2013 6th International Conference on Biomedical Engineering and Informatics (BMEI 2013)

# Tamper Restoration on DNA Sequences Based on Reversible Data Hiding

Guoli Ma, Qi Tang, Weiming Zhang*, Nenghai Yu
School of Information Science and Technology
University of Science and Technology of China
Hefei 230027, Anhui, P.R. China
Email: {magl, tq910413}@mail.ustc.edu.cn
{zhangwm, ynh}@ustc.edu.cn

***Abstract*—Nowadays more and more DNA sequences are discovered and most of them are stored in databases of NCBI or other organizations. Once tampered, restoring the entire database usually cost much and researches based on these tampered sequences will cause disastrous consequences. This paper proposes a reversible data hiding method on DNA sequences and a fragile watermarking scheme that is based on that method. By using these methods, tampered area(s) in a DNA sequence can easily be localized. Even further, if the tampering rate is low, the original sequence can be exactly retrieved. Then by sending the retrieved sequence back, the tampered database will be restored. Our method can be applied to an online database. When the client downloads a certain DNA sequence, which may be tampered during transmission, the proposed method can help to check if the sequence is tampered and recover the tampered region.**

## I. INTRODUCTION

DNA, an abbreviation of Deoxyribonucleic Acid, is a kind of biological macromolecules that contain "genetic codes". A DNA sequence is arranged as a string of 4 nitrogenous bases: adenine (*A*), thymine (*T*), guanine (*G*), and cytosine (*C*). Each 3 bases in DNA constitute a "Codon" and via a complex process, these codons are translated into chains of amino acids, which eventually form a protein. As the blueprint of vital activities of most living things on earth and the common intellectual property of human beings, DNA has important status and must be protected perfectly. In recent days, however, most discovered DNA sequences are stored in databases of NCBI [6] and other organizations. And researchers usually download sequences from those databases for research and pharmacy. Once the sequence is tampered by evil attackers, the consequences will be disastrous. So a method that can detect the tamper or even restore the entire sequence, at any time users are downloading these sequences, is of much help.

In digital data field, researchers have established powerful reversible watermark tools to protect data, software and other intellectual properties. Using reversible data hiding method, authentication information can be embedded into these digital hosts themselves. When tampered, modified area(s) can be detected and even restored using these embedded information. If no tamper occurs, the digital hosts can be losslessly restored after extracting the embedding information.

In general, there are three types of watermark: fragile watermark, semi-fragile watermark and robust watermark. Fragile watermark fails to be detectable after modification so it is widely used for tamper detection while a semi-fragile one, which resists benign transformations, can detect malignant transformations. As its name indicates, a robust watermark can tolerate a designated transformation and therefore, copy protection applications always choose this kind of watermark.

Driven by the desire to introduce traditionally used watermarking techniques into DNA sequence protection field, some scholars proposed their data hiding and watermarking methods. In [1], Shimanovsky et al. firstly proposed a method in which the redundancy of codon to amino acid mapping is utilized. There are $4^3 = 64$ different codon combinations while the types of amino acid are only about 20. 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. Chang et al. [2] also provide two methods based on software DNA sequences, which means that a DNA sequence, regardless of its biological properties, is treated as a four-units-coded data. One of them is using lossless compressing techniques to compress the DNA sequence and append the secret data to the end of the result. Another one adopted the difference expansion way to hide data. In [3], Shiu et al. proposed three methods: the insertion method, the complementary pair method and the substitution method. 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. [4] improved the substitution method. But the reference DNA sequence is still needed.

In this paper, we propose a reversible data hiding method on DNA sequences, in which no reference DNA sequence are needed. Based on that, a fragile watermark scheme is presented, in which the watermark to be embedded can be used to detect tampered area(s) in DNA sequences and even further, if tampered area(s) takes up only little portion of the entire sequence, the original content can be exactly retrieved.

## II. PROPOSED REVERSIBLE DATA HIDING METHOD

In this section, we will discuss our reversible data hiding method in detail. This method is called Repeated Segment

978-1-4799-2761-6/13/$31.00 ©2013 IEEE

Substitution method (RSS). Before we dive into it, we need more background knowledge about the so-called degenerate base symbols (or mixed base symbols). “Degenerate base symbols in biochemistry are an IUPAC[7] representation for a position on a DNA sequence that can have multiple possible alternatives.[8]” Despite of commonly used five bases, IUPAC system also includes eleven ambiguity characters associated with every possible combination of the four DNA bases. The ambiguity characters were designed to encode positional variations found among families of related genes. All symbols of IUPAC system and bases they represent are listed in Table I.

