Research Article

iMedPub Journals
www.imedpub.com

European Journal of Experimental Biology
ISSN 2248-9215

2017
Vol. 7 No. 3:19

DOI: 10.21767/2248-9215.100019

Improving Dependability and Precision of Data Encoding in DNA

Siguna Mueller[1], Farhad Jafari[1] and Don Roth[2]

1 Department of Mathematics, University of Wyoming, Laramie, WY, USA
2 Department of Molecular Biology and School of Energy Resources, University of Wyoming, Laramie, WY, USA

***Corresponding author:** Don Roth

RothDon@uwyo.edu

Department of Molecular Biology and School of Energy Resources, University of Wyoming, Laramie, WY, USA.

Tel: +446 224 5344

Citation: Mueller S, Jafari F, Roth D (2017) Improving Dependability and Precision of Data Encoding in DNA. Eur Exp Biol. Vol. 7 No. 3:19.

Abstract

DNA storage of information is emerging as the next-generation approach to archiving vast amounts of data. Various sophisticated approaches for data storage in DNA have been proposed. Herein we present a multistep algorithm designed to detect and/or correct errors introduced at any stage of the DNA storage process, including those during message DNA generation, and propose refinements designed to ensure authenticity and correctness of each individual encoded DNA block. In addition, the algorithm allows authentic decoding without a reference sequence or message meaning. The algorithm is designed based on principles underlying provably secure cryptographic systems. Importantly, our new algorithm compares favorably with current ones in terms of ease of implementation and message expansion. In cases where reads are error-free, our algorithm should be faster than current alignment techniques. Without knowing the original data, a certificate is generated that confirms that the obtained data are exactly the same as the original. Our algorithm has applications to DNA steganography, sequence alignment, fast identification of correct reads in next generation sequencing and to message security.

Keywords: Digital information storage in DNA; Error detection; Error correction; Next generation sequencing; Alignment algorithm

Received: March 21, 2017, **Accepted:** May 30, 2017, **Published:** June 15, 2017

Background

The potential of DNA has been realized for various information-theoretic protocols, including DNA steganography for the identification of genetically modified organisms [12,21,23] hiding of messages in DNA [9,11,16,20] and long-term data storage in DNA [3,7,8,10,14,21]. These applications require mechanisms to validate, ensure, and possibly verify message accuracy. This is particularly important when correctness of the retrieved data is not variable by a reference sequence or other means such as a comparison with a meaningful text template. DNA is a typical code and so are all the information theoretic algorithms that utilize it. However, what is seen by the receiver of the message might not be the same as what was initially sent or encoded. Thus, it is critical that processes using DNA for data storage have dependable and precise error correcting or detecting features.

Storing messages in DNA was first demonstrated in 1988 and the largest project to date encoded about 750 kilobytes of data, including text, tables, photos, and video [10]. As with any computer code, DNA coding approaches are susceptible to errors during construction of the code, storage, and read-out. Relative to these issues, algorithms to develop *in vivo* [23] and *in vitro* [7,10] based approaches to utilizing DNA as a framework for archiving large data sets have been developed.

Two of the most promising approaches for information archiving in DNA use *in vitro* algorithms and next-generation DNA synthesis and sequencing [7,10]. Both approaches [7,10] rely on the sequencing of multiple oligonucleotides (nt) per message component. In Ref. 7 they reported an average of ≈3000-fold coverage of each recovered nucleotide base. However, there were message sequences with only single coverage from amplification procedures and these contributed to process errors. The work in Goldman et al. [10] relied on the sequencing of ≈10^7 copies for each DNA string. Given that the sequencing reaction consumed ≈0.1% of the DNA of the initial library, two components were not sequenced at all and had to be specifically sequenced with manual techniques [10]. This result was suggested to be due to specific self-complementary regions that led to hybridization and sequencing failure. Although both projects report a significant average coverage, neither was sufficient for complete error protection.

To address these issues Church et al. [7] relied on multiple message copies to identify errors, and Goldman et al. [10] employed additional explicit error correcting features. However, with current approaches it is still difficult to consistently ensure appropriate coverage of every single nt that is necessary for correction of errors generated due to message mutation, especially if there is no reference for determining if the obtained reads reflect the authentic message. This is, in turn, is dependent on the length and quality of continuous sequences that provide appropriate alignment of overlapping reads.

Similar error correction issues impact next generation DNA sequencing technologies. Current alignment programs developed to handle the numerous individual reads use three main approaches (see [15] and the references therein): (i) “hashing” the read sequences and scanning through the reference sequence: (ii) “hashing" of the reference genome and (iii) merge-sorting of the reference subsequences and read sequences.

