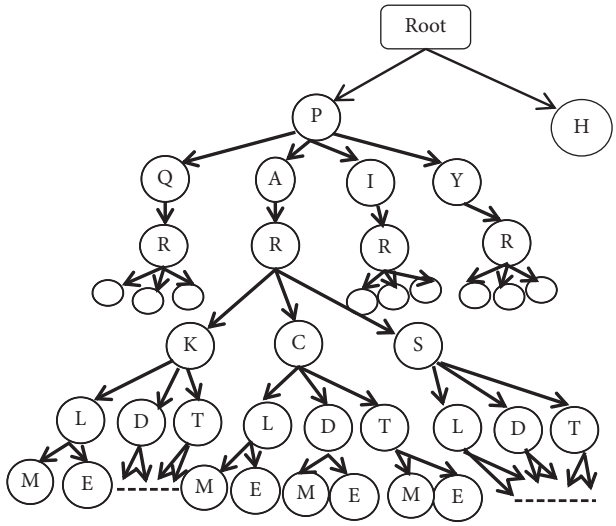

Figure 7: Subtree of all possible combinations of protein bases.

```
    Input: En_text, Fibonacci_Seed1, Fibonacci_Seed2, P2bits_path, P3bits_path;
    Output: Protein_cover;
    Begin.
    //Initialization.
    0:Protein_cover = “”; new = Fibonacci_Seed2;
(1) while (abs(sin(Fibonacci_Seed1))<0.3 or abs(sin(Fibonacci_Seed2))<0.3)
    new = Generate_fibonacci_term();//Generate a new Fibonacci term.
(2) while (not_end_of(En_text)) do
    {
(3) Byte = get_byte(En_text);//get a byte from the cipher text
    //return 2× 3 bits patterns and 1× 2 bits pattern starting from LSB.
(4) Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
(5) sin4FibonacciTerm = trunc(abs(sin(new))*10);
(6) if (sin4FibonacciTerm ≥ 3)
(7) { Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
    //fill the rest with protein bases.
(8) Randomly generate (sin4FibonacciTerm −3)) protein bases, RestBases;
    //string concatenation.
(9) Protein_cover = Protein_cover + bases_3 + RestBases;
(10) }
(11) new = Generate_fibonacci_term();
(12) }//while
(13) return binary code of Protein_cover to be sent to the receiver;
    End.//Hiding Algorithm.
(14) integer Generate_fibonacci_term()//Function to generate new Fibonacci term.
    Begin.
(15) new = Fibonacci_Seed2 + Fibonacci_Seed1;
(16) Fibonacci_Seed1 = Fibonacci_Seed2; Fibonacci_Seed2 = new;
(17) return new;
    End.
```

Algorithm 3: Hiding algorithm.

$$\text{generated number for each Fibonacci Term} = \text{trunc}\left(\text{abs}\left(\sin\left(\text{FibonacciTerm}\right)\right) * 10\right). \tag{2}$$

The absolute value is used to avoid the negative values of the sine function. These values will be multiplied by 10 to obtain a number ranging from 0 to 10 for each Fibonacci term. Recall that the proposed algorithm specifies 3 amino acids for each encrypted byte for hiding; therefore, each value less than 3 will be ignored and a new Fibonacci term will be calculated. In this way, the accepted values range from 3 to 10. For this reason, Step 1 is iterated to generate a new Fibonacci term when one or both seeds are initially less than 3. The while statement in Step 2 will be executed when 2 Fibonacci terms with values greater than 3 become available. It gnaws the text byte by byte via the get_byte function, which gets a byte and stores it in the Byte variable. Step 4 divides the Byte variable's content into three parts as follows ($x_8x_7|x_6x_{5x4}|x_3x_2x_1$), which corresponds to the variables P2bits, P3bits2, and P3bits1, respectively. Step 5 is an implementation of (2), which generates a number that ranges from 1 to 10. The result is stored in the sin4FibonacciTerm variable. The number which is less than 3 will be ignored and a new number will be generated in Step 11. Otherwise, steps 7, 8, and 9 will be executed under the control of the if statement in step 6. Step 7 generates 3 protein bases according to the agreed upon path number of 2-bit patterns and the path number of 3-bit patterns. These three bases are concatenated in the base_3 variable. Step 8 generates the rest (sin4FibonacciTerm-3) bases and stores them in the RestBases variable. Step 9 combines the bases generated in steps 7 and 8 with the previous value of the Protein_cover variable. This means that the proposed algorithm generates a corresponding number of protein amino acids; the first three are generated according to the agreed protein paths, while the rest will be generated randomly from the protein amino acids to maintain neutrality of the protein sequence. The first amino acid will be chosen to hide the 7th and 8th bits of a byte of cipher text. The next two amino acids of a protein motif will be chosen to hide the remaining 6 bits of the byte of cipher text.

