Reversible Data Hiding for DNA Sequences and Its Applications

Qi Tang, School of Information Science, University of Science and Technology of China, Hefei, China

Guoli Ma, School of Information Science, University of Science and Technology of China, Hefei, China

Weiming Zhang, School of Information Science, University of Science and Technology of China, Hefei, China

Nenghai Yu, School of Information Science, University of Science and Technology of China, Hefei, China

ABSTRACT

As the blueprint of vital activities of most living things on earth, DNA has important status and must be protected perfectly. And in current DNA databases, each sequence is stored with several notes that help to describe that sequence. However, these notes have no contribution to the protection of sequences. In this paper, the authors propose a reversible data hiding method for DNA sequences, which could be used either to embed sequence-related annotations, or to detect and restore tampers. When embedding sequence annotations, the methods works in low embedding rate mode. Only several bits of annotations are embedded. When used for tamper detection and tamper restoration, all possible embedding positions are utilized to assure the maximum restoration capacity.

Keywords: *Degenerate Bases, Embedding Positions, DNA Sequences, Reversible Data Hiding, Sequence Annotations*

1. INTRODUCTION

DNA, as we all know, is the genetic codes for human beings and other plants and animals. A DNA sequence is a string of 4 nitrogenous bases: adenine (*A*), thymine (*T*), guanine (*G*), and cytosine (*C*). Most of current discovered DNA sequences are stored in GenBank of NCBI and other DNA databases. Normally, a sequence file in GenBack consists of two parts: the original sequence and the annotation. The annotation includes several labels such as LOCUS, DEFINITION, ACCESSION, SOURCE and FEATURES, which help to describe the original sequence. These annotations usually occupy a lot of the storage space of the database. Besides the storage problem, once the sequence is tampered by evil attackers, researches based on those tampered sequences will result in disastrous consequences. Thus, a method that

DOI: 10.4018/ijdcf.2014100101

could either solve the storage problem or detect and restore tampers on DNA sequence is of much help.

Reversible data hiding is a widely used technique in digital media field for annotation, watermarking and authentication. As its name indicates, reversible data hiding losslessly hides data into different media. When this data is extracted, the original media are perfectly retrieved. With different types of data embedded, reversible data hiding can be applied to different applications. For example, embedding media-related annotations could make such media self-describable. This is extensively used in medical image processing. In addition, if the content reference information is embedded, we say that the medium is watermarked. In general, there are three types of watermark: fragile watermark, semi-fragile watermark, and robust watermark. Fragile watermark fails to be detectable so it is widely used for tamper detection. Semi-fragile watermark is designed to resist benign transformations in order to detect malignant transformation. A robust watermark can tolerate a designated transformation and therefore, copy protection applications always choose this kind of watermark.

Introducing the traditionally used reversible data hiding techniques to DNA sequences will help to solve the storage problem and to authenticate the sequences. Some scholars have proposed their data hiding and watermarking methods. Shimanovsky et al. (Shimanovsky, 2003) firstly proposed a method in which utilizing the redundancy of codon to amino acid mapping. There are $4^3 = 64$ different codon combinations with three Nucleotides. But all possible amino acids and signals are 22. So a codon in a sequence can be substituted by its peer codons that will be translated into the same amino acid as the original one does. Hence, several bits can be embedded into each codon. In (Chang, 2007), Chang et al. provide two methods.One of them is based on lossless compressing techniques. They losslessly compressed the DNA sequence and then append the secret data to the end of the compressing result. Another one adopted the difference expansion way to hide data, which is widely used in digital image data hiding. Another three methods: the insertion method, the complementary pair method and the substitution method were proposed by Shiu in (Shiu, 2010). Unfortunately, however, in order to restore the original content, all three methods have to make sure that a reference DNA sequence is transmitted to the client. With this premise, all three methods cannot be denoted as reversible data hiding methods. After that, Guo et al. (Guo, 2012) improved the substitution method. But the reference DNA sequence is still needed.

In this paper, we propose a new reversible data hiding method on DNA sequences. In this method, we utilized degenerate base symbols as labels to mark DNA sequences. Hence no reference DNA sequences are needed. Based on this method, two applications are proposed. One annotates DNA sequences by hiding GenBank sequence annotations to make sequences self-describable. The other is a fragile watermark scheme, in which the watermark to be embedded can be used to detect tampered area(s) in DNA sequences. Even further, if tampered area(s) takes up only little portion of the entire sequence, the original content can be exactly retrieved.

The rest of the paper is structured as follows. Section II describes the proposed reversible data hiding method. The fragile watermarking scheme is explained in Section III. Our experimental results are given in Section IV.