Herein, we present a multistep algorithm that increases the reliability and precision of existing approaches without negatively impacting efficiency and effectiveness. In particular, our algorithm establishes a significant guarantee of message validity, authenticity and integrity. It also can be applied for the archiving of vast data sets, in optimization of DNA sequencing algorithms, and in precisely tagging genetically modified organisms (GMOs). Our solution also specifically identifies which reads are correct prior to majority voting and prior to knowing the message meaning. Given appropriate reads this approach allows pooling of the correct sequences and facilitates rapid and accurate decoding.

Examples of errors in DNA storage protocols are summarized in **Tables 1 and 2**. Analysis of the robustness of current algorithms under single sequence coverage [7] identified significant distortions of encoded text, video, or pictures from sequence errors which may render decoding totally untranslatable. The most complete algorithm to date in terms of error correction is that of Goldman et al [10]. Analysis of [10] identified a potential problem during assembly of the decoded sequence reads where two specific regions were not recovered from any sequenced read. Repeats of this motif had a self-reverse complementary pattern and it was hypothesized that long, self-complementary DNA fragments might not be readily sequenced using the Illumina process [2] or other next generation sequencing protocols. It had previously been determined [7] that individual sequences, especially those containing large GC content or long self-complementary regions are difficult to accurately read or synthesize.

Church et al. [7] were the first to suggest that choosing bases randomly (*A* or *C* for 0 and *T* or *G* for 1) while disallowing homopolymer runs greater than three [7] may overrule any sequence properties that are detrimental to sequencing. Based upon analysis of potential sequencing failure encountered during decoding, Goldman et al. [7] suggested development of a code with no long self-complementary regions [9,10]. They suggested an additional step during encoding [10] whereby the initial message les could be pre-processed either by a one-time pad [19] or other stream cipher [17] with a standard or known key stream. This would lead to DNA segments having random properties [10], but would be difficult to implement practically. While introducing randomness can be crucial, the problem is how to do this in a way such that accurate decoding can be done without access to the reference message. A one-time pad requires a random key equal to the length of the encoded message [17] and any key stream that is used during message DNA generation needs to be available during decoding. This leads to the problem of how the random key can be made available to the decoding party. As a result, a sophisticated cryptographic scheme would be required to achieve randomness in the stream cipher further requiring a process to store and pass to the decoder a very long key. For long-term data storage these options may limit application.

True randomness is not needed to prevent detrimental structure that impact DNA synthesis and sequencing. In cryptography it is generally important to use good random-number generators but not as important as using good encryption algorithms and key management procedures [19]. In practice, therefore, crypto-graphic techniques use secure pseudo-number generators or keystreams [19]. We employ this approach in our multiscale algorithm described below.

Main Text

The key features of this algorithm can be subdivided into 5 steps. A glossary of abbreviations is given below. These features include (1) Generation of each block as a sequence of nt's that occurs at least *R*•*S*-fold and that is randomized by number 7 in STEP 1. This prevents any detrimental structure that would interfere with sequencing. (2) Randomization is only a function of the base 2 to base 4 conversion and hence, does not require a random key during decryption as it is not a stream cipher or a one-time pad. (3) Each block represents information about data as well as the address and is then further randomized. The code represents each (randomized and expanded) message block m_i exactly *R* times within each DNA segment.

$$\underbrace{m_1||m_1||\ldots||m_1}_{\Downarrow \text{Repeated } R \text{ times Encoded as nt's}},\quad \ldots\quad \underbrace{m_n||m_n\ldots||m_n}_{\Downarrow \text{Repeated } R \text{ times Encoded as nt's}}$$

$$\underbrace{\text{DNA segment } DNA_1}_{\Downarrow \text{Amplif. \& Sequ. to}},\quad \ldots\quad \underbrace{\text{DNA segment } DNA_n}_{\Downarrow \text{Amplif. \& Sequ. to}}$$

$$\text{sequ. } (DNA_1)_1,\ \ldots,\ \text{sequ. } (DNA_1)_{\tilde{S}} \quad \ldots \quad \text{sequ. } (DNA_n)_1,\ \ldots,\ \text{sequ. } (DNA_n)_{\tilde{S}}$$

Table 1: Algorithm STEPS and their Main Features.

