Journal of the Egyptian Mathematical Society

ORIGINAL RESEARCH Open Access

# Secure Hash Algorithm-2 formed on DNA

Dieaa I. Nassr

Correspondence:
dieaa.nassr@sci.asu.edu.eg;
diaa.rsa@gmail.com
Computer Science Division,
Department of Mathematics,
Faculty of Science, Ain Shams
University, Cairo, Egypt

**Abstract**
We present a new version of the Secure Hash Algorithm-2 (SHA-2) formed on artificial sequences of deoxyribonucleic acid (DNA). This article is the first attempt to present the implementation of SHA-2 using DNA data processing. We called the new version DNSHA-2. We present new operations on an artificial DNA sequence, such as (1) $\bar{R}^k(\alpha)$ and $\bar{L}^k(\alpha)$ to mimic the right and left shift by $k$ bits, respectively; (2) $\bar{S}^k(\alpha)$ to mimic the right rotation by $k$ bits; and (3) DNA-nucleotide addition (mod $2^{64}$) to mimic word-wise addition (mod $2^{64}$). We also show, in particular, how to carry out the different steps of SHA-512 on an artificial DNA sequence. At the same time, the proposed nucleotide operations can be used to mimic any hash algorithm of its bitwise operations similar to bitwise operations specified in SHA-2. The proposed hash has the following features: (1) it can be applied to all data, such as text, video, and image; (2) it has the same security level of SHA-2; and (3) it can be performed in a biological environment or on DNA computers.

**Keywords:** Secure hash function, SHA-2, DNA

**Mathematics Subject Classification (2000):** 68P25, 94A60, 92D20

## Introduction

A hash function is a function that maps a binary data of arbitrary size to a fixed-size string. For input data (often called message), the output of the hash function is called the hash value or digest of the message. Several applications use hash functions in hash tables to reduce the time cost for finding a data record given its search key. Typically, the domain size of a hash function is greater than its range. Therefore, there must be different massages (inputs) producing the same digest (output), and this is called a collision case. A hash function adapted to cryptographic applications has certain properties, including its resistance to collision, pre-image and second pre-image attacks [1–4], and to be a one-way function (infeasible to reverse). In this case, the hash function is called a secure hash function and it is used for providing message authentication, data integrity, password verification, and many other information security applications [5].

Secure Hash Algorithm-2 (SHA-2) is a set of secure hash functions standardized by NIST as part of the Secure Hash Standard in FIPS 180-4 [6]. Although there is a new version of the standard called SHA-3 [7], NIST does not currently intend to remove SHA-2 from the revised Secure Hash Standard as no significant attack on SHA-2 has been demonstrated. Rather, SHA-3 can be used in the information security applications that need to improve the robustness of NIST's overall hash algorithm toolkit. There are six hash functions belonging to SHA-2, and these hash functions have names corresponding to their digest length: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256.

These hash functions have very similar structures unlike only in the number of rounds, additive constants, shift amounts, and digest size.

The aim of this paper is to introduce a new version of SHA-2 in DNA model considering the security properties of SHA-2. To the best of our knowledge, there is no article that discusses the implementation of SHA-2 using DNA data processing. We are therefore interested in studying how to implement SHA-2 on the DNA environment. Since the hash functions belonging to SHA-2 have almost the same basic processes, we focus on the construction of SHA-512 to be processed in a DNA environment (DNSHA-512) and the other hash functions are similar. The construction of DNSHA-512 contains new imitation of the operations:

1. Right (and left) shift by $k$ bits
2. Right rotation by $k$ bits
3. Addition modulo $2^{64}$

In Table 1, we give the list of abbreviations used in this paper.

The paper is organized as follows. In the “DNA” section, we present some basic background of DNA required in this paper. A brief explanation of SHA-512 is given in the SHA-512” section. In the “DNSHA-2” section, we give the nucleotide operations that mimic the bitwise operations used in SHA-2 and the algorithm of DNSHA-512 of the proposed implementation of SHA-512 on an artificial DNA sequence. The “Implementation” section contains the implementation of DNSHA-512. In the “Conclusion” section, we include the conclusion.

**Table 1** List of abbreviations

| | |
|---|---|
| SHA-2 | Secure Hash Algorithm-2 |
| DNA | Deoxyribonucleic acid |
| $A$ | The nitrogenous base (adenine) |
| $C$ | The nitrogenous base (cytosine) |
| $G$ | The nitrogenous base (guanine) |
| $T$ | The nitrogenous base (thymine) |
| $(e_{n-1}\ldots e_1 e_0)_2$ | A binary string |
| $\oplus$ | Bitwise XOR |
| $\neg$ | Bitwise negation |
| $\wedge$ | Bitwise AND |
| $\vee$ | Bitwise OR |
| $+$ | Addition (mod $2^{64}$) |
| $R^k$ | Right shift by $k$ bits |
| $S^k$ | Right rotation by $k$ bits |
| $\bar{\neg}$ | The nucleotide operation to imitate the bitwise NOT |
| $\bar{\wedge}$ | The nucleotide operation to imitate the bitwise AND |
| $\bar{\vee}$ | The nucleotide operation to imitate the bitwise OR |
| $\bar{\oplus}$ | The nucleotide operation to imitate the bitwise XOR |
| $\bar{R}^k$ | The nucleotide operation to imitate the right shift by $k$ bits |
| $\bar{S}^k$ | The nucleotide operation to imitate the right rotation by $k$ bits |

## DNA

Deoxyribonucleic acid (DNA) is a huge molecule; most of them exist in the nucleus of the cells of the organism and in many viruses and contain a genetic code used during the reproduction and the evolution of these organisms. Most of the DNA molecules consist of two chains of biological polymers wrapped around a double strand. Each strand of DNA is made up of a long sequence of nucleotides. These nucleotides are for storing genetic information. They get the information needed to build proteins, DNA, or RNA. There are four types of nucleotides: adenine $A$, cytosine $C$, guanine $G$, or thymine $T$. Their names are usually abbreviated with the first letter only. A long chain (sequence) of nucleotides is written as a sequence of letters $A, C$, $G$, and $T$. This sequence (of nucleotides) forms the genetic code of cells. A sequence of nucleotides is connected together using a vertebra composed of phosphate and a sugar (deoxyribose). Nucleotides are sometimes called bases. Some results [8, 9] pointed out that it is possible to build and generate a chain of artificial nucleotides (DNA sequences) and create complex molecular machines. Because of the progress in the discovery of many properties of DNA [10, 11], there is a new data storage technique that depends on the DNA molecule. Several methods have been given in [12–19] for storing data in DNA sequences in which 1 g of DNA can be used to store about $10^6$ TB of data; thus, a small number of grams of DNA is enough to store all the data of our world for hundreds of years. Many results [20–24] have developed a new data processing in DNA environment known as DNA computing. Adelman [20] has shown that by biochemical DNA operations, molecules could be used to carry out the computation. This author exploited the biochemical operations of DNA to obtain a solution for the Hamiltonian path problem. Computations are carried out in efficient parallel operations. Additionally, Lipton [24] has offered an encoding schema, exploiting operations of DNA molecules, to obtain a solution for the satisfiability problem with a small number of variables. A generalization of Lipton's schema has been given in [22]. Boneh et. al. [25] has shown that the data encryption standard (DES) could be broken by using the concept of DNA computation. He has presented a molecular program to break DES. Now, the study of the features of DNA has several objectives not only in the gene sequences but also in carrying out computations and in the field of data protection, where a private data can be written in a secret location in a DNA molecule to protect this data for a long time from unauthorized persons [26–30].

In the literatures [12–17], encoding data in DNA sequence has been classified by two ways [18, 19]:

1. The binary data is transformed to a DNA sequence. For example [31–33], the binary digits "00," "01," "10," and "11" are transformed into the nucleotides $A, C, G$, and $T$, respectively.
2. Each specified number of bits, e.g., byte, is converted into a fixed number of nucleotides using a given encoding table, see [34].

## SHA-512

This section gives a brief description of the hash algorithm SHA-512 [6]. It is an iterated hash function that pads and parses the input message into $n$ 1024-bit message blocks $M^{(j)}$ and gets the output hash value of size 512 bits. The 512-bit hash value is generally computed, using a compression function $f$ :

$$H^{(0)} = IV, \text{IV is an initial hash value (512-bit block)}$$
$$H^{(j)} = f(H^{(j-1)}, M^{(j)}) \text{ for } 1 \le j \le n.$$

The final 512-bit block $H^n$ is the hash value.