Example 2. Table 10 presents a tracing of the proposed hiding algorithm. Let the 3-bit pattern be P, Q, R, C, D, E, F, and G and the 2-bit pattern be A, H, Q, and Y. Also, suppose that the Fibonacci sequence starts from 8 to 10. To hide the encrypted message of the encryption algorithm example, consider Table 10. The generated Fibonacci terms will be 8, 10, 18, 28, 46, 74, 120, 194, and 314 and the corresponding sequence generated according to (2) will be 1, 1, 3, 4, 7, 9, 8, 2, and 7.

4.4. Discovery Phase. This phase is the first one on the receiver side. It recovers the data from protein cover depending on the Fibonacci sequence and the 2-bit and 3-bit patterns. The self-documented algorithm, Algorithm 4, depicts this process.

This algorithm simply performs its duty by generating the sequence of numbers by applying Equation 2 on the Fibonacci numbers and extracts two bits from the specified byte starting from bit number four and then extracts three bits from each one of its two consecutive bytes. The extracted bits are collected to form one byte of the hidden data that will be decrypted by the Decryption algorithm.

4.5. Decryption Algorithm. As it is mentioned in Section 4.2, the encrypted message is hidden in a monad protein motif. At the receiver side, the message will be uncovered and passed to the Decryption algorithm to obtain the plain text. Algorithm 5. depicts the pseudocode of this algorithm.

The initial steps are similar to those of the encryption algorithm. Starting from step 5, many decoding steps will be done. Step 5.1 demutates the encrypted message using a NOT gate for each bit, and the result will be stored in Out_Not. Step 5.2 performs the crossover GA operation in the same manner as the crossover of an encryption algorithm. Step 7 accomplishes the XORing of the crossover message bytes and the key bytes with the same index. The XORed bytes are then converted to decimal numbers depending on the ASCII code table.

Example 3. The decoding process begins when the protein sequence reaches the recipient and the cipher text is extracted from it. It starts first by generating the key as shown in Table 11, which will be used in the decryption process. This key is the same key used in the encryption process.

Now we will start the decoding process by selecting two successive bytes from the ciphertext (En_text) and inserting them into the NOT gate sequentially. Table 12 declares the results of the NOT gate operation.

The two results obtained from NOT gate will be taken and the crossover operation will be performed on them after determining the type of cutting used as explained previously. These steps will be repeated for all bytes in the cipher text. The resulting chromosomes from crossover operations will be stored in the Out_Not matrix instead of their parents. The results of these operations are shown in Table 13.

In the last stage of decoding, a byte of the Out_Not matrix and a byte of the generated key will be selected and fed to the XOR gate. The resulting byte of this gate will be converted to decimal and then tested. If it is greater than 256, the mode of 256 will be taken for it. This process will be repeated for all the elements of the Out_Not matrix. All of these operations are shown in Table 14.

5. Data Payload of the Proposed Steganography Algorithm

In steganography, the payload is represented by the maximum number of bits that can be covered by the cover medium. In the cases of bio-sequences such as DNA, RNA, and protein, the payload unit is bit per nucleotide (bpn). In the proposed steganography algorithm, each byte of the encrypted data, E, requires 3 bytes to be covered; that is, if the length of E is L bytes, then (3L) protein bases actually embed the bytes of E. Recall (2), where the accepted range of the generated protein bases is 3–10 for a term of Fibonacci sequence. Hence, the length of the protein cover which hides E can be defined in three cases: the worst case,

Table 10: Tracing of hiding algorithm.