Sequence of Steps	Main Features	Comment
STEP 1, STEP 2	Detection of substitution errors	To detect t substitution errors, choose R such that R•S ≥ 2t + 1
STEP 3, STEP 2	Improved randomization	The randomization function number 7 in STEP 1 may lead to self-complementary patterns, e.g., when encoding consecutive 1's. Simple cryptographic techniques yield a higher degree of randomness to prevent self-complementarity and other harmful patterns
STEP 3, STEP 4, STEP 2	Detection of any errors, including those during message DNA preparation	Authenticity is established via a cryptographic hash value that is appended to the message part before sequencing. This enables the choice of the correct reads (provided at least one exists, i.e., that S>0)
STEP 3, STEP 5, STEP 2	Proof of Authenticity of the individual segments	Similar to previous steps, but more efficient

Table 2: Primary potential for errors when using DNA as data storage. This table provides a comparison of type. of problems that can arise when using DNA as data storage, the previous approaches in dealing with these problems and our approach. In combining the strengths of the existing methods via coding theory with our proposed cryptographic solutions, the algorithm is simplified and made more efficient. The last step utilizes a specific optimal scheme from cryptography (Optimal Asymmetric Encryption Padding) which is probably the most efficient way to ensure correctness of the *entire* string along with adequate randomization.

Summary of potential errors in storing data in DNA with existing approaches		
Type of problem	**Previous approaches**	**Our approach**
Efficient generation de-novo of DNA according to predetermined design	Generation of short DNA segments with small potential for errors	Same as [7,10]
Individual pieces require correct identification and alignment during decoding	Segmentation into both data and addressing info	Same as [7,10]
ID and address of individual segments needs to be included in the code	Parity check to test obtained indexing identification	Included as part of robustness features of algorithm techniques
Sequences with specific properties or structure cannot be sequenced	Reverse-complementation, 25 bp of set the homopolymer rule, manual correction, randomization	Refined randomization
Errors can occur in each individual step of the Algorithm	Coding theory and ampli cation and/or sequencing of many strings	Tools derived from provably secure crypto systems
Errors that occur early cannot be identified or corrected.	Manual intervention	Proof of correctness of obtained strings and Proof of completeness of recovered data via a simple cryptographic solution

(4) The above is essentially an *R•S* repetition code. Each of the recovered ≥ *S* sequences $(DNA_i)_j$ is made up of a collection of R blocks of m_i. (5) The choice of *R* is dictated by the parameter *S* as well as the anticipated and required security. Both R and S work in concert. The larger *R* is, the more "expensive" in terms of message expansion but is more secure (**Figure 1**). In [7], there was no repetition and some sequences were only recovered once, increasing the probability of error introduction.

As in [7,10] oligonucleotide library is sequenced using next-generation sequencing technologies. The decoding scheme identifies each base of encoded information based on a majority vote of all the read bases corresponding to its position. The final decoding into the message file from the sequencing reads is obtained by exactly reversing the encoding process.

STEP 1-Encoding

The original text or data are subdivided into equal length blocks. Each block contains both the data part and the corresponding addressing information. Combining both into one block before repetition and sequencing helps to ensure proper placement during decoding.

1. Represent the original message M in binary and let *len*(M) be the length (in characters) of the binary representation of M.
2. Let l_M be the representation of *len*(M) in binary, as appropriate pre-pend with zeros to give a fixed length, generate the fixed length binary block $\tilde{M}=M||0...0||l_M$ by adding in zeros so that the length of $\tilde{M}$ is a multiple of $l/R\text{-}l_{id}$.
3. Divide $\tilde{M}$ into pieces $\tilde{M}_i$ of equal length $l/R\text{-}l_{id}$.

$$\tilde{M} = \underbrace{\tilde{M}_1}_{\text{of length } l/R-l_{id}} || \underbrace{\tilde{M}_2}_{\text{of length } l/R-l_{id}} ||...|| \underbrace{\tilde{M}_n}_{\text{of length } l/R-l_{id}} . \quad (1)$$