2. PROPOSED REVERSIBLE DATA HIDING METHOD

In this section, we will discuss our reversible data hiding method in detail. The method is called Repeated Segment Substitution (RSS) method. Before we dive into it, we will first explain the degenerate base symbols that we used to mark the DNA sequence. “Degenerate base symbols in biochemistry are an IUPAC (IUPAC) representation for a position on a DNA sequence that can have multiple possible alternatives”(Degenerate). When doing gene sequencing, not all bases in the sequence can

Table 1. Degenerate base symbols

Symbol	Bases Represented	Symbol	Bases Represented
A	A	K	G+T
C	C	R	A+G
G	G	Y	C+T
T	T	B	C+G+T
U	U	D	A+G+T
W	A+T	H	A+C+T
S	C+G	V	A+C+G
M	A+C	N	A+C+G+T

be detected. Usually, bases on some position cannot be determined. For example, the base in one position could either be A or C, but we cannot make sure which one is correct. So a degenerate base is used in this position. All symbols of IUPAC system and bases they represent are listed in Table 1.

Based on degenerate bases, base adding (+) and subtracting (−) operations are defined. For adding, every operand and result is the same as Table 1 showing except for N, which is the result of a three operands adding operation that two operands are the same bases. Subtracting is an inverse operation of adding so that we won't give it out here. In the following discussion, the result of adding is denoted as mixed-base and correspondingly, the result is mixed by two or three bases.

2.1. Data Embedding Method

In natural DNA sequences, bases usually appear repeatedly. The following is a piece of sequence we take from AC002126.1: *TTTTTTGTTT-GGGTTTTTTGTTTTGTTTTGTTTCTTTG TTTTTGAGACGGAGTGTTGCTCTTTTT*. The bolded characters are those repeated bases. This redundancy can be used to hide data. When we know the positions of those repeated-base segments, we can use the first base to represent the segment and substitute the following bases by our data. So in our method, a DNA sequence is divided into embeddable segments and non-embeddable segments according to repeated base. A segment is embeddable only when the length of the repeated bases is longer than 1, like *TT* or *AAAA*.

2.1.1. Segment Label Construction

In order to extract the data in all embeddable segments, the starting and ending position of those segments must be known to the client. So for each embeddable segment, we will construct two labels to mark the positions using the adding operation defined above. Denote the numbers of segments as N and all segments as Seg_i, $i = 1, 2,..., N$. While constructing the starting and ending labels of Seg_i, generally four situations will come:

a. Seg_{i+1} is nonembeddable: Just add the repeated base of Seg_i to the beginning of Seg_{i+1} to set the ending label of Seg_i. For example *CCCAT* where $Seg_i = CCC$ and $Seg_{i+1} = A$, add C to A and then we get *CCCMT*. The ending label is *M*.
b. Seg_{i-1} *is nonembeddable:* Just add the repeated base of Seg_{i-1} to the beginning of Seg_i to set the beginning label of Seg_i. For example *TACCC* where $Seg_{i-1} = A$ and $Seg_i = CCC$, add A to C and then we get *TAMCC*. The beginning label is *M*.
c. Seg_{i+1} *is embeddable:* In this case, the ending label of Seg_i and the beginning label of Seg_{i+1} overlaps. In order to distinguish this with situation a), we use three operands

adding to add the repeated base of Seg_{i-1} and Seg_i to the beginning of Seg_{i+1}. For example *ATTCCC* where $Seg_{i-1} = A$, $Seg_i = TT$ and $Seg_{i+1} = CCC$. Add *A* and *T* to *C* and then we get *ATTHCC*. The ending label is *H*.

d. *Seg_{i-1} is embeddable:* This situation equals to situation c) if we see Seg_{i-1} as Seg_i.

Using this label construction method, the client will know the beginning and ending position of each embeddable segment and bases between labels can be substituted to embed data.

2.1.2. Base Substitution and Data Embedding

To embed 2 bits per repeated base, we use the base substitution method that is first proposed in (Guo, 2012). This method utilized a complementary rule to establish a one-to-one mapping. That is, each base x is assigned with a complement base, denoted as $C(x)$. An example of such complementary rule is like: $(A\ T)(T\ C)(C\ G)(G\ A)$. All available rules must obey the property that for each DNA base x, x, $C(x)$, $C(C(x))$ and $C(C(C(x)))$ are not the same. With this property, a one-to-one mapping is established between 2 bits data and a repeated base.

For example, let's say that the complementary rule is as listed above and the repeated base is *A*. If the 2 bits data is 00, then *A* stay unchanged. If the data is 01, 10 or 11, then *A* is respectively substituted by $C(A) = T$, $C(C(A)) = C$ and $C(C(C(A))) = G$.