The hash function SHA-512 is described in Algorithm 1. We use the notation in Table 1, where all operators perform on 64-bit words.

The initial hash value $H^{(0)}$ is given in Table 2. We parse $H^{(0)}$ into eight 64-bit blocks $H_1^{(0)}, H_2^{(0)}, \ldots H_8^{(0)}$. The first 64 bits of $H^{(0)}$ are denoted $H_1^{(0)}$, the next 64 bits are $H_2^{(0)}$, and so on up to $H_8^{(0)}$.

Suppose that the input message is of $m$ bits. The input message is prepared as follows:

1. The input message $M$ is padded in the usual method: add the bit "1" to the end of $M$, and after that add $k$ zero bits, where $k$ is the minimal solution (non-negative) to the equation $m + 1 + k \equiv 896 \pmod{1024}$. Next, to this addition, append 128-bit block that represents the number $m$ written in binary. For example, the binary data of the message "BOB" are "01000010 01001111 01000010." This data has 24 bits. By joining the bit "1" to the end of this message, we get "01000010 01001111 01000010 1." Solving the equation $24 + 1 + k \equiv 896 \pmod{1024}$, we have $k = 871$. Therefore, preparing the message, we get:

$$010000100100111101000010\ 1\ \underbrace{00\ldots0}_{\text{871 zeros}}\ \underbrace{000\ldots11000}_{\text{24 is written in binary (128-bit)}}.$$

2. The number of bits of the padded message becomes a multiple of 1024. Therefore, the padded message is parsed into $n$ 1024-bit blocks' $M^{(1)}, M^{(2)}, \ldots, M^{(n)}$. The block $i$ is parsed into 16 words, where each word has 64 bits. The words of block $i$ are given by $M_0^{(i)}, M_1^{(i)}, \ldots M_{15}^{(i)}$. Note that the first 64 bits of block $i$ is stored in the word $M_0^{(i)}$, where the leftmost bit is stored in the most significant bit position. By the same way, the word $M_1^{(i)}$ is the second 64 bits, and so on up to $M_{15}^{(i)}$. For example, the message "BOB" after padding is one 1024-bit block, and the words $M_j^{(1)}, j = 0, 1, \ldots, 15$ are given as:

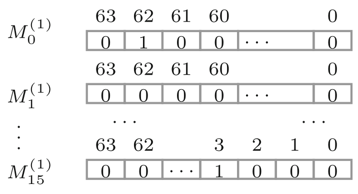

**Table 2** The initial hash $H^{(0)}$

| | |
|---|---|
| $H_1^{(0)}$ | 6a09e667f3bcc908 |
| $H_2^{(0)}$ | bb67ae8584caa73b |
| $H_3^{(0)}$ | 3c6ef372fe94f82b |
| $H_4^{(0)}$ | a54ff53a5f1d36f1 |
| $H_5^{(0)}$ | 510e527fade682d1 |
| $H_6^{(0)}$ | 9b05688c2b3e6c1f |
| $H_7^{(0)}$ | 1f83d9abfb41bd6b |
| $H_8^{(0)}$ | 5be0cd19137e2179 |

**Algorithm 1 SHA-512**

**Input:** $n$ 1024-bit blocks' $M = M^{(1)}, M^{(2)}, \ldots, M^{(n)}$.
**Output:** $H = H_1^{(n)}, H_2^{(n)}, \ldots, H_8^{(n)}$ is the hash of $M$.
**Begin**

```
1:  for i=1 to n do        ▷ n= number of 1024-bit blocks in the padded message
2:      $r_1 = H_1^{(i-1)}$
3:      $r_2 = H_2^{(i-1)}$
4:      $r_3 = H_3^{(i-1)}$
5:      $r_4 = H_4^{(i-1)}$
6:      $r_5 = H_5^{(i-1)}$
7:      $r_6 = H_6^{(i-1)}$
8:      $r_7 = H_7^{(i-1)}$
9:      $r_8 = H_8^{(i-1)}$
10:     for j=0 to 79 do        ▷ the SHA-512 compression function
11:         $C = CH(r_5, r_6, r_7)$        ▷ Eq. 1
12:         $U = MAJ(r_1, r_2, r_3)$        ▷ Eq. 2
13:         $S_0 = \Sigma_0(r_1)$        ▷ Eq. 3
14:         $S_1 = \Sigma_1(r_5)$        ▷ Eq. 4
15:         use Algorithm 2 to compute $W_j$
16:         $T_1 = h + S_1 + C + K_j + W_j$
                                    ▷ $K_0, K_1, \ldots, K_{79}$ are constant
                                    ▷ words used in SHA-512 [6].
17:         $T_2 = S_0 + U$
18:         $r_8 = r_7$
19:         $r_7 = r_6$
20:         $r_6 = r_5$
21:         $r_5 = r_4 + T_1$
22:         $r_4 = r_3$
23:         $r_3 = r_2$
24:         $r_2 = r_1$
25:         $r_1 = T_1 + T_2$
26:     end for
                                    ▷ Compute the $i^{th}$ intermediate hash value
27:     $H_1^{(i)} = r_1 + H_1^{(i-1)}$
28:     $H_2^{(i)} = r_2 + H_2^{(i-1)}$
29:     $H_3^{(i)} = r_3 + H_3^{(i-1)}$
30:     $H_4^{(i)} = r_4 + H_4^{(i-1)}$
31:     $H_5^{(i)} = r_5 + H_5^{(i-1)}$
32:     $H_6^{(i)} = r_6 + H_6^{(i-1)}$
33:     $H_7^{(i)} = r_7 + H_7^{(i-1)}$
34:     $H_8^{(i)} = r_8 + H_8^{(i-1)}$
35: end for
36: $H = H_1^{(n)}, H_2^{(n)}, \ldots, H_8^{(n)}$ is the hash of $M$
```

**End**

The algorithm of SHA-512 is given in Algorithm 1. Now, we define the logical function used in Algorithm 1:

$$CH(r_1, r_2, r_3) = (r_1 \wedge r_2) \oplus (\neg r_1 \wedge r_3) \tag{1}$$

$$MAJ(r_1, r_2, r_3) = (r_1 \wedge r_2) \oplus (r_1 \wedge r_3) \oplus (r_2 \wedge r_3) \tag{2}$$

$$\Sigma_0(r_1) = S^{28}(r_1) \oplus S^{34}(r_1) \oplus S^{39}(r_1) \tag{3}$$

$$\Sigma_0(r_1) = S^{14}(r_1) \oplus S^{18}(r_1) \oplus S^{41}(r_1) \tag{4}$$

The following algorithm, is to compute $W_j$.

**Algorithm 2 Compute $W_j$**

```
Input: one 1024-bit block $M^{(i)}$.
Output: $W_j$ 64-bit block.
Begin
1: if $0 \leq j \leq 15$ then
2:     $W_j = M_j^{(i)}$
3: else
4:     $\sigma_0 = S^1(W_{j-15}) \oplus S^8(W_{j-15}) \oplus R^7(W_{j-15})$
5:     $\sigma_1 = S^{19}(W_{j-2}) \oplus S^{61}(W_{j-2}) \oplus R^6(W_{j-2})$.
6:     $W_j = \sigma_1 + W_{j-7} + \sigma_0 + W_{j-16}$
7: end if
End
```

## DNSHA-2

In this section, we propose modern operations on nucleotides that mimic the bitwise operations used in SHA-2 and can therefore be used to mimic all members of SHA-2, i.e., to give a new version of SHA-2 called DNSHA-2. This section contains seven subsections. In the "DNA coding" section, we give how to represent data in artificial DNA sequences. In the "Basic DNA-nucleotide operations" section, we present the nucleotide operations that mimic the bitwise operations (NOT, AND, OR, XOR). In the "DNA right and left shift" and "DNA right rotation" sections, we show how to implement the nucleotide operations $\bar{R}^k$, $\bar{L}^k$, and $\bar{S}^k$ which mimic the bitwise operations (shown in Table 1), $R^k$, $L^k$, and $S^k$, respectively. The nucleotide operation that mimic the word-wise addition $\pmod{2^{64}}$ is given in the "DNA-nucleotide addition (mod $2^{64}$)" section. In the "DNA initialization and preprocessing" section, we show how initialization and preprocessing operations, especially in SHA-512, are imitated in DNA computing. In the following, sometimes, we refer to any choice of the nucleotide bases ($A, C, G$, or $T$) by the symbols $x_i$, $y_i$, and $z_i$ (or $x_i'$, $y_i'$, or $z_i'$).