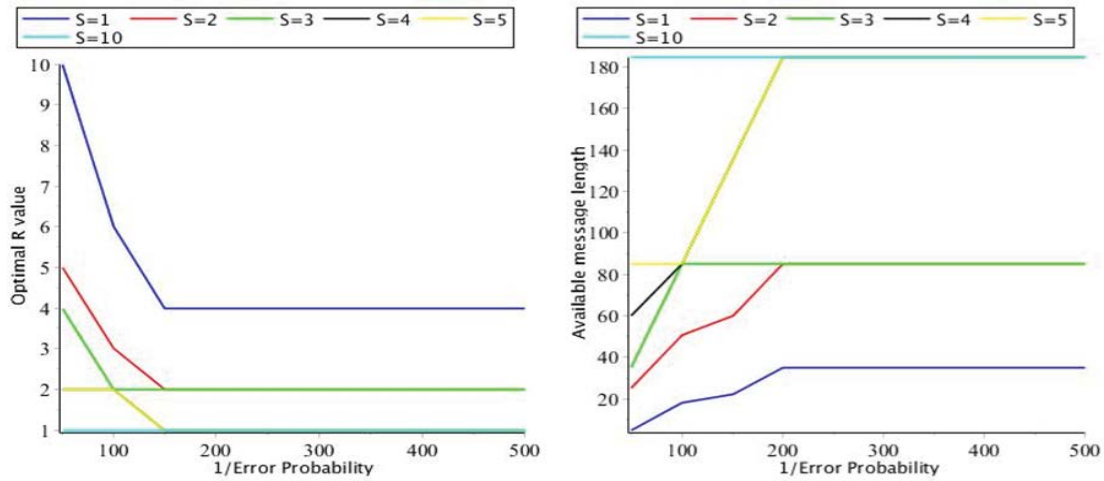

Figure 1 Error correction in our basic encoding scheme**.** The Error Probability during the entire protocol, synthesis and sequencing, versus optimal *R* value for different *S*. To correct *t* substitution errors the Hamming distance of the code, R•S must be ≥ *2t+1*. Once S is known, *R* is chosen accordingly. The figure gives examples of such choices for which all errors in synthesis and sequencing can be corrected. The optimal *R* value (left) is the *R* value that maximizes the largest available effective message length, i.e. the length of each segment $\tilde{M}_i$. The plot on the right shows the result for our basic encryption scheme, where this length is l/R-l_{id}: In these plots, l_{id}=15 and l=200.

4. This exactly incorporates all of $\tilde{M}$.

5. Let ID_{bin} be a (fixed length) binary string identifying the original file and unique with a given experiment. For each counter *i* obtained in step 2, find the binary representation of *i*. Let ID_i be the concatenation of ID_{bin} and the binary representation of *i*. If needed, prepend the latter with zeros to give the fixed length l_{id} for each ID_i.

6. Incorporate the identifier ID_i into each block of M_i to get

$$m = \underbrace{\tilde{M}_1||ID_1}_{\text{of length } l/R} \;||\; \underbrace{\tilde{M}_2||ID_2}_{\text{of length } l/R} \;||\ldots||\; \underbrace{\tilde{M}_n||ID_n}_{\text{of length } l/R} . \qquad (2)$$

7. This divides *m* into equal size blocks $m=m_1||m_2||\ldots||m_n$ where $m_i=\tilde{M}_i||ID_i$.

8. Represent each block *R* times which will give the individual DNA segments of length *l*, i.e., let

$$t = \underbrace{m_1||m_1||\ldots||m_1}_{\text{represented as } R \text{ copies}} , \underbrace{m_2||m_2||\ldots||m_2}_{\text{represented as } R \text{ copies}} , \ldots , \underbrace{m_n||m_n||\ldots||m_n}_{\text{represented as } R \text{ copies}} .$$
$$= \underbrace{t_1}_{\text{has length } l} , \underbrace{t_2}_{\text{has length } l} , \ldots , \underbrace{t_n}_{\text{has length } l}$$

9. In the basic scheme we use standard methods to achieve randomization. This is generated by the base-2 to base-4 conversion. Generally, given the binary string $s=s_1s_2\ldots s_n$, this gives one of the four values *A, C, G, T* according to A or C randomly, if s_i=0 and T or G randomly if s_i=1 for each *i*. This randomization is applied here to $t=t_1, t_2,\ldots, t_n$ but with the stronger restriction of avoiding homopolymer runs of length ≥ 2. This gives the base four oligos $f_1=F(t_1), f_2=F(t_2),\ldots, f_n=F(t_n)$ in terms of the four nts *A, C, G, T*.

STEP 1 may be improved via a 7 step process outlined below. The original DNA sequence along with the addressing information is subdivided into blocks of length *l*. Instead of amplifying and sequencing each block as it contains the base four representation of the data along with their identifiers that might lead to sequencing errors, each of the blocks are masked by some randomizing features. The key to code and decode this randomization is appended to each block. Consequently, both the original part containing the nt and the appended randomizer convert each block into arbitrary sequences with (pseudo) random properties. As random strings, they can be amplified and sequenced. Since representation in base 3 is somewhat more efficient than in base 2, base 3 representation is used, with base 3 to base 4 (i.e., DNA) representation accomplished via the algorithm [10].

1. Represent the original message M as a concatenation of base 3 strings that are obtained via the Huffman algorithm and append enough zeros to *M* to get $\tilde{M}=M0\ldots0$ so that the length of this is a multiple of l/R-l_{id}-l_r. The Huffman step adds further efficiency and

uniqueness in decoding. Without it, the length of *M*, l_M needs to be appended, as above (see the example below).

2. Split $\tilde{M}$ into pieces $\tilde{M}_i$ of equal length $l/R\text{-}l_{id}\text{-}l_r$.

$$\tilde{M} = \underbrace{\tilde{M}_1}_{\text{of length } l/R - l_{id} - l_r} || \underbrace{\tilde{M}_2}_{\text{of length } l/R - l_{id} - l_r} ||...|| \underbrace{\tilde{M}_n}_{\text{of length } l/R - l_{id} - l_r} . \quad (3)$$

By step 1 this exactly divides out all of $\tilde{M}$.