2.1.3. Location Map

Although rare, some DNA sequences in databases have already contains degenerate bases. In this situation, a location map *LM* must be used to record these bases. *LM* is a bit sequence whose length equals to the length of DNA sequence. Each bit of *LM* indicates whether the base in the corresponding position of DNA sequence is a degenerate base. If so, set the bit to 1 and otherwise 0. Since degenerate bases in the original DNA sequence are rare, *LM* can be lossless-compressed with high compression ratio. The compressed bitstream is denoted as *L*. An end of message symbol is at the end of *L*. After generating the location map, all existed degenerate bases are collected to form a recording *R* and replaced by any normal bases. In order to reconstruct the original sequence, the data to be embedded are like:

$$M = L \,||\, R \,||\, P \tag{1}$$

where *M* is the message to be embedded, *L* is the compressed location map, *R* is the recorded string of degenerate bases, *P* is the payload data, and || means concatenation.

2.1.4. Embedding Ordering

As we can see, to mark the starting and ending position of repeated segments, we introduced the degenerate base symbols. Most sequences in Genbank database do consist of degenerate bases, but these symbols only take tiny parts of sequences. So marking repeated segments with degenerate bases will add too many distortions to the sequences. Since two degenerate bases are needed to mark one repeated segments, hiding data into longer segments will cause lower distortion. So longer repeated segments have higher order of embedding. In this case, when the size of the embedding data is small, like the sequence annotations, most data are embedded into longer repeated segments. Therefore, the distortion will be lower.

The entire embedding procedure is like this: When getting payload data and a DNA sequence, we first scan the sequence to setup the Location Map L and build up an array A[n], where n represent for the longest length of the repeated segments. Denote l the length of a repeated segment, then A[l] stands for a queue that records the starting positions of all repeated segments whose lengths are l, in the order of their appearance. Then we generate the data to be embedded (that is, M); for l from n down to 2 till no data are available, embedding several data into each segment in the queue A[l]: construct the beginning and ending labels

and then embed several bits of M into bases between these labels.

2.2. Data Extraction and Reconstruction Method

The data extraction and reconstruction method is easy. When receiving a sequence, the client will need to find two mixed bases to locate the first embeddable segment. Via a two operands subtraction, the starting label will be restored. Since the base substitution method is a one-to-one mapping, extracting those embedded data can easily be done by the inverse mapping. As for the ending label, if it is mixed by two bases, we know that the following segment is nonembeddable. Thus a two operands subtraction will restore the ending label. However, if the ending label is mixed by three bases, simply restoring the ending label by subtraction will lose the beginning label of the next embeddable segment. Therefore, the beginning label of next segment should be reconstructed. Re-add the repeated base of current segment to the beginning of the next segment would accomplish that. After extracting and reconstructing the first embeddable segment, the second embeddable segment now becomes the first one. So repeating the steps above till no mixed bases are found will finish data extracting. When each segment is restored, we will know the length of that segment. Then push the extracted data into the queue of the corresponding length. When extracting is done, concatenating the data in every queue in the segment length descending order. Now we get extracted data:

$$M=L \parallel R \parallel P \quad (2)$$

As the bit stream has an end of message symbol at its end, L can be exactly separated. Decompress L we will get the location map. From location map, all positions and the number of the original degenerate bases will be known. Separate R from M via the number of degenerate bases. Restore the original sequence by replacing R to its original position. Now the DNA sequence has been exactly restored and the payload data have been extracted.

3. PROPOSED WATERMARKING METHOD

In this section we propose an algorithm for the clients to localize and reconstruction the tampered DNA bases. We use a set of random matrices to compute the reference and then the tampered bases can be reconstructed by solving the linear equation set created by the reference data and the matrices. The proposed algorithm can fully reconstruct the origin sequence when the tamper rate is not too high. When the tampering rate is too high that not all tampered bases can be retrieved, our algorithm can still retrieve majority of the tampering, which is helpful to reduce transmission overhead of the communication channel.

3.1. Watermarking Embedding Procedure

We divide the sequence into blocks and compress each block to get the reference bits. Then the original block and the reference bits are fed into a hash function to create the hash bits, after that we embed hash bits and reference bits into the DNA sequence.

3.1.1. Preprocessing Step

Before we start our work, we need to transform the DNA sequence into digital sequence, we use number {0,1,2,3} to represent base {*A*, *G*, *C*, *T*}. For example, sequence *ATGGTCA* is represented by 0311320, or 00110101111000 in binary system. In the rest of the article, we use the word "base" to represent an integer between 0 and 3.