### DNA coding

In classical computing, data is stored in the binary form (sequence of bytes). There are results [31–33] which encode the binary data in a DNA sequence, where the two binary digits “00,” “01,” “10,” and “11” are transformed into the nucleotides $A, C, G$, and $T$, respectively. For example, the binary string “01001110” is transformed into the nucleotides “CATG.”

We conclude this by defining the transformation $\lambda$ :

$$\lambda(e_{i+1}e_i) = \begin{cases} A, & \text{if } e_{i+1}e_i = 00; \\ C, & \text{if } e_{i+1}e_i = 01; \\ G, & \text{if } e_{i+1}e_i = 10; \\ T, & \text{if } e_{i+1}e_i = 11. \end{cases}$$

Algorithm 3 describes the representation of a data in an artificial DNA sequence. Since the byte (8-bit) is the commonly used data storage unit, we suppose in Algorithm 3 (also, in this article) that the binary data is of an even number of bits.

**Algorithm 3 DNA-encoding**

```
Input: $e = (e_{m-1}e_{m-2}\dots e_0)_2$ is a binary data, where $m$ is an even number
Output: $\alpha = x_{m/2-1}x_{m/2-2}\dots x_0$ is an artificial DNA sequence.
Begin
1: for i=0 to m/2-1 do
2:     $x_i = \lambda(e_{2i+1}e_{2i})$
3: end for
4: $\alpha = x_{m/2-1}x_{m/2-2}\dots x_0$
End.
```

We give the following example to illustrate steps of Algorithm 3.

**Example 1** *Let* $e = (100111)_2$ *be a binary data. The DNA nucleotides of* $e$ *gives the artificial DNA sequence* $\alpha = GCT$ *since:*

1. *At* $i = 0, x_0 = \lambda(11) = T$,
2. *At* $i = 1, x_1 = \lambda(01) = C$,
3. *At* $i = 2, x_2 = \lambda(10) = G$.

Algorithm 4 shows how to decode binary data from an artificial DNA sequence. Note that in the following algorithm we use $\lambda^{-1}$ to give the inverse transformation of $\lambda$.

**Algorithm 4 DNA-decoding**

```
Input: $\alpha = x_{m-1}x_{m-2}\dots x_0$ is an artificial DNA sequence.
Output: $e = (e_{2m-1}e_{2m-2}\dots e_0)_2$ a binary data that corresponds to $\alpha$
Begin
1: for i=0 to m-1 do
2:     $e_{2i+1}e_{2i} = \lambda^{-1}(x_i)$
3: end for
4: $e = (e_{2m-1}e_{2m-2}\dots e_0)_2$
End
```

We give the following example to illustrate steps of Algorithm 4.

**Example 2** *Let $\alpha = GCT$ be an artificial DNA sequence. The binary data of $\alpha$ gives $e = (100111)_2$ since:*

1. *At $i = 0, e_1e_0 = \lambda^{-1}(T) = 11$,*
2. *At $i = 1, e_3e_2 = \lambda^{-1}(C) = 01$,*
3. *At $i = 2, e_5e_4 = \lambda^{-1}(G) = 10$.*

## Basic DNA-nucleotide operations

In literatures [12–17], the nucleotide operations that imitate bitwise operations (NOT, AND, OR, XOR) are defined. The symbols $(\neg, \wedge, \vee, \oplus)$ are commonly used to express the bitwise operations (NOT, AND, OR, XOR), respectively. Throughout this paper, the symbols $(\bar{\neg}, \bar{\wedge}, \bar{\vee}, \bar{\oplus})$ are used to give the nucleotide operations that imitate the bitwise operations (NOT, AND, OR, XOR), respectively. Note that we are putting a bar sign over most of the DNA operations or above the DNA terms to differ from bitwise operations.

The nucleotide operation $\bar{\neg}$ is defined as:

$$\bar{\neg}A = T$$
$$\bar{\neg}C = G$$

In literatures [12–17], the nucleotide operations between two nucleotides $x$ and $y$ are defined as in Table 3

## DNA right and left shift

In this subsection, we propose two new operations on DNA sequence that used to mimic the right and left shift by $k$ bits. Let $\alpha = x_{m-1}x_{m-2}\ldots x_0$ be a DNA sequence and $e = (e_{2m-1}e_{2m-2}\ldots e_0)_2$ be the binary data encoded in $\alpha$. We have to mimic the operation $R^k(e)$ (right shift by $k < 2m$ bits) in SHA-2 to be $\bar{R}^k(\alpha)$ in DNSHA-2. In this regard, we take into consideration whether $k$ is an even number or odd. In case of $k$ is an even number, the operation $R^k(e)$ can be imitated in $\alpha$ by deleting $k/2$ nucleotides from right and then appending $k/2$ nucleotides $A$ from left. Therefore,

$$\bar{R}^k(\alpha) = \underbrace{AA\ldots A}_{\frac{k}{2}\,nucleotides}\ x_{m-1}\ldots x_{k/2}$$

For example, if $\alpha = TAGC$, $e = (11001001)_2$, and $k = 4$, then

$$R^4(e) = 00001100 \quad (5)$$
$$\bar{R}^4(\alpha) = AATA \quad (6)$$

**Table 3** Nucleotide operations $\bar{\wedge}$, $\bar{\vee}$, and $\bar{\oplus}$

| $x$ | $y$ | $\bar{\wedge}$ | $\bar{\vee}$ | $\bar{\oplus}$ |
|---|---|---|---|---|
| A | A | A | A | A |
| A | C | A | C | C |
| A | G | A | G | G |
| A | T | A | T | T |
| C | C | C | C | A |
| C | G | A | T | T |
| C | T | C | T | G |
| G | G | G | G | A |
| G | T | G | T | C |
| T | T | T | T | A |

In case of $k$ is an odd number, the operation $\bar{R}^k(\alpha)$ can be computed in two steps. The first step is calculating $\bar{R}^{k-1}(\alpha)$ since $k-1$ is even. The second step is calculating the right shift by one bit in DNA sequence where we denote to this operation as $RSOB(\alpha)$ and define it in Algorithm 5.

Let $\alpha = x_{m-1}x_{m-2}\ldots x_0$ be an artificial DNA sequence and $\lambda^{-1}(x_i) = e_{2i+1}e_{2i}$. Then, $RSOB(\alpha)$ is $y_{m-1}y_{m-2}\ldots y_0$, where $\lambda^{-1}(y_i) = e_{2i+2}e_{2i+1}$ for $i = 0, 1, \ldots, m-2$ and $\lambda^{-1}(y_{m-1}) = 0e_{2m-1}$. To illustrate how to perform this step, we give the following notes:

1. If $\beta$ is a DNA sequence of $m$ nucleotides $G$, then $\alpha\bar{\wedge}\beta$ yields nucleotides $z_{m-1}z_{m-2}\ldots z_0$, where $\lambda^{-1}(z_i) = e_{2i+1}0$ for $i = 0, 1, \ldots m-1$, i.e., $z_i$ is either nucleotide $A$ or $G$.
2. If $\alpha' = Ax_{m-1}x_{m-2}\ldots x_1$ and $\beta'$ is a DNA sequence of $m$ nucleotides $C$, then $\alpha'\bar{\wedge}\beta'$ yields nucleotides $Az'_{m-1}\ldots z'_1$, where $\lambda^{-1}\left(z'_i\right) = 0e_{2i}$ for $i = 1, 2, \ldots m-1$, i.e., $z'_i$ is either nucleotide $A$ or $C$.
3. Therefore, we need to define the new nucleotide operation $\bar{\boxtimes}$ as follows:
   If $\lambda^{-1}(z_i) = e_{2i+1}0$ and $\lambda^{-1}(z'_{i+1}) = 0e_{2i+2}$, then $\lambda^{-1}\left(z_i\bar{\boxtimes}z'_{i+1}\right) = e_{2i+2}e_{2i+1}$. We define this nucleotide operation in Table 4.

```
Algorithm 5 RSOB operation
Input: $\alpha = x_{m-1}x_{m-2}\ldots x_0$ is an artificial DNA sequence.
Output: $RSOB(\alpha)$
Begin
  1: define $\beta_1$ to be a DNA sequence of $m$ nucleotides $C$. i.e., $\beta_1 = CC\ldots C$
  2: define $\beta_2$ to be a DNA sequence of $m$ nucleotides $G$. i.e., $\beta_2 = GG\ldots G$
  3: $\beta_3 = Ax_{m-1}x_{m-2}\ldots x_1$
  4: $\beta_4 = \beta_1\bar{\wedge}\beta_3$     ▷ $\beta_4$ contains only two types of nucleotides $A$ and $C$
  5: $\beta_5 = \alpha\bar{\wedge}\beta_2$     ▷ $\beta_5$ contains only two types of nucleotides $A$ and $G$
  6: return $\beta_4\bar{\boxtimes}\beta_5$
End
```

The following example illustrates steps of Algorithm 5.