Statements	Values	Remarks
Protein_cover = “”;	“”	
While (abs(sin(Fibonacci_Seed1))<0.3 or abs(sin(Fibonacci_Seed2))<0.3) { new = Generate_fibonacci_term(); }	Fib. Sequence: 8,10, new = 18. Eq.2 sequence: 1, 1	Seed1 = 8 Seed2 = 10 New = 18

Byte = get_byte(En_text);
Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
sin4FibonacciTerm = trunc(abs(sin(new))* 10);
If (sin4FibonacciTerm ≥ 3
Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
new = Generate_fibonacci_term();
	Byte = 10101010
	sin4FibonacciTerm = 3
	Protein_cover = QER
	New = 28
		8, 10, 18, 28.
		The protein sequence is in binary

Byte = get_byte(En_text);
Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
sin4FibonacciTerm = trunc(abs(sin(new))* 10);
If (sin4FibonacciTerm ≥ 3
Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
Randomly generate (sin4FibonacciTerm−3)) protein bases, RestBases;
	Byte = 10110110
	sin4FibonacciTerm = 4
	Protein_cover = QERQFFT
	New = 46
	Amino acids in blue color are randomly generated
		8, 10, 18, 28, 46

Byte = get_byte(En_text);
Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
sin4FibonacciTerm = trunc(abs(sin(new))* 10);
If (sin4FibonacciTerm ≥ 3
Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
Randomly generate (sin4FibonacciTerm−3)) protein bases, RestBases;
	Byte = 01 010 010
	sin4FibonacciTerm = 7
	Protein_cover = QERQFFTHRRDEWV
	New = 74
		8, 10, 18, 28, 46, 74

Byte = get_byte(En_text);
Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
sin4FibonacciTerm = trunc(abs(sin(new))* 10);
If (sin4FibonacciTerm ≥ 3
Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
Randomly generate (sin4FibonacciTerm−3)) protein bases, RestBases;
	Byte = 01000001
	sin4FibonacciTerm = 9
	Protein_cover = QERQFFTHRRDEWVHPQNMGFLA
	New = 120
		8, 10, 18, 28, 46, 74, 120

Byte = get_byte(En_text);
Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
sin4FibonacciTerm = trunc(abs(sin(new))* 10);
If (sin4FibonacciTerm ≥ 3
Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
Randomly generate (sin4FibonacciTerm −3)) protein bases, RestBases;
	Byte = 01110001
	sin4FibonacciTerm = 8
	Protein_cover = QERQFFTHRRDEWVHPQNMGFLAHFQQQVTS
	New = 194
		8, 10, 18, 28, 46, 74, 120, 194

Byte = get_byte(En_text);
Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
sin4FibonacciTerm = trunc(abs(sin(new))* 10);
If (sin4FibonacciTerm ≥ 3
Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
Randomly generate (sin4FibonacciTerm −3)) protein bases, RestBases;
	Byte = 11011010
	sin4FibonacciTerm = 2 ignored
	Protein_cover = QERQFFTHRRDEWVHPQNMGFLAHFQQQVTS
	New = 314
		8, 10, 18, 28, 46, 74, 120, 194, 314

Table 10: Continued.

Statements	Values	Remarks

Byte = get_byte(En_text);
Divide_Byte(Byte, P2bits, P3bits2, P3bits1);
sin4FibonacciTerm = trunc(abs(sin(new))* 10);
If (sin4FibonacciTerm ≥ 3
Generate 3 protein bases according to the P2bits_path and P3bits_path, bases_3;
Randomly generate (sin4FibonacciTerm –3)) protein bases, RestBases;
	Byte = 11011010
	sin4FibonacciTerm = 7
	Protein_cover = QERQFFTHRRDEWVHPQNMGFLAHFQQQVTSYCRHIYG
	New = . . .
		8, 10, 18, 28, 46, 74, 120, 194, 314

```
Input: Protein_cover.
Fibonacci_Seed1.
Fibonacci_Seed2;
Output: Encrypted Data;
Begin.
0://Initialization.
Enc_data = ""; new = Fibonacci_Seed2;
while (abs(sin(Fibonacci_Seed1))<0.3 or abs(sin(Fibonacci_Seed2))<0.3).
{ new = Generate_fibonacci_term(); }
Step = 1;
(1) while (not_end_of(Protein_cover)) do
{//get byte from cipher text.
sin4FibonacciTerm = trunc(abs(sin(new))*10);
Byte1 = get_byte_at_location(Protein_cover, Step);
P2bits = extract(Byte1, 4,2);//extract 2 bits starting from bit#4.
Byte2 = get_byte_at_location(Protein_cover, Step+1);
P3bits2 = extract(Byte2, 1,3);//extract 3 bits starting from bit#1.
Byte3 = get_byte_at_location(Protein_cover, Step+2);
P3bits1 = extract(Byte3, 1,3);//extract 3 bits starting from bit#1.
Enc_data = Enc_data + concatenate(P2bits, P3bits2, P3bits1);
Step = Step + sin4FibonacciTerm;
new = Generate_fibonacci_term();
}//while.
return Encrypted data, Enc_data, to be decrypted by the decryption algorithm;
End//Hiding Algorithm.
```