3.1.2. Reference Data Generation

Denote the origin sequence sized as $N \times L$, where N and L are integers and L is divisible by 4. We permute and divide the sequence into N blocks each sized L by a key-dependent method. Denote

the ith block as b_i and the bases in b_i as $b_i(1)$, $b_i(2)$, …, $b_i(L)$. For each block the reference data sized $L / 4$ bases are generated in the following way:

$$\begin{bmatrix} r_i(1) \\ r_i(2) \\ \vdots \\ r_i(L/4) \end{bmatrix} = M_i \times \begin{bmatrix} b_i(1) \\ b_i(2) \\ \vdots \\ b_i(L) \end{bmatrix} \quad (3)$$

where $r_i(1)$, $r_i(2)$,… $r_i(L / 4)$ represent the $L / 4$ bases in the ith reference block and M_i is a key-dependent matrix sized $(L / 4) \times L$. Then we concatenate the all N reference blocks to create the reference data R as:

$$R = r_1 \,||\, r_2 \,||\, \ldots || \, r_n \quad (4)$$

where r_i represents the ith reference block. These reference bases can be used to reconstruct the origin sequence.

3.1.3. Hash Data Generation

We permute R and divide it into N blocks the same way as we do to the origin sequence. Denote the ith block of the reference data after permutation as r'_i. The hash data is generated in the following way:

$$h_i = hash(r'_i \,||\, a_i) \quad (5)$$

where a_i represents the ith block of the origin sequence without permutation and h_i represents the hash of the ith block which will be used in integrity authentication. Any hash function with enough security can be used.

3.1.4. Watermark Embedding:

To fit the feature of our information hiding method, we put all the information we want to embed together to make the watermark as the following way:

$$W_i = r_i \,||\, h_i \quad (6)$$

For those applications in which tamper localization is of most importance and sequence restoration is not cared, we suggest the watermark as the following way:

$$W_i = h_i \quad (7)$$

Then we convert W_i into binary and embed it into a_i with our information hiding method. We do the same thing to all the blocks and a watermarked DNA sequence is generated.

3.1.5. Key-Dependent Permutation and Matrix Generation

We have involved some key-dependent permutation and matrix generation in the procedures above. Actually, any system with enough security which is related to a secret key can be used to do the work. For example, we put the key of the client and the ID of the block into a feedback shift register(FSR), which is shared by server and client. It can produce pseudo-random numbers one by one and we can use them to do permutation and matrix generation. Any reversible permutation method can be used as long as it is based on pseudo-random numbers. For example, if we wants to do permutation to a sequence sized L, for each number X, we concatenate the first X bases to the end of the sequence:

$$X = X \bmod L \quad (8)$$

$$\begin{aligned} &a(1)\ldots a(X')\,\|\,a(X'+1)\ldots a(L) \\ &\rightarrow a(X'+1)\ldots a(L)\,\|\,a(1)\ldots a(X') \end{aligned} \quad (9)$$

where a(1) . . . a(L) represent the L bases in the group. (8) is to ensure that X is between 0

and L. For security, we repeat these two steps at least $2L$ times and the permutation is done.

In the first step, if L equals 4, then we will get a set of pseudo-random numbers between 0 and 3, which can be used in matrix generation. In reference data generation, we need a matrix sized $(\mathrm{L}/4)\times\mathrm{L}$. We can just produce $(\mathrm{L}/4)\times\mathrm{L}$ numbers to create a matrix but the rows may not be linear independent, which is useless in reconstruction. The matrix generation procedure we apply is as following:

A. Input the key and ID and create L numbers as the first row.
B. Create L numbers and check if it is linear independent of all the rows above, if not repeat this step.
C. Check if the row number is L / 4, if not go to step b).

Through these steps a matrix we need is created. If the client repeats the whole procedure he will get exactly the same matrix so it is not necessary to share the matrix online. We must state that although we can increase the probability of reconstruction by ensuring the rows linear independent, it will bring overhead create the matrix. For a randomized binary matrix sized $i \times j$, we can calculate the probability of the rows being linearly dependent through the following way:

$$p(1,j)=\frac{1}{2^j} \tag{10}$$

$$p(i+1,j)=p(i,j)+\left[1-p(i,j)\right]\times\frac{2^i}{2^j} \tag{11}$$

$$p(i,j)=1,\ if\ \ i>j \tag{12}$$

where $p(i,j)$ is the probability. We chose a matrix sized $(L/2)\times L$ to simulate the matrix generating procedure because it can show more dots that helps us investigate the trend. The figure 1 is as following:

From the figure 1 we can see that the probability decreases as L grows and when L is greater than 28 the probability is below 10^{-4}, which is a very small value. What's more, the probability of a matrix sized $(L/2)\times L$ being linearly dependent is larger than that of $(L/4)\times \mathrm{L}$. (Actually it will be $(L/4)\times 2L$ if we change it into binary and the probability will be even lower.) So we can say that it will not be much burden to create a random matrix sized $(L/4)\times \mathrm{L}$ with its rows linearly independent.