Based on degenerate bases, we define a base adding ($+$) and subtracting ($-$) operation. For adding, every operand and result is the same as Table I. shows except for N, which is the result of a three operands adding operation that two operands are the same bases. For subtracting, it is an inverse operation of adding and we won’t give it out here. In the following discussion, we will call the result of adding as mixed-base and correspondingly, the result is mixed by two or three bases.

TABLE I
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 |

### A. Data Embedding Method

In natural DNA sequences, there exists an interesting redundancy. That is, bases usually appear repeatedly. The following is a piece of sequence we take from AC002126.1: ***TTTTTTT**G**TTTGGGTTTTTTT**G**TTTT**G**TTTT**G**TTT**C**TTT**G **TTTTT**GAGAC**GG**AGTG**TT**GCTC**TTTTT***. The bolded characters are those repeated bases. All repeated bases, except for the beginning of the repeating, can be used to embed data once the positions of those repeated-base segments are known. So when embedding, a DNA sequence is divided into embeddable segments and nonembeddable 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***.

*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 $Seg_i$, $i = 1, 2, \ldots, N$. 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 *CCC**M**T*. 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 *TA**M**CC*. 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 *ATT**H**CC*. 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) Base Substitution and Data Embedding:* To embed 2 bits per repeated base, we use the base substitution method that is first proposed in [4]. 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 $C(\boldsymbol{A}) = \boldsymbol{T}$, $C(C(\boldsymbol{A})) = \boldsymbol{C}$ and $C(C(C(\boldsymbol{A}))) = \boldsymbol{G}$.

*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:

$$M = L \mid\mid R \mid\mid P \tag{1}$$

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.

The entire embedding procedure is like this: When get payload data and a DNA sequence, first generate the data to be embedded (that is, *M*); Then for each segment between $Seg_2$ and $Seg_{N-1}$, if it is nonembeddable, do nothing and skip to the next segment; If it is embeddable, construct the beginning and ending labels and then embed several bits of *M* into bases between these labels.

### *B. 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. Now we get extracted data:

$$M = L \mid\mid R \mid\mid P \tag{2}$$

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 both 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.

## III. PROPOSED WATERMARKING METHOD

In this section we propose an algorithm for the client to localize and reconstruction the tampered DNA bases. We use a set of random matrices to compute the reference data and reconstruct the tampered bases 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 tampered region is not too large; when the tampering rate is too high that not all tampered bases can be retrieved, our algorithm can still retrieve the majority of the tamper, which can reduce transmission overhead of the communication channel.

### *A. Watermark Embedding Procedure*

We divide the sequence into blocks and do irreversible compression on 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 the hash bits and the reference bits are embedded into the DNA sequence.

*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.

*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:

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

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:

$$R = r_1 \mid\mid r_2 \mid\mid \ldots \mid\mid r_N \tag{4}$$

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

*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:

$$h_i = hash(r_i^{'} \mid\mid a_i) \tag{5}$$

where $a_i$ represents the ith block of the origin sequence without permutation, $h_i$ represents the hash of the ith block which will be used in integrity authentication, and any hash function with enough security can be used as $hash()$.

*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:

$$W_i = r_i^{'} \mid\mid h_i \tag{6}$$

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:

$$W_i = h_i \tag{7}$$

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.

*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:

$$X^{'} = X \ mod \ L \tag{8}$$

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

where $a(1)\ldots a(L)$ 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 $(L/4) \times L$. We can just produce $(L/4) \times L$ 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 goto 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 making the rows mutually linear independent, it will bring overhead creating the matrix, but when $L$ is not too big, it is not a heavy burden.

### *B. Content Reconstruction Procedure*

The sequence maybe 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.

*1) Tampered Block Localization:* When the client receives a sequence from the database, for every block, he extracts watermark from it. 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 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.

*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 $b_i$ and $r_i$, 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

$$\begin{bmatrix} r_{i,e}(1) \\ r_{i,e}(2) \\ \vdots \\ r_{i,e}(u_{r_i}) \end{bmatrix} = M_{i,e} \times \begin{bmatrix} b_i(1) \\ b_i(2) \\ \vdots \\ b_i(L) \end{bmatrix} \tag{10}$$

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 (10) can be rewrote as

$$M_{i,e} \times \begin{bmatrix} b_i(1) \\ b_i(2) \\ \vdots \\ b_i(L) \end{bmatrix} = M_{i,e,r} \times C_{i,u_{b_i}} + M_{i,e,t} \times C_{i,L-u_{b_i}} \tag{11}$$

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 (10) and (11) into

$$\begin{bmatrix} r_{i,e}(1) \\ r_{i,e}(2) \\ \vdots \\ r_{i,e}(u_{r_i}) \end{bmatrix} - M_{i,e,r} \times C_{i,u_{b_i}} = M_{i,e,t} \times C_{i,L-u_{b_i}} \tag{12}$$

(12) 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.