**Example 3** *We use the same symbols in the algorithm. Let $\alpha = TAC$ be an artificial DNA sequence encoding the binary data $e = (110001)_2$. We have $\beta_1 = CCC$, $\beta_2 = GGG$, and $\beta_3 = ATA$. Then, $\beta_4 = \beta_1\bar{\wedge}\beta_3 = ACA$ and $\beta_5 = \alpha\bar{\wedge}\beta_2 = GAA$. The result is given by $\beta_4\bar{\boxplus}\beta_5 = CGA$ encoding the binary data $(011000)_2$.*

We give the operation $\bar{R}^k(\alpha)$ in Algorithm 6. Similarly, we have to mimic the operation $L^k(e)$ (left shift by $k < 2m$ bits) in SHA-2 to be $\bar{L}^k(\alpha)$ in DNSHA-2. In case of $k$ is even,

**Table 4** The nucleotide operation $\bar{\boxtimes}$

| $x$ | $y$ | $\bar{\boxtimes}$ |
|---|---|---|
| A | A | A |
| A | C | G |
| A | G | C |
| G | C | T |

**Algorithm 6 The operation $\bar{R}^k(\alpha)$**

```
Input: $\alpha = x_{m-1}x_{m-2}\dots x_0$ is an artificial DNA sequence.
Output: $\bar{R}^k(\alpha)$
Begin
1: if $k$ is even then
2:     return $\underbrace{AA\dots A}_{\frac{k}{2}\,nucleptides}\ x_{m-1}\dots x_{k/2}$
3: else
4:     $\alpha' = \underbrace{AA\dots A}_{\frac{k-1}{2}\,nucleptides}\ x_{m-1}\dots x_{(k-1)/2}$        ▷ $\alpha' = \bar{R}^{k-1}(\alpha)$
5:     return $RSOB(\alpha')$        ▷ Algorithm 5
6: end if
End
```

the operation $L^k(e)$ can be imitated in $\alpha$ by deleting $k/2$ nucleotides from left and then appending $k/2$ nucleotides $A$ from right. Therefore,

$$\bar{L}^k(\alpha) = x_{k/2-1}\dots x_0\ \underbrace{AA\dots A}_{\frac{k}{2}\,nucleptides}$$

For example, if $\alpha = TAGC$, $e = (11001001)_2$, and $k = 4$, then

$$L^4(e) = 10010000 \tag{7}$$

$$\bar{L}^4(\alpha) = GCAA \tag{8}$$

In case of $k$ is odd, $\bar{L}^k(\alpha)$ can be computed in two steps. The first step is calculating $\bar{L}^{k-1}(\alpha)$ since $k-1$ is even. The second step is calculating the left shift by one bit in DNA sequence where we denote this operation as $LSOB(\alpha)$ and define it in Algorithm 7.

Let $\alpha = x_{m-1}x_{m-2}\dots x_0$ be an artificial DNA sequence and $\lambda^{-1}(x_i) = e_{2i+1}e_{2i}$. Then, $LSOB(\alpha)$ is $y_{m-1}y_{m-2}\dots y_0$, where $\lambda^{-1}(y_i) = e_{2i}e_{2i-1}$ for $i = 1, 2, \dots, m-1$ and $\lambda^{-1}(y_0) = e_0 0$.

**Algorithm 7 LSOB operation**

```
Input: $\alpha = x_{m-1}x_{m-2}\dots x_0$ is an artificial DNA sequence.
Output: $LSOB(\alpha)$
Begin
1: define $\beta_1$ to be a DNA sequence of $m$ nucleotides $C$. i.e., $\beta_1 = CC\dots C$
2: define $\beta_2$ to be a DNA sequence of $m$ nucleotides $G$. i.e., $\beta_2 = GG\dots G$
3: $\beta_3 = x_{m-2}\dots x_0 A$
4: $\beta_4 = \beta_2 \bar{\barwedge} \beta_3$        ▷ $\beta_4$ contains only two types of nucleotides $A$ and $G$
5: $\beta_5 = \alpha \bar{\barwedge} \beta_1$        ▷ $\beta_5$ contains only two types of nucleotides $A$ and $C$
6: return $\beta_4 \bar{\boxtimes} \beta_5$
End
```

The following example illustrates steps of Algorithm 7.

**Example 4** *We use the same symbols in the algorithm. Let* $\alpha = GTC$ *be an artificial DNA sequence encoding the binary data* $e = (101101)_2$. *We have* $\beta_1 = CCC$, $\beta_2 = GGG$, *and* $\beta_3 = TCA$. *Then,* $\beta_4 = \beta_2 \bar{\wedge} \beta_3 = GAA$ *and* $\beta_5 = \alpha \bar{\wedge} \beta_1 = ACC$. *The result is given by* $\beta_4 \bar{\boxplus} \beta_5 = CGG$ *encoding the binary data* $(011010)_2$.

We give the operation $\bar{L}^k(\alpha)$ in Algorithm 8.

**Algorithm 8 The operation $\bar{L}^k(\alpha)$**

```
Input: $\alpha = x_{m-1}x_{m-2}\ldots x_0$ is an artificial DNA sequence.
Output: $\bar{L}^k(\alpha)$
Begin
1: if $k$ is even then
2:     return $x_{k/2-1}\ldots x_0 \underbrace{AA\ldots A}_{\frac{k}{2} nucleptides}$
3: else
4:     $\alpha' = x_{(k-1)/2-1}\ldots x_0 \underbrace{AA\ldots A}_{\frac{k-1}{2} nucleptides}$        ▷ $\alpha' = \bar{L}^{k-1}(\alpha)$
5:     return $LSOB(\alpha')$        ▷ Algorithm 7
6: end if
End
```

### DNA right rotation

In this subsection, we introduce a new operation on DNA sequence that used to mimic the right rotation by $k$ bits. In Algorithm 9, we give the operation $\bar{S}^k(\alpha)$ on DNA sequence $\alpha$ to imitate the operation $S^k(e)$ (right rotation by $k$ bits), where $e$ is the binary data encoded in $\alpha$.

Let $\alpha = x_{m-1}x_{m-2}\ldots x_0$ be a DNA sequence and $e = (e_{2m-1}e_{2m-2}\ldots e_0)_2$ be the binary data encoded in $\alpha$. To compute $\bar{S}^k(\alpha)$, we first compute $\bar{R}^k(\alpha)$ using Algorithm 6 and then compute $\bar{L}^{2m-k}(\alpha)$ using Algorithm 8. Therefore, $\bar{S}^k(\alpha) = \bar{R}^k(\alpha)\bar{\vee}\bar{L}^{2m-k}(\alpha)$. The

**Algorithm 9 The operation $\bar{S}^k(\alpha)$**

```
Input: $\alpha = x_{m-1}x_{m-2}\ldots x_0$ is an artificial DNA sequence and a positive integer $k < 2m$.
Output: $\bar{S}^k(\alpha)$
Begin
1: $\beta_1 = \bar{R}^k(\alpha)$
2: $\beta_2 = \bar{L}^{2m-k}(\alpha)$
3: return $\beta_1 \bar{\vee} \beta_2$
End
```

following example illustrates steps of Algorithm 9.

**Example 5** *We use the same symbols in the algorithm. Let* $\alpha = AGT$ *be an artificial DNA sequence encoding the binary data* $e = (001011)_2$ *and* $k = 4$. *We have* $\beta_1 = \bar{R}^4(\alpha) =$

*AAA, and* $\beta_2 = \bar{L}^2(\alpha) = GTA$*. The result is given by* $\beta_1 \bar{\triangledown} \beta_2 = GTA$ *encoding the binary data* $(101100)_2$.