Algorithm 4: Discovery Algorithm.

the average case, and the best case. The worst case happens when the sine of all Fibonacci terms is 1, which leads to generating 10 protein bases, and the best case scenario happens when the sine of all Fibonacci terms is 0.3, which leads to generating 3 protein bases. Practically, the best and worst cases did not occur for all Fibonacci terms. Therefore, the real payload ranges between 0.8 and 2.666 according to (3), 4, and 5.

$$\text{worst payload} = \frac{(\text{L} * 8\,\text{bit})}{10 * \text{L nucleotides}} = \frac{8}{10} = 0.8\,\text{bpn}, \quad (3)$$

$$\text{best payload} = \frac{(\text{L} * 8\,\text{bit})}{3 * \text{L nucleotides}} = \frac{8}{3} = 2.666\,\text{bpn}, \quad (4)$$

$$\text{average payload} = \frac{(\text{L} * 8\,\text{bit})}{(10+3)/2 * \text{L nucleotides}} = \frac{8}{6.5} = 1.23\,\text{bpn}. \quad (5)$$

6. Experimental Results

Five data files of different sizes are used to test the execution time and the scalability of the proposed algorithms. Three or more Fibonacci series are applied to each data file. Table 15 depicts the data file properties and the results obtained.

The fifth data file is the C++ programs used to implement the proposed algorithms. The results show that the proposed algorithms are scalable to the data size and the payload

```
Input: En_text,K;
Output: PlainText;
Begin.
(1) key = Generated_Key(K);
(2) L_P = length (En_text);
(3) L_K = length(key);
(4) i = 1; x = 0;
(5) While(i ≤ L_P)
    {
    for (j = 0; j < 2; j++).
    (5.1)Out_Not[j + i] = Not_gate(En_text[j + i]);//complement the cipher text
    (5.2)Cut_type = x mod 4;
    if (Cut_type = = 0).
    then one_point_crossover(Out_Not[i], Out_Not[i+1]);
    else if(Cut_type = = 1).
    then two_point_crossover(Out_Not[i], Out_Not[i+1]);
    else if(Cut_type = = 2).
    then multi_point_crossover(Out_Not[i], Out_Not[i+1]);
    else uniform_point_crossover(Out_Not[i], Out_Not[i+1]);
    i = i+2;
    x = x+1;
    }//while.
(6) j = 1;
(7) for i = 1 to L_P do
    {
    Out_XOR[i] = XOR_gate(Out_Not[i],key[j]);
    Dec[i] = binary_to_decimal(ascii(Out_XOR[i]);
    If(Dec[i]>257) then Dec[i] = Dec[i] mod 256;
    PlainText[i] = decimal_to_char(ascii(Dec[i]));
    If (j = = L_K) then j = 1;
    Else j = j+1;
    }//for.
    End.//Algorithm.
```

Algorithm 5: Decryption algorithm.

Table 11: The generated key.

Key	00010100	00100100	11000100	10000011	00011111	10110111

Table 12: Mutation using NOT gate.

En_text	10101010	10110110	01010010	01000001	01110001	11011010
Result of NOT gate	01010101	01001001	10101101	10111110	10001110	00100101

Table 13: Crossover operations.

Types of cut	One-point cut	Two-point cut	Multipoint cut (three points or more)
Sequence of chromosome	Crossover (1 and 2)	Crossover (3 and 4)	Crossover (5 and 6)
Crossover (A&B)	01010101 01001001	10101101 10111110	10001110 00100101
Result of crossover (C&D)	01011001 01000101	10111101 10101110	00101101 10000110

Table 14: The XOR_Get operations to extract the plaintext.

Out_Not (crossover result)	01011001	01000101	10111101	10101110	00101101	10000110
Key	00010100	00100100	11000100	10000011	00011111	10110111
Out_Not (XOR) key	01001101	01100001	01111001	00101101	00110010	00110001
Decimal	77	97	121	45	50	49
PlainText	M	A	Y	-	2	1

Table 15: The properties and results of data file.