3. Let ID_{ter} be a (fixed length) base-3 string identifying the original file and unique with a given experiment. For each counter *i* obtained in step 2, let ID_i be the concatenation of ID_{ter} and the base-3 representation of *i*, the latter pre-pended with zeros as needed to give the fixed length l_{id} for each ID_i.
4. Incorporate the identifier ID_i into each block of $\tilde{M}_i$ to get the below sequence.

$$\tilde{m} = \underbrace{\tilde{M}_1||ID_1}_{\text{of length } l/R - l_r} || \underbrace{\tilde{M}_2||ID_2}_{\text{of length } l/R - l_r} ||...|| \underbrace{\tilde{M}_n||ID_n}_{\text{of length } l/R - l_r} . \quad (4)$$

This divides $\tilde{m}$ into equal size blocks $\tilde{m}=\tilde{m}_1||\tilde{m}_2||\ ...\ ||\tilde{m}_n$.

5. For each piece in $\tilde{m}$ chose a random string r_i of constant length l_r and compute.

$$m = \underbrace{\tilde{m}_1 \oplus_3 e(r_1)||r_1}_{\text{of length } l/R} || \underbrace{\tilde{m}_2 \oplus_3 e(r_2)||r_2}_{\text{of length } l/R} ||...|| \underbrace{\tilde{m}_n \oplus_3 e(r_n)||r_n}_{\text{of length } l/R} . \quad (5)$$

Here, $\oplus_3$ denotes direct sum modulo 3. This divides m into equal size blocks $m=m_1||m_2||\ ...\ ||m_n$ where $mi=\tilde{m}_i \oplus 3\ e(r_i)||r_i$.

6 Represent each block R times, i.e., let

$$t = \underbrace{m_1||m_1||\dots||m_1}_{\text{represented as } R \text{ copies}} , \underbrace{m_2||m_2||\dots||m_2}_{\text{represented as } R \text{ copies}} , \dots , \underbrace{m_n||m_n||\dots||m_n}_{\text{represented as } R \text{ copies}} .$$
$$= \underbrace{t_1}_{\text{has length } l} , \underbrace{t_2}_{\text{has length } l} , \dots , \underbrace{t_n}_{\text{has length } l} .$$

7 Represent DNA as *A, C, G, T* (rather than base 3) using the base-3 to DNA encoding [10].

While STEP 1 is very easy to apply, the randomization step number 7 in step 1 may not always provide sufficient random effect to avoid any detrimental structures and properties of the obtained nt sequence [17]. Number 7 in step 1 by itself is random, but for longer sequences of, for example, 0's, may still give rise to *AC* repeats, violating the desired homopolymer condition.

STEP 2-Generation of DNA

1. Determine the code as a collection of DNA segments $G(f)=G(f_1)$: ... :$G(f_n)$, where *G* denotes the physical representation and synthesis of DNA corresponding to the design given in segment t_i of length *l* (oligo library synthesis).
2. Perform the characterization algorithms (e.g., [4], see [10]) to validate the correctness of the synthesized library.
3. Amplify and sequence a sufficient number of copies (i.e., Š times) to ensure that each base of the original message will be recovered at least *S* times.

STEP 3-Improved encoding

A cryptographic randomization function is chosen as follows to ensure sufficient randomness in number 7 in step 1-Select a cryptographic expansion function *e* that takes as input short random strings *r* and outputs longer (for the algorithm in [10] - ternary) strings *e(r)* that are random. Select l_r the length of the input string *r* and let the output length be $l/R\text{-}l_r$.

For the improved encryption scheme, the repetition also results in a R.S repetition code. The main difference is the randomization via $\tilde{m}_i \oplus_3 e(r_i)||r_i$ which makes each component completely (pseudo)-random. This step could be realized in binary (as above), or designed in combination with other codes from coding theory [18]. The Huffman code that we apply has several additional robustness features. The decoding is the reverse of coding protocol. In addition, each $\tilde{m}_i$ needs to be recovered in (5) by recovering r_i and recomputing $e(r_i)$. This allows the computation of $\tilde{m}_i$ from the first part of each m_i. Continuing as above this yields M. The original data length does not have to be included but may be. Rather, decoding follows from properties of the Huffman table. Beginning at the first part of $\tilde{M}$ find

the corresponding Huffman text words of length 5 or 6 (whichever is found in the Huffman table). Because there is no Huffman code word in the table that consists only of zeros the process terminates when the next block of length 5 or 6 are all zeros.