## DNA-nucleotide addition (mod $2^{64}$)

In this subsection, we mimic word-wise addition $\left(\text{mod } 2^{64}\right)$. We use the symbol $\boxplus$ to express nucleotide addition. In Table 5, the addition of two nucleotides $x$ and $y$ takes the form:

$$(z, \epsilon) = x \boxplus y$$

where $z$ is the addition of two nucleotides $x$ and $y$, and $\epsilon$ is called the carry nucleotide.

In Algorithm 10, we mimic the binary addition (mod $2^{64}$). Note that the binary sequence of 64 bits can be encoded in a DNA sequence of 32 nucleotides. Therefore, in Algorithm 10, we have the inputs which are two DNA sequences each of 32 nucleotides.

We use the symbol $\boxplus$ between two DNA sequences each of 32 nucleotides to express the nucleotide addition $\left(\text{mod } 2^{64}\right)$ given in Algorithm 10.

**Algorithm 10 Nucleotides Addition** (mod $2^{64}$)

**Input:** two artificial DNA sequences $x_{31}x_{30}\dots x_0$ encoding binary data $b_1$ and $y_{31}y_{30}\dots y_0$ encoding binary data $b_2$ .

**Output:** DNA sequence $z_{31}z_{30}\dots z_0$ encoding binary data $b_3$ , where $b_3 = b_1 + b_2$ (mod $2^{64}$).

**Begin**

```
1: $(z_0, \epsilon) = x_0 \boxplus y_0$
2: for i=1 to 31 do
3:     $(x, \epsilon_x) = x_i \boxplus \epsilon$
4:     $(z_i, \epsilon_y) = x \boxplus y_i$
5:     $(\epsilon, A) = \epsilon_x \boxplus \epsilon_y$
6: end for
7: return $z_{31}z_{30}\dots z_0$
```

**End**

**Table 5** Nucleotide operations $\boxplus$

| | | $\boxplus$ | |
|---|---|---|---|
| $x$ | $y$ | $z$ | $\epsilon$ |
| A | A | A | A |
| A | C | C | A |
| A | G | G | A |
| A | T | T | A |
| C | C | G | A |
| C | G | T | A |
| C | T | A | C |
| G | G | A | C |
| G | T | C | C |
| T | T | G | C |

Let

$$\alpha_1 = TCTTTTCAGTACAATTTGCATAACGTGTGGAA,$$
$$\alpha_2 = TGATAGCTATTCGATTTACTAAGCATATGTGA$$

be inputs for Algorithm 10. The following example illustrates how to compute $\alpha_1 \boxplus \alpha_2$., i.e., steps of Algorithm 10.

**Example 6** *We use the same symbols in the algorithm. We have $x_0 = A, y_0 = A, z_0 = A$, and $\epsilon = A$. Also, we have the following:*

1. $At\, i = 1, x_1 = A, x = A, \epsilon_x = A, y_1 = G, z_1 = G, \epsilon_y = A, \epsilon = A.$
2. $At\, i = 2, x_2 = G, x = G, \epsilon_x = A, y_2 = T, z_2 = C, \epsilon_y = C, \epsilon = C.$
3. $At\, i = 3, x_3 = G, x = T, \epsilon_x = A, y_3 = G, z_3 = C, \epsilon_y = C, \epsilon = C.$
4. $At\, i = 4, x_4 = T, x = A, \epsilon_x = C, y_4 = T, z_4 = T, \epsilon_y = A, \epsilon = C.$
5. $At\, i = 5, x_5 = G, x = T, \epsilon_x = A, y_5 = A, z_5 = T, \epsilon_y = A, \epsilon = A.$
6. $At\, i = 6, x_6 = T, x = T, \epsilon_x = A, y_5 = T, z_5 = G, \epsilon_y = C, \epsilon = C.$
7. $At\, i = 7, x_7 = G, x = T, \epsilon_x = A, y_7 = A, z_7 = T, \epsilon_y = A, \epsilon = A.$
8. $At\, i = 8, x_8 = C, x = C, \epsilon_x = A, y_8 = C, z_8 = G, \epsilon_y = A, \epsilon = A.$
9. $At\, i = 9, x_9 = A, x = A, \epsilon_x = A, y_9 = G, z_9 = G, \epsilon_y = A, \epsilon = A.$
10. $At\, i = 10, x_{10} = A, x = A, \epsilon_x = A, y_{10} = A, z_{10} = A, \epsilon_y = A, \epsilon = A.$
11. $At\, i = 11, x_{11} = T, x = T, \epsilon_x = A, y_{11} = A, z_{11} = T, \epsilon_y = A, \epsilon = A.$
12. $At\, i = 12, x_{12} = A, x = A, \epsilon_x = A, y_{12} = T, z_{12} = T, \epsilon_y = A, \epsilon = A.$
13. $At\, i = 13, x_{13} = C, x = C, \epsilon_x = A, y_{13} = C, z_{13} = G, \epsilon_y = A, \epsilon = A.$
14. $At\, i = 14, x_{14} = G, x = G, \epsilon_x = A, y_{14} = A, z_{14} = G, \epsilon_y = A, \epsilon = A.$
15. $At\, i = 15, x_{15} = T, x = T, \epsilon_x = A, y_{15} = T, z_{15} = G, \epsilon_y = C, \epsilon = C.$
16. $At\, i = 16, x_{16} = T, x = A, \epsilon_x = C, y_{16} = T, z_{16} = T, \epsilon_y = A, \epsilon = C.$
17. $At\, i = 17, x_{17} = T, x = A, \epsilon_x = C, y_{17} = T, z_{17} = T, \epsilon_y = A, \epsilon = C.$
18. $At\, i = 18, x_{18} = A, x = C, \epsilon_x = A, y_{18} = A, z_{18} = C, \epsilon_y = A, \epsilon = A.$
19. $At\, i = 19, x_{19} = A, x = A, \epsilon_x = A, y_{19} = G, z_{19} = G, \epsilon_y = A, \epsilon = A.$
20. $At\, i = 20, x_{20} = C, x = C, \epsilon_x = A, y_{20} = C, z_{20} = G, \epsilon_y = A, \epsilon = A.$
21. $At\, i = 21, x_{21} = A, x = A, \epsilon_x = A, y_{21} = T, z_{21} = T, \epsilon_y = A, \epsilon = A.$
22. $At\, i = 22, x_{22} = T, x = T, \epsilon_x = A, y_{22} = T, z_{22} = G, \epsilon_y = C, \epsilon = C.$
23. $At\, i = 23, x_{23} = G, x = T, \epsilon_x = A, y_{23} = A, z_{23} = T, \epsilon_y = A, \epsilon = A.$
24. $At\, i = 24, x_{24} = A, x = A, \epsilon_x = A, y_{24} = T, z_{24} = T, \epsilon_y = A, \epsilon = A.$
25. $At\, i = 25, x_{25} = C, x = C, \epsilon_x = A, y_{25} = C, z_{25} = G, \epsilon_y = A, \epsilon = A.$
26. $At\, i = 26, x_{26} = T, x = T, \epsilon_x = A, y_{26} = G, z_{26} = C, \epsilon_y = C, \epsilon = C.$
27. $At\, i = 27, x_{27} = T, x = A, \epsilon_x = C, y_{27} = A, z_{27} = A, \epsilon_y = A, \epsilon = C.$
28. $At\, i = 28, x_{28} = T, x = A, \epsilon_x = C, y_{28} = T, z_{28} = T, \epsilon_y = A, \epsilon = C.$
29. $At\, i = 29, x_{29} = T, x = A, \epsilon_x = C, y_{29} = A, z_{29} = A, \epsilon_y = A, \epsilon = C.$
30. $At\, i = 30, x_{30} = C, x = G, \epsilon_x = A, y_{30} = G, z_{30} = A, \epsilon_y = C, \epsilon = C.$
31. $At\, i = 31, x_{31} = T, x = A, \epsilon_x = C, y_{31} = T, z_{30} = T, \epsilon_y = A, \epsilon = C.$

*Thus, the result is the DNA sequence:*

$$TAATACGTTGTGGCTTGGGTTAGGTGTTCCGA.$$

**DNA initialization and preprocessing**

Since the initialization and preprocessing operations in the hash functions belonging to SHA-2 are almost similar, but differ only in initial values, we will focus on these operations for SHA-512 to be imitated in DNA computing. We give DNSHA-512 as the member of DNSHA-2 that mimics SHA-512 formed on an artificial DNA sequence.