3.2. Content Reconstruction Procedure

The sequence may be tampered so not all the reference blocks are usable. We first locate the tampered blocks by checking the hash and then we use the intact reference data to recover the tamper blocks. On the receiving side, the client can check the hash to know whether the sequence is tampered and the reference data is used to reconstruct the original sequence by means of linear independent equation set.

3.2.1. Tampered Block Localization

When the client receives a sequence from the database, for every block, he extracts watermark from them. As the length of hash and watermark is public, he can separate the hash and the reference data. Then he put the DNA block and the reference block into the hash function and check if the result is the same as the hash data he received, if so, the block is marked as "reserved"; if not, the block is marked as "tampered".

The attacker cannot know which group a base belong to without the key, since the number of permutation is $(N \times L)!$, it is impossible to perform a brute force attack. Even if the attacker knows the key, what he can do is to extract the watermark and recover the origin sequence.

The attacker cannot do tamper to the sequence without being noticed by the client as far as the hash data is long enough. In fact, 32-bit hash data can ensure that the probability of a "tampered" block being marked as "reserved" is 2^{-32}, which is extremely low, so we can trust the hash data to localize the tamper. For higher security, we can add the ID of DNA he wants to download into the input of the FSR so different DNA sequences will have different permutations and matrices. If the attacker gets to know the permutation of a specific sequence by luck, it won't help when he wants to attack other sequences.

3.2.2. Content Reconstruction

After extracting the "reserved" reference data, we change the bits into bases and start reconstruction. The client can use the key to rebuild $\mathrm{b_i}$ and $\mathrm{r_i}$, but they may not be intact due to the attack. Denote the number of the intact bases in b_i as u_{b_i} and that in r_i as u_{r_i}. The missing reference bases cannot be recovered, so we use the left u_{r_i} bases to do the reconstruction. We rewrite (3) as

$$\begin{bmatrix} \mathrm{r_{i,e}}(1) \\ \mathrm{r_{i,e}}(2) \\ \vdots \\ \mathrm{r_{i,e}}(\mathrm{u_{r_i}}) \end{bmatrix} = \mathrm{M_{i,e}} \times \begin{bmatrix} \mathrm{b_i}(1) \\ \mathrm{b_i}(2) \\ \vdots \\ \mathrm{b_i}(\mathrm{L}) \end{bmatrix} \tag{13}$$

where $M_{i,e}$ is a matrix sized $u_{r_i} \times L$, whose rows are taken from M_i corresponding to the "reserved" reference data and $b_i(1)$, $b_i(2)$, …, $b_i(L)$ represent the bases in b_i.

And the right side of (13) can be rewrote as:

$$\mathrm{M_{i,e}} \times \begin{bmatrix} \mathrm{b_i}(1) \\ \mathrm{b_i}(2) \\ \vdots \\ \mathrm{b_i}(\mathrm{L}) \end{bmatrix} = \mathrm{M_{i,e,r}} \times \mathrm{C_{i,u_{b_i}}} + \mathrm{M_{i,e,t}} \times \mathrm{C_{i,L-u_{b_i}}} \tag{14}$$

where $C_{i,u_{b_i}}$ and $C_{i,L-u_{b_i}}$ represent the two column vectors consisting of the "reserved" and "tampered" bases in b_i, $M_{i,e,r}$ and $M_{i,e,t}$ are matrices with columns corresponding to bases in $C_{i,u_{b_i}}$ and $C_{i,L-u_{b_i}}$. What we want to recover is $C_{i,L-u_{b_i}}$ so we transform (13) and (14) into

$$\begin{bmatrix} \mathrm{r_{i,e}}(1) \\ \mathrm{r_{i,e}}(2) \\ \vdots \\ \mathrm{r_{i,e}}(\mathrm{u_{r_i}}) \end{bmatrix} - \mathrm{M_{i,e,r}} \times \mathrm{C_{i,u_{b_i}}} = \mathrm{M_{i,e,t}} \times \mathrm{C_{i,L\text{-}u_{b_i}}} \tag{15}$$

Equation (15) is a linear equation set with $L - u_{b_i}$ unknowns and u_{r_i} equations, since we have ensured the rows mutually linear independent, the equation set has a unique solution as long as $L - u_{b_i}$ is not bigger than u_{r_i}. Obviously the missing bases must be one solution, so we can retrieve the original block when $L - u_{b_i}$ is not bigger than u_{r_i}. We know that u_{b_i} and u_{r_i} both grow when the tampering rate grows, so if the tampered region is not too large, the intact reference data can provide enough information to recover the missing bases.