Using a Huffman code as in [10] has independent advantages, in addition to what is needed for error correction. It leads to shorter average text lengths than for fixed length codes. Moreover, Huffman code words that are tabulated give additional protection against errors. Indeed, there are a total of $3^5 + 3^6$ ternary words of length 5 or 6, but only a few hundred of these are words used in the table. Finally, it automatically offers additional robustness via the individual word lengths, e.g., the original text length follows from the individual Huffman codewords and does not need to be included explicitly in $\tilde{M}$.

The randomization step provides true pseudo-randomness that should provide adequate scrambling of the nt's to prevent detrimental DNA structure and formation. Each component in the message is a completely random string. Nonetheless the randomization key is automatically obtained during decoding and does not require any additional steps (e.g., sending the key to the decoder) [19]. Different r_i's are chosen to avoid the presence of the same part r. Randomization provides a mechanism to identify mutations in living systems. Algorithms described for *in vitro* applications can be extended to *in vivo* approaches like [24].

Throughout it is assumed that both R and S are ≥ 1. Thus, after sequencing each original base is recovered at least once, and each block is repeated R ≥ 1 times. As in [10], the decoding scheme identifies each base of encoded information based on a majority vote of all the read bases corresponding to its position. As each base of encoded information is represented R times, due to their minimal base coverage S, each will be represented R•S time in our R•S repetition code. From this it follows that the minimum distance of our encoding scheme is R•S [18]. Practically, the distribution of the mean number of times each base of encoded information is sequenced and the base coverage follows a normal distribution. In both [7] and [10], the mean is quite large. The limiting issue is the minimum of these, termed S. Quality control and evaluation of current sequencing algorithms dictates how many sequences need to be generated (i.e., how large Š needs to be) to obtain S>1. These do not correct situations where S=0. This occurred in Ref. 10 and required manual intervention.

STEP 4-Detection of any errors, including those during message DNA preparation

Using an R•S-fold repetition code as above, allows the detection and correction of many substitution errors during sequencing, storage, and decoding, but it cannot correct errors due to insertion or deletion, or if the error occurred earlier [20], i.e., during message DNA preparation, even when R or S is large. Alternatively, if $S=0$ for some message bases, as was the case in [10] where the sequencing reaction destroyed two regions of nt's, then the data are lost and cannot be recovered.

In fact, [21] argued that unfortunately there is no system that would provide security and protection against errors that are introduced at the level of message DNA preparation. Such errors cannot be fixed by the synthesis of enough oligos. These types of errors may lead to inaccurate decoding and change the meaning of the recovered message. This would be extremely harmful in situations when the change of meaning is not obvious or when there is no way to assign meaning, as with a cryptographic key or access code.

In such a case it would be desirable, at least, to be aware of such errors. Goldman et al. [10] employed some quality control mechanisms after DNA synthesis, library preparation, and sequencing. They compared the GC content and the k-mer frequencies along the reads with the designed DNA strings. This approach may not be able to detect data loss during the sequencing reaction. It would also not identify errors made during initial message DNA generation. For instance, faulty oligos can mimic a desired *GC* content and thereby pass the checking routine [10]. The following approaches will address these issues.

1. During encoding, append the hash value H($\tilde{M}$) to $\tilde{M}$.
2. During decoding recompute the hash value of the rst part of $\tilde{M}$ and compare it to the given hash value. These values can only agree if no errors had occurred.

In information theory, cryptographic hash functions are a means for ensuring authenticity. For example, they are used to check whether a le has been changed. The hash value of the le is stored separately and the integrity of the le is checked by computing the hash value of the actual le and comparing it with the stored hash value. If the hash values are the same, then the le is unchanged. Secure hash functions have been developed that are mathematically strong in a range of situations [1,6,22].

This step improves the previous algorithms by providing proof if errors occurred during message DNA generation. Practically, hash functions generate a very short output, so the cost in message expansion via H($\tilde{M}$) is minimal. While it is easy to appended the hash value to $\tilde{M}$, the added number of strings impacts efficiency. Effectively, the available message length is shortened by whatever length the hash value requires. Below, a more efficient approach is described.

STEP 5-Establishing authenticity of the individual segments

In order to achieve this without knowledge of the message meaning two modifications to the above scheme are needed during encoding.

1. Choice of security parameter and identifiers: Let k be a small number, e.g., k=10 or 15 (this will be further described below). Modify

number 3 in STEP 3 as follows. Let $ID_i=ID_{ter}||1^k||0 \ldots ||0i_3$, where i_3 is the base- 3 representation of i, prepended with zeros so that ID_i is of fixed length l_{id}: The difference to above is that k 1's are inserted in each identifier. This special structure will be used during a verification step during decoding.