The initial hash value $H^{(0)}$ is encoded in the DNA sequence $\bar{H}^{(0)}$ as in Table 6.

In this paper, we suppose that a binary data encoded in a DNA sequence is of an even number of bits. This is because, in the usual way, binary data are stored in some number of bytes (8-bit unit). In the following, we need to mimic the beginning computation in SHA-512 to be done similarly in DNSHA-512:

1. Pad the DNA sequence (supposed to be hashed) as follows: Suppose the length of the DNA sequence is $m$ nucleotides. We append the nucleotide $G$ to the end of the sequence, and after that $k$ nucleotides of type $A$, where $k$ is the minimal solution (non-negative) to the relation $m + 2 + k \equiv 448 \pmod{512}$. Next, to this append, we add a DNA sequence of 64 nucleotides encoded the binary data of the value of $2m$. We have the length of the padded DNA sequence which is a multiple of 512 nucleotides.
2. We parse the DNA sequence into $n$ 512-nucleotide blocks' $\bar{M}^{(1)}, \bar{M}^{(2)}, \ldots, \bar{M}^{(n)}$. The first 32 nucleotides of nucleotide block $i$ are denoted $\bar{M}_0^{(i)}$, the next 32 nucleotides are $\bar{M}_1^{(i)}$, and so on up to $\bar{M}_{15}^{(i)}$. The nucleotide block $i$ $\bar{M}^{(i)}$ (of 512 nucleotides) in DNSHA-512 has to imitate the 1024-bit block $M^{(i)}$ in SHA-512. Therefore, the 32 nucleotides of $\bar{M}_j^{(i)}$ have to be the DNA sequence that encodes $M_j^{(i)}$.

To show how to prepare the DNA sequence to be hashed, we give Example 7.

**Example 7** *The binary data of the message "BOB" are "01000010 01001111 01000010." This binary data is encoded in the DNA sequence "CAAGCATTCAAG" with* $m = 12$. *By appending the nucleotide G to the end of this sequence, we get "CAAGCATTCAAG G." Solving the equation* $12 + 2 + k \equiv 448 \pmod{512}$, *we have* $k = 434$. *Therefore, preparing the DNA sequence, we get:*

$$CAAGCATTCAAG\ G\quad \underbrace{AA\ldots A}_{434\ nucleotides}\quad \underbrace{AAAAAAAAAAAAAAAAAAAAAAAAAAAAAACGA}_{64\ nucleotides\ encode\ the\ binary\ of\ 24}$$

*The 32 nucleotides of* $\bar{M}_j^{(1)}, j = 0, 1, \ldots, 15$ *are given as:*

**Table 6** The DNA sequence $\bar{H}^{(0)}$

| | |
|---|---|
| $\bar{H}_1^{(0)}$ | = *CGGGAAGCTGCGCGCTTTATGTTATAGCAAGA* |
| $\bar{H}_2^{(0)}$ | = *GTGTCGCTGGTGGACCGACATAGGGGCTATGT* |
| $\bar{H}_3^{(0)}$ | = *ATTACGTGTTATCTAGTTTGGCCATTGAAGGT* |
| $\bar{H}_4^{(0)}$ | = *GGCCCATTTTCCATGGCCTTACTCATCGTTAC* |
| $\bar{H}_5^{(0)}$ | = *CCACAATGCCAGCTTTGGTCTGCGGAAGTCAC* |
| $\bar{H}_6^{(0)}$ | = *GCGTAACCCGGAGATAAGGTATTGCGTAACTT* |
| $\bar{H}_7^{(0)}$ | = *ACTTGAATTCGCGGGTTTGTCAACGTTCCGGT* |
| $\bar{H}_8^{(0)}$ | = *CCGTTGAATATCACGCACATCTTGAGACCTGC* |

$$
\begin{array}{l}
\begin{array}{c|c|c|c|c|c|c}
 & 31 & 30 & 29 & 28 & & 0 \\
\bar{M}_0^{(1)} & \text{C} & \text{A} & \text{A} & \text{G} & \cdots & \text{A}
\end{array} \\
\begin{array}{c|c|c|c|c|c|c}
 & 31 & 30 & 29 & 28 & & 0 \\
\bar{M}_1^{(1)} & \text{A} & \text{A} & \text{A} & \text{A} & \cdots & \text{A}
\end{array} \\
\quad\vdots \qquad \cdots \qquad\qquad \cdots \\
\begin{array}{c|c|c|c|c|c|c|c}
 & 31 & 30 & & 3 & 2 & 1 & 0 \\
\bar{M}_{15}^{(1)} & \text{A} & \text{A} & \cdots & \text{A} & \text{C} & \text{G} & \text{A}
\end{array}
\end{array}
$$

**DNSHA-512**

We give Algorithm 11 for DNSHA-512 that mimics Algorithm 1.

Now, we define functions used in Algorithm 11 (DNA functions):

$$DNACH(r_1, r_2, r_3) = (r_1 \bar{\wedge} r_2) \bar{\oplus} (\bar{\neg} r_1 \bar{\wedge} r_3) \tag{9}$$

$$DNAMAJ(r_1, r_2, r_3) = (r_1 \bar{\wedge} r_2) \bar{\oplus} (r_1 \bar{\wedge} r_3) \bar{\oplus} (r_2 \bar{\wedge} r_3) \tag{10}$$

$$\bar{\Sigma}_0(\alpha) = \bar{S}^{28}(\alpha) \bar{\oplus} \bar{S}^{34}(\alpha) \bar{\oplus} \bar{S}^{39}(\alpha) \tag{11}$$

$$\bar{\Sigma}_1(\alpha) = \bar{S}^{14}(\alpha) \bar{\oplus} \bar{S}^{18}(\alpha) \bar{\oplus} \bar{S}^{41}(\alpha) \tag{12}$$

Now, we give the algorithm needed to compute $\bar{W}_j$.

## Implementation

This section, presents an implementation of DNSHA-512. Typically, all members of SHA-2 can similarly be implemented on an artificial DNA sequence. In Table 7, we consider some metrics to evaluate DNSHA-512 compared to SHA-512.

We made a computer program that simulates each step of DNSHA-512. Then, we apply the program to hash two types of data: text and image.

The text used for the hash is "BOB." As previously stated in Example 7, the binary data for this message is encoded in the DNA sequence "CAAGCATTCAAG." After padding the DNA sequence, we get:

$$CAAGCATTCAAG\ G\ \ \underbrace{AA\ldots A}_{\text{434 nucleotides}}\ \ \underbrace{AAAAAAAAAAAAAAAAAAAAAAAAAAAAAACGA}_{\text{64 nucleotides encode the binary of 24}}$$

The hash of this message using DNSHA-512 is given by the 32 nucleotides of $\bar{H}_1^{(1)}, \bar{H}_2^{(1)}, \ldots, \bar{H}_8^{(1)}$ as follows:

**Table 7** Evaluation metrics for DNSHA-512 and SHA-512

| Metrics | SHA-512 | DNSHA-512 |
|---|---|---|
| Storage unit | Bit | DNA nucleotide |
| The input size in every iteration | 1024 bits | 512 nucleotides |
| The output size in every iteration | 512 bits | 256 nucleotides |
| Implementation on DNA computers | Not configured | Configured |