4. EXPERIMENTAL RESULTS

The proposed reversible data hiding method and the tamper localization and reconstruction algorithm for reversible watermarking was implemented in C++ and tested on a set of test sequences collected from the NCBI genbank database. For the reversible data hiding method, our test mainly focuses on the embedding capacity. That is, how many bits of data can be embedded into each base on average. As for the watermarking algorithm, our test is mainly based on two aspects: the probability of the

Table 2. The definition of some terminologies

Terminology	Definition
Capacity	The total length of data a DNA sequence can hold
Payload	The length of useful payload data
bpn	The payload data each base can hold

algorithm reconstructing all tampered bases and the percentage of tampering our algorithm can retrieve when the tampering rate is too high for it to reconstruct all tampered bases.

4.1. Capacity of Reversible Data Hiding Method

Several terminologies are defined in (Shiu, 2010), but they are not compatible to those terminologies defined in traditional image data hiding field, so we re-define these in Table 2.

In order to conduct the reversible data hiding test, about 1054 DNA sequences are downloaded and examined. The length of these sequences ranges from 5358 bases to 12849792 bases. The bpn of these sequences ranged from 0.51 bit / base to 0.67 bit / base. The average bpn is 0.588 bit / base.

For the generally used eight DNA sequences, the result is presented in Table 3.

In Table 4, we compare our method with current proposed method based on several standards. As we can see, all methods in (Shiu, 2010) have to utilize reference sequences to achieve the reversibility. So essentially they cannot be called reversible data hiding methods and should not be used to hide watermarks. Compression method in (Chang, 2007) has high embedding capacity and need no reference sequences. But the DNA sequence must be long enough to make it easy to compress. For that the DNA sequence is divided into short segments in order to localize tampered areas, the compression ratio of each segment is not as high as that of the whole sequence. So the embedding capacity of each segment is not large enough to hide the hashes and reference data. Difference Expansion method in (Chang, 2007) has low average bpn. It can be used to embed authentication information for tamper detecting but not suitable for sequence restoring.

4.2. Fully-Recovered Probability and Localization Accuracy

During watermarking test, we used DNA sequence sized 3×2^{17} bases with $L = 768$ and

Table 3. The experimental results

Locus	Number of nucleotides	Capacity (bits)	Payload (bits)	bpn (bit/base)
AC153526	200118	114896	114896	0.574
AC166252	149885	86560	86560	0.578
AC167221	204842	115212	115212	0.562
AC168874	205188	121376	121376	0.588
AC168897	195017	112376	112376	0.561
AC168901	191206	111154	111154	0.580
AC168907	193417	114182	114182	0.588
AC168908	217110	127376	127376	0.584

Table 4. Comparisons among current methods

Method	Average bpn	Reversible	Expansion(1)	Reference Sequence Needed
Compression Method	0.78	Yes	No	No
Difference Expansion Method	0.11	Yes	No	No
Insertion Method	0.58	Yes	Yes	Yes
Complementary Pair Method	0.07	Yes	Yes	Yes
Substitution Method	0.82	Yes	No	Yes
Our Method	0.59	Yes	No	No
(1): Expansion means that whether the method expands the length of the original DNA sequence.				

32-bit hash. The reason why we chose 768 as the size of a block will be discussed later. The results are shown in Figure 2 and 3:

In Figure 2, abscissa represents the attack ratio which is defined as following:

$$\alpha = \frac{N}{M} \tag{16}$$

where N represents the number of times that we do the "random attack" to the length of DNA where only one base can be tampered each time and M represents the number of base the whole DNA sequence has. Although this "based on ratio" pattern of attack is harder for us to do the statistical work, we still choose it because it can better characterize the feature of tamper attack the DNA will encounter during the transmission. And ordinate shows the fully-reconstructed probability which can be defined as following:

$$\beta = \frac{T_s}{T_w} \tag{17}$$

Figure 1. Probability of being linearly dependent of different L

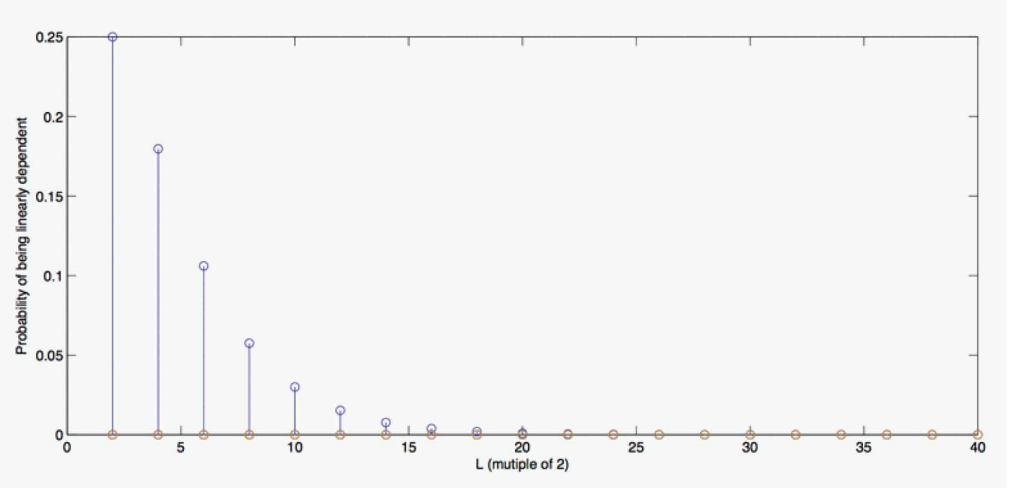
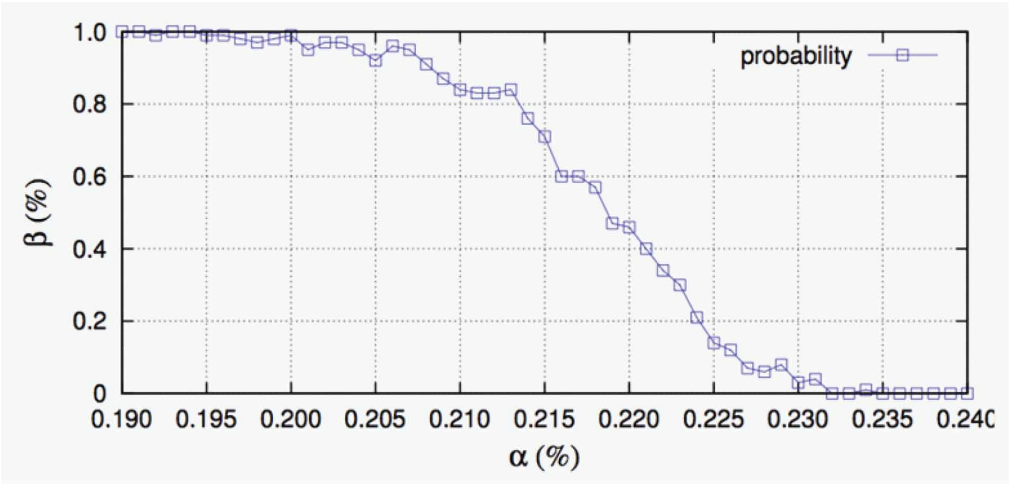


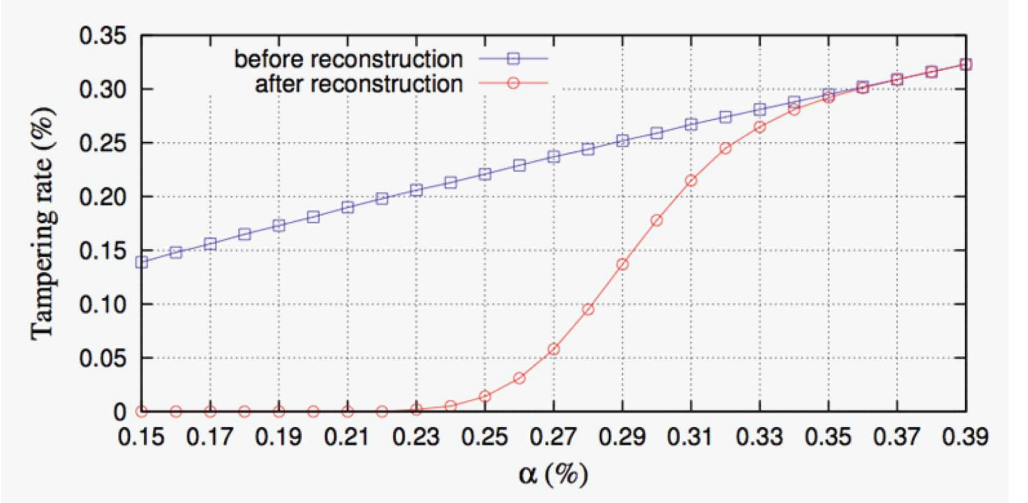
Figure 2. Probability of reconstructing all tampered bases



where T_w represents the number of times we do the test on one specific sequence and during our test T_w equals 100, T_s represents the number of times we can reconstruct the whole sequence. During the test we found that when the "attack ratio" was below 0.19, the fully-reconstructed probability was always 100% and when it rose over 0.24, the probability dropped to 0%. So we paid more attention to the interval between 0.19 and 0.24.