Data file	Size	Fibonacci series number	Encryption time	Avg. of hiding time	Avg. of discovery time	Decryption time	Avg. of generated cover size (base)	Avg. of payload (bpn), capacity
1	1256	3	2	4	4.5	2.2	2088	1.66
2	2015	3	3	5.2	6	3	3699	1.83
3	3777	3	5	7.1	7.33	5.1	5482	1.45
4	6123	4	8.4	9.3	9.4	8.5	8150	1.33
5	10277	4	11.2	10.9	12	11.5	15743	1.53

Table 16: Comparison between the proposed algorithm and four DNA-based algorithms.

Research	Approach	Payload	Functionality conservation	Blindness
Khalifa, 2015 [16]	LSBase	0.333	√	√
Shimanovsky [23]	Mathematical encoding	Not defined	√	√
Chang, 2007 [24]	Lossless compression-based	0.78	×	√
Chang, 2007 [24]	Difference expansion-based	0.11	×	√
Shiu, 2010 [25]	Insertion method	0.58	×	×
Shiu, 2010 [25]	Complementary method	0.07	×	×
Shiu, 2010 [25]	Substitution method	0.82	×	×
The proposed algorithm	GA and protein coding	1.23 average	√	√

revolves around the average payload. Also, we considered that the results are sensitive to the initial terms of the Fibonacci series.

6.1. Comparison with Other Techniques. The idea of hiding information in protein motifs is unprecedented, so it is impossible to compare it with other hiding methods. However, to demonstrate the efficiency, we will compare it with some methods that depend on DNA chains. Table 16 shows a comparison among four DNA-based algorithms [16, 23–25] and the proposed one. The comparison depends on three factors: blindness, which means that the recovery process does not need a reference to the cover sequence; function conservation, which means the identicalness of amino acid sequence after and before the hiding process; and capacity. The table shows that the proposed algorithm outperforms the mentioned algorithms.

7. Conclusion and Future Works

In this research, we protect data by using robust GA-based cipher algorithm and novel protein motif steganography techniques to obtain a high level of security. GA operations provided high degree of randomness for the bits of plain data while the wide range of protein bases, 20 bases, makes the concealment process unquestionable. The high number of protein bases provides large numbers of 2-bit and 3-bit patterns which support the encryption and hiding processes. In this paper, the effective features of GA are utilized to cipher the plain data and then generate a protein sequence to hide the encrypted data depending on a Fibonacci series to keep the natural distribution of the protein sequence. The protein cover is generated according to the initial Fibonacci terms and the agreed protein paths; therefore, hidden data can be extracted without references to predefined sequences which leads to regarding the proposed method as blind and functionally conserved.

The proposed stego-protein algorithm provides the highest capacity among its relative DNA-based algorithms. The proposed algorithms' scalability makes them applicable in a variety of fields, including image, audio, video, bioinformatics, cryptosystems, and so on. We intend to use the proposed algorithms in future work in the so-called medical cyber physical systems, or MCPSs [26–29], because they represent an environment rich in data of various types, formats, and resources.

Data Availability

The algorithms used in this work are generated and created in the paper.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] Y. B. Gyawali and B. Subedi, *Encryption Algorithm Advanced Encryption Standard*, Caldwell University, U. S. A, 2020.

[2] M. Y. T. Irsan and S. C. Antoro, "Text Encryption Algorithm based on Chaotic Map," *Journal of Physics: Conference Series*, vol. 1341, p. 6, 2019.

[3] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *"Handbook Of Applied Cryptography"*, 1 May 2020.

[4] R. S. S. Ítalo, S. P. Ricardo, S. D. M. Davi, and O. S. J. Paulo, "Implementation of RSA cryptography Algorithm in language C in Exchange of text messages," *International Journal for Innovation Education and Researchwww.ijier.net*, vol. 7, no. 10, pp. 427–440, 2019.

[5] T. Alam, S. Qamar, D. Amit, and B. Mohamed, "Genetic algorithm: reviews, implementations, and applications," *International Journal of Engineering Pedagogy (IJEP)*, vol. 34, 2020.

[6] S. S. A. Al-Murieb and F. J. Abd Al-Razaq, "Data encryption employing genetic algorithm based on a new algorithm of key

generator," *Journal of Engineering and Applied Sciences*, vol. 12, pp. 8679–8682, 2017.

[7] C. Chunka, R. S. Goswami, and S. Banerjee, "An efficient mechanism to generate dynamic keys based on genetic algorithm," *Security Privacy*, vol. 4, 2018.