**Algorithm 11 DNSHA-512**

```
Input: n 512-nucleotide blocks' $\bar{M} = \bar{M}^{(1)}, \bar{M}^{(2)}, \ldots, \bar{M}^{(n)}$.
Output: $\bar{H} = \bar{H}_1^{(n)}, \bar{H}_2^{(n)}, \ldots, \bar{H}_8^{(n)}$ is the hash of $\bar{M}$.
Begin
 1: for i=1 to n do ▷ n= number of 512-nucleotide blocks' in the padded DNA sequence
 2:     $r_1 = \bar{H}_1^{(i-1)}$
 3:     $r_2 = \bar{H}_2^{(i-1)}$
 4:     $r_3 = \bar{H}_3^{(i-1)}$
 5:     $r_4 = \bar{H}_4^{(i-1)}$
 6:     $r_5 = \bar{H}_5^{(i-1)}$
 7:     $r_6 = \bar{H}_6^{(i-1)}$
 8:     $r_7 = \bar{H}_7^{(i-1)}$
 9:     $r_8 = \bar{H}_8^{(i-1)}$
10:     for j=0 to 79 do                    ▷ Mimic the SHA-512 compression function
11:         $C = DNACH(r_5, r_6, r_7)$                                    ▷ Eq. 9
12:         $U = DNAMAJ(r_1, r_2, r_3)$                                   ▷ Eq. 10
13:         $S_0 = \bar{\Sigma}_0(r_1)$                                          ▷ Eq. 11
14:         $S_1 = \bar{\Sigma}_1(r_5)$                                          ▷ Eq. 12
15:         use Algorithm 1 to compute $\bar{W}_j$
16:         $T_1 = r_8 \boxplus S_1 \boxplus C \boxplus K_j \boxplus \bar{W}_j$
                        ▷ $K_0, K_1, \ldots, K_{79}$ are constant words used in SHA-512 [6]
                        ▷ These constant words considered here to be encoded in
                                                        ▷ constant DNA sequences
17:         $T_2 = S_0 \boxplus U$
18:         $r_8 = r_7$
19:         $r_7 = r_6$
20:         $r_6 = r_5$
21:         $r_5 = r_4 \boxplus T_1$
22:         $r_4 = r_3$
23:         $r_3 = r_2$
24:         $r_2 = r_1$
25:         $r_1 = T_1 \boxplus T_2$
26:     end for
                            ▷ Compute the $i^{th}$ intermediate hash value
27:     $\bar{H}_1^{(i)} = r_1 \boxplus \bar{H}_1^{(i-1)}$
28:     $\bar{H}_2^{(i)} = r_2 \boxplus \bar{H}_2^{(i-1)}$
29:     $\bar{H}_3^{(i)} = r_3 \boxplus \bar{H}_3^{(i-1)}$
30:     $\bar{H}_4^{(i)} = r_4 \boxplus \bar{H}_4^{(i-1)}$
31:     $\bar{H}_5^{(i)} = r_5 \boxplus \bar{H}_5^{(i-1)}$
32:     $\bar{H}_6^{(i)} = r_6 \boxplus \bar{H}_6^{(i-1)}$
33:     $\bar{H}_7^{(i)} = r_7 \boxplus \bar{H}_7^{(i-1)}$
34:     $\bar{H}_8^{(i)} = r_8 \boxplus \bar{H}_8^{(i-1)}$
35: end for
36: $\bar{H} = \bar{H}_1^{(n)}, \bar{H}_2^{(n)}, \ldots, \bar{H}_8^{(n)}$ is the hash of $\bar{M}$
End
```

```
Algorithm 12 Compute $\bar{W}_j$
Input: one 512-nucleotide block $\bar{M}^{(i)}$.
Output: $\bar{W}_j$ DNA sequence of 32 nucleotides.
Begin
1: if $0 \le j \le 15$ then
2:     $\bar{W}_j = \bar{M}_j^{(i)}$
3: else
4:     $\sigma_0 = \bar{S}^1(\bar{W}_{j-15}) \bar{\oplus} \bar{S}^8(\bar{W}_{j-15}) \bar{\oplus} \bar{R}^7(\bar{W}_{j-15})$
5:     $\sigma_1 = \bar{S}^{19}(\bar{W}_{j-2}) \bar{\oplus} \bar{S}^{61}(\bar{W}_{j-2}) \bar{\oplus} \bar{R}^6(\bar{W}_{j-2})$
6:     $\bar{W}_j = \sigma_1 \boxplus \bar{W}_{j-7} \boxplus \sigma_0 \boxplus \bar{W}_{j-16}$
7: end if
End
```

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\bar{H}_1^{(1)}$ | T | T | T | C | A | G | G | A | A | T | A | C | C | A | A | A | A | C | A | C | G | C | G | A | C | G | G | T | T | C | C | A |
| $\bar{H}_2^{(1)}$ | G | A | T | G | G | T | T | C | G | T | C | T | C | A | A | A | C | C | C | A | G | A | C | T | G | C | C | C | C | C | A | G |
| $\bar{H}_3^{(1)}$ | G | C | T | A | G | T | T | C | A | T | C | T | G | G | A | C | A | G | T | C | C | G | C | T | C | T | C | T | C | A | G | G |
| $\bar{H}_4^{(1)}$ | G | G | T | G | G | C | C | A | C | G | G | T | C | C | C | G | A | C | C | C | A | T | A | G | A | T | G | A | C | A | G | G |
| $\bar{H}_5^{(1)}$ | G | C | T | A | C | A | T | T | A | C | C | G | T | C | A | T | C | T | C | G | G | T | T | C | C | G | T | T | T | C | C | A |
| $\bar{H}_6^{(1)}$ | A | C | G | A | C | C | A | G | A | G | T | T | G | A | A | G | G | T | A | T | C | A | T | T | T | A | C | T | C | C | T | C |
| $\bar{H}_7^{(1)}$ | T | T | G | A | C | G | C | T | C | C | T | A | G | A | T | G | A | C | T | T | T | G | A | C | T | G | C | A | C | G | C | T |
| $\bar{H}_8^{(1)}$ | A | C | T | T | T | A | G | A | A | A | A | G | T | C | T | A | T | T | G | A | A | G | A | C | T | C | G | T | A | T | G | C |

The corresponding hash of this message using SHA-512 is given by 64-bit words of $H_1^{(1)}, H_2^{(1)}, \ldots, H_8^{(1)}$ as follows:

| | |
|---|---|
| $H_1^{(1)}$ | fd28314011986bd4 |
| $H_2^{(1)}$ | 8ebdb74054879552 |
| $H_2^{(1)}$ | 9cbd37a12d67774a |
| $H_4^{(1)}$ | ae946b561532384a |
| $H_5^{(1)}$ | 9c4f16d376bd6fd4 |
| $H_6^{(1)}$ | 18522f82b34fc75d |
| $H_7^{(1)}$ | f8675c8e1fe1e467 |
| $H_8^{(1)}$ | 1fc802dcf821db39 |

The image used for the hash is the lake image declared in Fig. 1.

This image has 4,200,848 bits. After padding, the binary data of this image has 4103 message blocks (1024-bit). The hash of this image using DNSHA-512 is given by the 32 nucleotides of $\bar{H}_1^{(4103)}, \bar{H}_2^{(4103)}, \ldots, \bar{H}_8^{(4103)}$ as follows:

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\bar{H}_1^{(4103)}$ | A | A | G | G | T | G | T | C | C | G | C | C | C | T | T | C | T | G | C | G | G | C | T | T | T | C | G | A | T | G | C | G |
| $\bar{H}_2^{(4103)}$ | T | C | T | A | G | T | T | C | G | T | C | A | C | C | C | C | G | G | T | T | T | A | T | G | C | C | A | C | T | G | C | T |
| $\bar{H}_3^{(4103)}$ | G | G | T | C | T | A | A | C | G | C | T | T | G | C | A | C | G | G | A | G | G | G | C | G | A | A | T | C | A | C | C | T |
| $\bar{H}_4^{(4103)}$ | G | G | T | T | G | A | A | T | C | G | C | T | G | C | G | C | T | C | C | G | A | T | G | T | C | A | C | C | A | G | G | A |
| $\bar{H}_5^{(4103)}$ | A | C | C | A | T | C | A | A | T | T | A | T | A | G | A | T | G | T | T | C | C | A | C | T | C | C | G | A | G | G | A | G |
| $\bar{H}_6^{(4103)}$ | C | A | T | A | C | A | T | T | C | A | T | G | T | C | A | T | C | A | T | C | T | G | C | G | G | C | T | C | A | T | G | C |
| $\bar{H}_7^{(4103)}$ | A | A | G | G | T | T | G | C | T | T | A | G | C | T | G | A | G | A | A | C | A | A | G | T | T | G | G | G | A | C | G | C |
| $\bar{H}_8^{(4103)}$ | A | T | C | T | C | G | A | A | A | A | G | T | A | G | T | T | G | C | G | G | T | T | A | A | C | G | A | T | G | A | G | T |

The corresponding hash of this image using SHA-512 is given by 64-bit words of $H_1^{(4103)}, H_2^{(4103)}, \ldots, H_8^{(4103)}$ as follows:

| | |
|---|---|
| $H_1^{(4103)}$ | 0aed657de69fd8e6 |
| $H_2^{(4103)}$ | dcbdb455afce51e7 |
| $H_2^{(4103)}$ | adc19f91a2a60d17 |
| $H_4^{(4103)}$ | af836799d63b4528 |
| $H_5^{(4103)}$ | 14d0f323bd4758a2 |
| $H_6^{(4103)}$ | 4c4f4ed34de69d39 |
| $H_7^{(4103)}$ | 0af9f278810bea19 |
| $H_8^{(4103)}$ | 37600b2f9af0638b |

## Conclusion

We have presented the implementation of SHA-2 using DNA data processing. To the best of our knowledge, this result is the first attempt to model a standard hash function using DNA data processing. We have shown how to encode binary data into a DNA sequence,

**Fig. 1** The image used for the hash

and we have given nucleotide operations that mimic the bitwise operations used in SHA-2. In particular, we have presented the DNA operations $\bar{R}^k(\alpha), \bar{L}^k(\alpha)$, and $\bar{S}^k(\alpha)$ that used to mimic the bitwise operations $R^k(e), L^k(e)$, and $S^k(e)$, where $e$ (binary data) is encoded in the the DNA sequence $\alpha$. Therefore, this work can be used to mimic any hash algorithm of its bitwise operations limited to bitwise operations specified in SHA-2. Similarly, the nucleotide operations proposed in this result can be exploited to lead to a preliminary result to perform SHA-3 on DNA sequences.