2. Computation of m: Update equation (5) as follows:

$$m = \underbrace{\tilde{m}_1 \oplus_3 e(r_1)||r_1 \oplus_3 H(\tilde{m}_1 \oplus_3 e(r_1))}_{\text{of length } l/R} \;||\ldots||\; \underbrace{\tilde{m}_n \oplus_3 e(r_n)||r_n \oplus_3 H(\tilde{m}_n \oplus_3 e(r_n))}_{\text{of length } l/R}. \qquad (6)$$

As above, this divides m into equal size blocks $m=m_1||m_2|| \ldots ||m_n$, where now $m_i=\tilde{m}_i \oplus_3 e(r_i)||r_i \oplus_3 H(\tilde{m}_i \oplus_3 e(r_i))$.

During decoding, each m_i is validated for correctness by itself, as follows:

1. Define the first l/R- l_r digits in m_i as $\tilde{m}_i \oplus_3 e(r_i)$.
2. Compute the hash value.
3. Define r_i as the last l_r digits in m_i subtracted bitwise modulo 3 from the value in step (2).
4. Compute $e(r_i)$.
5. Obtain $\tilde{m}_i$ via $e(r_i)$ from the value obtained in step (1).
6. Obtain $\tilde{M}_i$ and ID_i from each $\tilde{m}_i$ according to their respective lengths.
7. Define z as the k digits in ID_i immediately following the (fixed length) identifying info ID_{ter} in ID_i.

Check if z is a string of k 1's. If so, return i, $\tilde{m}_i$ *and* ID_{ter}. Otherwise report an error.

STEP 5 relies on verifying the final product and proves that this is the original encoded data. Without knowing what the original data are, the algorithm can verify whether the original and final data are the same. The algorithmic security check (number 7 in STEP 5) will only pass for blocks that are entirely correct. This includes correctness for the data as well as the addressing component. In particular, any nucleotide errors as well as wrong placement will lead to an error message. To find which one is correct, only one string needs to be identified. This idea has been used for various cryptographic applications [1,6,22]. It was originally designed for achieving provable optimal asymmetric encryption [5]. That context requires the additional feature of maintaining secrecy. We adapted the underlying premise of this approach to our algorithm. A formal proof of security is given in [5]. In essence, it is impossible to get a valid cipher text, other than starting with the message (i.e., the data to be encoded) and encoding it with the above algorithm.

A necessary condition for the security check in number 7 in STEP 5 is that all of the encoded part of the block is obtained and decoded correctly. Any error during decoding will almost certainly lead to a decoded value of z that is not all 1's. This is based on the fact that if the decoded value of $e(r)$ *or of* $H(\tilde{m}_i \oplus_3 e(r_i))$ and the original value are not the same, then it is unlikely that the random cryptographic functions will result in the desired value of z. A more refined proof (relative to plaintext-awareness) is given in [5].

Any possible error during the entire synthesis process will be detected but may not be corrected at this point. This includes errors during message DNA preparation, or if the sequencing depth for some bases is zero, i.e., if some data were consumed during the process. In our algorithm, the message part contains both the original data component as well as its corresponding indexing information. The check guarantees validity of the data part and establishes that the addressing information will be obtained correctly as the check applies to both; thus, wrong placement is impossible. In particular, the algorithm can verify if the decoded message is the same as the original message, and any nt error or wrong placement will lead to an error message. Typically, with next generation sequencing, there will be numerous correct reads. Even though the original message is never seen, it can be verified as correct. Thus, alignment and comparison are not necessary. When one correct string has been identified, the process of identifying the next correct string can proceed. Finally, on average, this should be a great saving over current alignment based techniques to identify correct strings. Only one correct string needs to be identified. If there is no single correct string, this will be identified quickly by the algorithmic check, in which case majority voting needs to be applied for each base individually.

The price for this security check is the number k of 1's added to ID_i. In provably secure cryptographic systems there is considerable uncertainty regarding acceptable limits to k [1,6,13,22]. We suggest k=10 or 15 should be large enough such that a block of 1's is unlikely to be present by chance. The length of k effectively defines the possible length of $\tilde{M}$ and of ID_i.

Figure 1 summarizes examples of choices for R and S, illustrating the strengths of the described algorithm steps above without the use of cryptography. Small values of S are included to reflect poor sequencing quality. In reality, S should be much larger, making the schemes more robust. With current technologies, the probability of errors for a DNA sequence is about 1/500 during synthesis, and 1/1000 during sequencing. In [10], the mean error rate per base at the level of sequencing reads was 1/250, which was higher than the combined synthesis and sequencing error rate reported above because of additional errors that were introduced by incorrect indexing.

In information theory, cryptographic hash functions are a means for providing authenticity. Secure hash functions have been developed that are mathematically strong in a range of situations. For example, they are used to check whether a file has been changed. The hash value of the file is stored separately and the integrity of the le is checked by computing the hash value of the actual file and comparing it with the stored hash value. If the hash values are the same, then the file is unchanged.