[8] D. Lee, "Steganography of Complex Networks," 2021, https://www.researchgate.net/publication/355496260_Steganography_of_Complex_Networks.

[9] Y. Tong, Y. Liu, J. Wang, and G. Xin, *Text Steganography on RNN-Generated Lyrics*, Mathematical Biosciences and Engineering, 2019.

[10] T. Sarkar, K. Selvakumar, L. Motiei, and David, "Message in a molecule," *Nature Communications*, vol. 34, May 2016.

[11] A. Ahmed and A. Ahmed, "A secure image steganography using LSB and double XOR operations," *IJCSNS International Journal of computer science and Network Security*, vol. 20, no. 5, 2020.

[12] S. Parikibandla and S. Alluri, "Low area field-programmable gate array implementation of PRESENT image encryption with key rotation and substitution," *ETRI Journal*, vol. 12, 2021.

[13] S. Adil Kadum, "Data hiding based DNA issues: a review," *Journal of University of Babylon for Pure and Applied Sciences*, vol. 28, no. 2, 2020.

[14] M. Liu and G. Ye, "A new DNA coding and hyper chaotic system based asymmetric image encryption algorithm," *Mathematical Biosciences and Engineering*, vol. 18, no. 4, pp. 3887–3906, 2021.

[15] Na Dokyun, *"DNA Steganography: Hiding Undetectable Secret Messages within the Single Nucleotide Polymorphisms of a Genome and Detecting Mutation-Induced Errors"*, Microbial Cell Factories, 2020.

[16] A. Khalifa and S. Helmy Hamad, "Hiding secret Information in DNA sequences using silent mutations", january 2015," *British Journal of Mathematics & Computer Science*, vol. 11, no. 5, pp. 1–11.

[17] J. H. Holland, *"Adaptation In Natural And Artificial Systems"*, The MIT Press, Cambridge, MA, 1992.

[18] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard, *Induction: Processes Of Inference, Learning, and Discovery". Computational Models of Cognition and Perception*, The MIT Press, Cambridge, MA, 1986.

[19] B. Ulrich, *Genetic Algorithms: Theory and Applications"*, SCCH, 1999.

[20] S.-Y. Zhang and S.-L. Liu, *Brenner's Encyclopedia of Genetics*, Second Edition, 2013.

[21] H. K. Al-Khafaji and Z. M. Jameel, "A new Approach to DNA, RNA, and protein motifs templates Visualization and Analysis via compilation technique," *IOSR Journal of Computer Engineering*, vol. 19, no. 02, pp. 15–25, 2017.

[22] A. B. Yousif, H. K. Al-Khafaji, and T. Abbas, "A survey of exact motif finding algorithms," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 27, no. No. 2, pp. 1109–1118, August 2022.

[23] B. Shimanovsky, J. Feng, and M. Potkonjak, *Hiding data in DNA", Information Hiding*, Springer, vol. 2578, pp. 373–386, 2002.

[24] C. C. Chang, T. C. Lu, Y. F. Chang, and R. C. T. Lee, "Reversible data hiding schemes for deoxyribonucleic acid (DNA) medium," *International Journal of Innovative Computing, Information and Control*, vol. 3, no. 1, p. 16, 2007.

[25] H. J. Shiu, K. L. Ng, J. F. Fang, R. C. T. Lee, and C. H. Huang, "Data hiding methods based upon DNA sequences," *Information Sciences*, vol. 180, pp. 2196–2208, 2010.

[26] Xu Cheng, F. Chen, X. Dong, H. Sun, and H. Cheng, *"Design Of a Secure Medical Data Sharing Scheme Based On Blockchain"*, Springer Science+Business Media, LLC, part of Springer Nature, 2020.

[27] F. Chen, YonglongLuo, Ji Zhang et al., *An Infrastructure Framework for Privacy protection of Community Medical Internet of Things Transmission protection, Storage protection and Access Control*Springer Science+Business Media, New York, 2017.

[28] F. Chen, Y. Tang, C. Wang et al., *"Medical Cyber–Physical Systems: A Solution To Smart Health And the State Of the Art"*, IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, 2021.

[29] F. Chen, J. Huang, C. Wang et al., *"Data Access Control Based On Blockchain In Medical Cyber Physical Systems"*, Hindawi, Security and Communication Networks, 2021.