Before analyzing the test results, we want to discuss the block size we chose during the test. We all know that a host block must be capable to embed a reference block and a 32-bit hash. During the capacity test, the average bpn is 0.588 bit per base, so theoretically, the

Figure 3. Tampering before and after reconstruction



requirement that the host block size must meet is as follows:

$$\frac{L / 2+32}{L} \leq 0.588 \tag{18}$$

where 32 means the length of hash, L represents the block size and since we embed a reference block sized $L/4$ bases and one base must be represented by 2 bits so overall there are $L/2+32$ bits to embed. Obviously, when localizing the tampered region, smaller block size means we can do the localization more precisely so we need to make it as small as possible. After solving the inequality we found the best solution seemed to be 364, but during the test, we found that the capacity differed between blocks, even when we tried 512 as the block size we still cannot ensure that all blocks can embed the watermark, finally we chose 768 as the block size because almost all sequences can succeed the embedding procedure without one block failing to embed the watermark. In rare case, if a block cannot afford the capacity, we suggest to embed a part of the reference and a 32-bit hash, it may diminish the probability of recovery but we can make full use of the capacity through this way.

According to the figure, the probability can be kept to a high level when the attack ratio is less than 0.21. As the ratio growing, the probability drops rapidly. When the ratio is over 0.23, our algorithm can hardly recover all the tampered bases. It quite conforms to our assuming: we embed 25% of the information into the DNA sequence, according to the information theory, we can recover 25% of the sequence at best. Actually, we are not able to make the attack completely random and the pigeonhole principle tells us that when more than 25% of the sequence is tampered the proposed algorithm can never recover all the tampered bases. But when the ratio is not too extensive (below 20%), the proposed algorithm can reconstruct the whole sequence in a high probability.

When we focus only on localization accuracy we merely embed the hash data into the host sequence as (7). In this case, we found that a host block sized 128 bases is adequateto embed the 32-bit hash for most sequences. That means the accuracy of locating the position of tampering in DNA sequence is 128 bases.

5. PERCENTAGE OF TAMPER RETRIEVED

According to Figure.3, when attack ratio is below 0.33, the tampering rate after recovering is evidently lower than that before. Even though the proposed algorithm cannot recover all the bases, it can still reduce the loss of the DNA, especially when the ratio is between 0.25 and 0.30. Limited by the amount of information we embedded, the percentage decreases when the ratio grows. We can see when the ratio is over 0.35, the proposed algorithm cannot recover even one group, which means its ability of recovering is lost.

ACKNOWLEDGMENT

This work was supported in part by the Natural Science Foundation of China under Grant 61170234 and Grant 60803155, by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDA06030601, and by the Funding of Science and Technology on Information Assurance Laboratory under Grant KJ-13-003.

REFERENCES

Chang, C. C., Lu, T. C., Chang, Y. F., & Lee, R. C. T. (2007). Reversible data hiding schemes for deoxyribonucleic acid (DNA) medium. *International Journal of Innovative Computing, Information, & Control*, *3*(5), 1145–1160.

Guo, C., Chang, C. C., & Wang, Z. H. (2012). A new data hiding scheme based on DNA sequence. *International Journal of Innovative Computing, Information, & Control*, *8*, 139–149.

Naskar, R., & Rajat Subhra Chakraborty. (2013). A generalized tamper localization approach for reversible watermarking algorithms. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP). vol. 9.* NCBI Database: National Center for Biotechnology Information, from http://www.ncbi.nlm.nih.gov/

Shimanovsky, B. Jessica Feng, & Miodrag Potkonjak. (2003). Hiding data in DNA, Information Hiding (pp. 373–386).

Shiu, H. J., Ng, K. L., Fang, J. F., Lee, R. C. T., & Huang, C. H. (2010). Data hiding methods based upon DNA sequences. *Information Sciences*, *180*(11), 2196–2208. doi:10.1016/j.ins.2010.01.030

Wikipedia. (n.d.). *Degenerate base symbols: A nucleic acid notation*. Retrieved from http://en.wikipedia.org/wiki/Nucleic_acid_notation

Wikipedia. (n.d.). *IUPAC: International Union of Pure and Applied Chemistry*. Retrieved from http://en.wikipedia.org/wiki/International_Union_of_Pure_and_Applied_Chemistry