## IV. 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 [6]. 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: 1) the probability of the algorithm reconstructing all tampered bases and localization accuracy; 2) the percentage of tamper our algorithm can retrieve when the tampering rate is too high for it to reconstruct all tampered bases.

### *A. Capacity of Reversible Data Hiding Method*

Several terminologies are defined in [3], but they are not compatible to those terminologies defined in traditional image data hiding field, so we re-define these in Table II.

TABLE II
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 |

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 III.

TABLE III
THE EXPERIMENTAL RESULT

| **Locus** | **Number of nucleotides** | **Capacity (bits)** | **Payload (bits)** | **bpn (bits/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 |

In Table IV, we compare our method with current proposed method based on several standards. As we can see, all methods in [3] 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 [2] 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 [2] has low average bpn. It can be used to embed authentication information for tamper detecting but not suitable for sequence restoring.

TABLE IV
COMPARASONS AMONG CURRENT METHODS

| **Method** | **Average bpn** | **Reversible** | **Expansion$^{(1)}$** | **Reference Sequence Needed** |
|---|---|---|---|---|
| Compression Method in [2] | 0.78 | Yes | No | No |
| Difference Expansion Method in [2] | 0.11 | Yes | No | No |
| Insertion Method in [3] | 0.58 | Yes | Yes | Yes |
| Complementary Pair Method in [3] | 0.07 | Yes | Yes | Yes |
| Substitution Method in [3] | 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.

### *B. Fully-recovered Probability and Localization Accuracy*

During watermarking test, we used DNA sequence sized $3 \times 2^{17}$ bases with $L = 768$ and 32-bit hash. The reason why we chose 768 as the size of a block will be discussed later. The results are shown in Fig.1 and Fig.2:

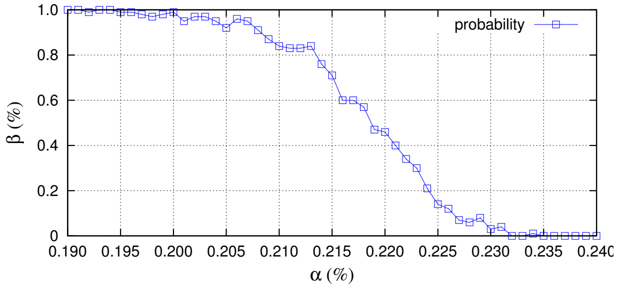

Fig. 1. Probability of reconstructing all tampered bases

In Fig.1, abscissa represents the attack ratio which is defined as following:

$$\alpha = \frac{N}{M} \tag{13}$$

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:

$$\beta = \frac{T_s}{T_w} \tag{14}$$

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

$$\frac{L \;/\; 2 + 32}{L} \leq 0.588 \tag{15}$$

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 adequate to embed the 32-bit hash for most sequences. That means the accuracy of locating the position of tampering in DNA sequence is 128 bases.

### C. Percentage of Tamper Retrieved

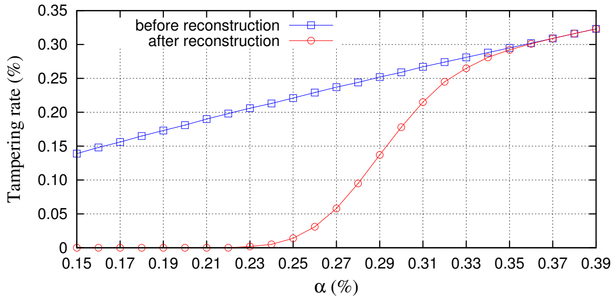

Fig. 2. Tampering rate before and after reconstruction

According to the figure, 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

[1] Shimanovsky, Boris, Jessica Feng, and Miodrag Potkonjak. "Hiding data in DNA" *Information Hiding*. pp. 373–386, January 2003.

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

[3] 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, June 2010.

[4] C. Guo, C. C. Chang and Z. H. Wang. "A new data hiding scheme based on DNA sequence" *International Journal of Innovative Computing, Information and Control*. vol. 8, pp 139–149, January 2012.

[5] Naskar, Ruchira, and Rajat Subhra Chakraborty. "A generalized tamper localization approach for reversible watermarking algorithms" *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*. vol. 9, June 2013.

[6] NCBI Database: National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov/

[7] IUPAC: International Union of Pure and Applied Chemistry, http://en.wikipedia.org/wiki/International_Union_of_Pure_and_Applied_Chemistry

[8] Degenerate base symbols: A Nucleic Acid Notation, http://en.wikipedia.org/wiki/Nucleic_acid_notation