**Acknowledgements**
We are grateful to Hatem M. Bahig for his support, valuable comments, and remarks. Furthermore, we are thankful to the referees for their precious comments, which lead to the improvement of the paper.

**Authors' contributions**
DIN is the only author of this article, and he has performed all the analysis, verifications, and completions of the results included in this article. The author read and approved the final manuscript.

**Funding**
Not applicable

**Availability of data and materials**
Not applicable.

**Competing interests**
The author declares that he has no competing interests.

Received: 20 March 2019 Accepted: 23 August 2019
Published online: 18 September 2019

**References**

1. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for step-reduced SHA-2. In: Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings, Vol. 5912 of Lecture Notes in Computer Science, pp. 578–597. Springer, (2009). https://doi.org/10.1007/978-3-642-10366-7_34
2. Indesteege, S., Mendel, F., Preneel, B., Rechberger, C.: Collisions and other non-random properties for step-reduced SHA-256. In: Selected Areas in Cryptography, pp. 276–293. Springer, (2009). https://doi.org/10.1007/978-3-642-04159-4_18
3. Kelsey, J., Kohno, T.: Herding hash functions and the nostradamus attack. In: Advances in Cryptology - EUROCRYPT 2006, pp. 183–200. Springer, (2006). https://doi.org/10.1007/11761679_12
4. Sanadhya, S., Sarkar, P.: New collision attacks against up to 24-step SHA-2. In: Progress in Cryptology-INDOCRYPT 2008, pp. 91–103. Springer, (2008). https://doi.org/10.1007/978-3-540-89754-5_8
5. Menezes, A. J., van Oorschot, P. C., Vanstone, S. A.: Handbook of Applied Cryptography, CRC Press, Inc., USA (1996 )
6. N.I. of Standards, Technology, FIPS PUB 180-4: Secure Hash Standard, pub-NIST (2012). http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
7. N.I. of Standards, Technology, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions: FiPS PUB 202, pub-NIST (2015). https://books.google.com.eg/books?id=hCwatAEACAAJ
8. Friedman, M., Rogers, Y., Boyce-Jacino, M.: Gene pen devices for array printing, WO Patent App, No. 6235473 (2000). http://www.freepatentsonline.com/6235473.html
9. Kimoto, M., Matsunaga, K., Hirao, I. I.: DNA aptamer generation by genetic alphabet expansion SELEX (ExSELEX) using an unnatural base pair system. Springer, New York (2016)
10. Calladine, C., Drew, H., Luisi, B., Travers, A.: Understanding DNA: The Molecule and How itWorks. 3rd ed. Academic Press, Cambridge (2004)
11. Watson, J.: Molecular biology of the gene, Benjamin/Cummings (1987). https://books.google.com.eg/books?id=cM0fAQAAIAAJ
12. Atito, A., Khalifa, A., Rida, S. Z., Khalifa, A.: DNA-based data encryption and hiding using playfair and insertion techniques. J. Commun. Comput. Eng. **2**, 44–49 (2012)
13. Guo, C., Chang, C., Wang, Z.: A new data hiding scheme based on DNA sequence. Int. Innov. J. Comput. Inf. Control. **8**, 1–11 (2012)
14. Khalifa, A.: Lsbase: a key encapsulation scheme to improve hybrid crypto-systems using DNA steganography. In: 2013 8th International Conference on Computer Engineering & Systems (ICCES), pp. 105–110, (2013). https://doi.org/10.1109/icces.2013.6707182
15. Khalifa, A, Atito, A: High-capacity DNA-based steganography. In: 8th International Conference on Informatics and Systems. IEEE, (2012). BIO–76–BIO–80
16. Skariya, M., Varghese, M.: Enhanced double layer security using RSA over DNA based data encryption system. Int J Comput Sci Eng Technol. **4**, 746–750 (2013)
17. Taur, J., Lin, H., Lee, H., Tao, C.: Data hiding in DNA sequences based on table lookup substitution. Int J Innov Comput Inf Control. **8**, 6585–6598 (2012)
18. UbaidurRahmana, N. H., Balamuruganb, C., Mariappanab, R.: A novel DNA computing based encryption and decryption algorithm. Procedia Comput. Sci. **46**, 463–475 (2015)

19. UbaidurRahmana, N. H., Balamuruganb, C., Mariappanab, R.: A novel string matrix data structure for DNA encoding algorithm. Procedia Comput. Sci. **46**, 820–832 (2015)
20. Adleman, L.: Molecular computation of solutions to combinatorial problems. Science. **266**(11), 1021–1024 (1994)
21. Bahig, H. M., Nassr, D. I.: DNA-based AES with silent mutations. Arab. J. Sci. Eng. **44**, 1–15 (2018). https://doi.org/10.1007/s13369-018-3520-8
22. Boneh, D., Dunworth, C., Lipton, R., Sgall, J: On the computational power of DNA. Discret. Appl. Math. **71**(1-3), 79–94 (1996)
23. Kari, L., Seki, S., Sosík, P.: DNA Computing—Foundations and Implications. Springer, Berlin (2012)
24. Lipton, R.: Using DNA to solve np-complete problems. Science. **268**, 542–545 (1995)
25. Boneh, D., Dunworth, C., Lipton, R.: Breaking DES using a molecular computer. In: DNA Based Computers, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, April 4, 1995, pp. 37–66, (1995). https://doi.org/10.1090/dimacs/027/04
26. Abbasy, M., Manaf, A., Shahidan, M.: Data Hiding Method Based on DNA Basic Characteristics. Springer (2011). https://doi.org/https://doi.org/10.1007/978-3-642-22603-8_5
27. Abbasy, M., Nikfard, P., Ordi, A., Torkaman, M. DNA base data hiding algorithm. **1**, 183–193 (2012)
28. Gehani, A., LaBean, T., Reif, J.: DNA-based Cryptography. Springer, Berlin (2004)
29. Hamed, G., Marey, M., El-Sayed, S. S., Tolba, F.: DNA Based Steganography: Survey and Analysis for Parameters Optimization. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-21212-8_3
30. Tang, Q., Ma, G., Zhang, W., Yu, N.: Reversible data hiding for DNA sequences and its applications. Int. Digit. J. Crime For. **6**(4), 1–13 (2014)
31. Cui, G., Qin, L., Wang, Y., Zhang, X.: An encryption scheme using DNA technology. In: Third International Conference on Bio-Inspired Computing: Theories and Applications, pp. 37–42, (2008). https://doi.org/10.1109/bicta.2008.4656701
32. Sabry, M., Hashem, M., Nazmy, T., Khalifa, M. E.: Design of DNA-based advanced encryption standard (AES). In: 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS), pp. 390–397, (2015). https://doi.org/10.1109/intelcis.2015.7397250
33. Xin-she, L., Lei, Z., Yu-pu, H.: A novel generation key scheme based on DNA. In: International Conference on Computational Intelligence and Security, pp. 264–266, (2008). https://doi.org/10.1109/cis.2008.113
34. Wang, X., Zhang, Q.: DNA computing-based cryptography. In: 2009 Fourth International on Conference on Bio-Inspired Computing, pp. 1–3, (2009). https://doi.org/10.1109/bicta.2009.5338153

## Publisher's Note