The incorporation of hash functions improves the previous algorithms by providing proof as to whether errors occurred during message DNA generation. Practically, hash functions generate a very short output, so the cost in message expansion via H($\tilde{M}$) is minimal.

With current technologies, synthesis and sequencing results in a large pool of correct oligos, or strings. This was confirmed in [10] where after amplification and sequencing the read duplication level was high, providing many reads covering any single string.

Modern sequence alignment algorithms have several challenges. The two most important relate to the enormous amount of short reads generated by next generation DNA sequencing technologies and the appropriate choice of the reference genome, allowing for mismatches and gaps. Contrary to algorithms developed for sequence alignment for DNA, encoding the decoder has no known reference sequence. A large number of reads are received, many with errors. Currently, majority voting for each individual base is used to inform decision making regarding the most likely correct base [24]. The main problem with this approach is the enormous amount of data that need to be stored, compared and evaluated, as well as the problem of systemic errors. We present a solution that does two things. First, it identifies one complete correct sequence among the many assuming that at least one complete correct sequence exists. The main task is to differentiate it from the many without having to analyze enormous numbers of comparisons. Second, we add randomization to deter any systemic errors.

Example: Generally, the value if R is chosen such that R.S is bigger than a required Hamming distance (**Figure 1**). Here, we use an example for R=3 to illustrate application of the algorithm. If l=198 is the largest integer less or equal to l_0=200 for which l/R=66 is an integer. In the example our le identifier has fixed length 3, ternary string ID_{ter}:= 021.

Let the message be given in ternary as M=220021200222222102101001102221012211110021120 1-121111210120202212221101001112220000200022101111021012100102120002002001102101212221111202110001002021200012000111200210212101110210100112001000220122220112210012110. The length of this is 210 and can be written in ternary as l_3:= 210=(21210)$_3$.

Choice of parameters and encoding

- Let l_r=5 be the length of the randomizers r_i.
- Let l_{id}=17 be the length of the identifier, i.e. the addressing information for each message component.
- Let the fixed length of the ternary representation of l_M be l_b=8. Recall that l_M is l_3 prepended with zeros to give this desired length. Here, l_M=00021210.
- To obtain $\tilde{M}$, one has to append exactly p zeros to M, such that the length of M plus l_b plus p is divisible by $(l/R\text{-}l_{id}\text{-}l_r)=44$. Here, 210+8+2=220=5.44 shows that p=2. Therefore, $M=M||0000021210$, where M is as above in ternary.
- Then $n:=len(\tilde{M})=(l/R\text{-}l_{id}\text{-}l_r)$ is the number of blocks that will have to be sequenced. Here, $len(\tilde{M})$ is the length of the ternary representation of M and therefore n=220/44=5.
- Let the length of the ternary representation of the internal counter i be 10. This is the counter of the blocks, i.e., i=1,….,n.
- Let the size of the security parameter k be 10. This is the number of, say, 1s that is appended to the counter i inside of each identifier.

This results in n=5 identifiers ID_1=02111111111110001, ..., ID_5=02111111111110012 which are obtained from ID_{ter}=021, appended with the k 1s and the counter i (in ternary) with internal padding to give the fixed length of each ID_i to be l_{id}=17.

Specifically,$\tilde{M}$=220021200222222102101001102221012211110021120111211112101202022122211010011122200002000221011110210121001021200020020011021012122211112021100010020212000120001112002102121011102101001120010002201222201122100121100000021210 (which is of length 220). Further, when incorporating the specific identifiers including the string of the k 1s, the n=5 internal blocks $\tilde{m}_i$ are $\tilde{m}_1$=220021200222222102101001102221012211110021120211111111110001, ..., $\tilde{m}_5$=010011200100022012222011221001211000000212100211111111110012. Each of these has length $l/R\text{-}l_r$=61.

Let r_1=02000, ..., r_5=11211 random ternary strings of length l_r=5.

`The expansion function e may be computed via the internal hash function of Maple, by computing the hash of these r_i. The output of the hash function is a hexadecimal string of length 32 that can be converted into the corresponding ternary string. To find the $e(r_i)$, the ternary representation of hash of r_i, $H(r_i)$, is chosen for only the first $61=len(\tilde{m}_i)$ ternary digits. For instance, H(r_1)=82fc813ae79eea7e3c24af961f59e6cf and therefore $e(r_1)$=11200020220212101012022120122121212110201111221110101200100 21.Bitwise addition modulo 3 of $\tilde{m}_i$ and e(r_1) results in 00202110212101011222122200011222100221200220212221212012001200 22 which is the first part

of m_1. which is the first part of m_1.

Analogously, the hash of this is now computed via the hexadecimal hash of Maple converted to ternary and then truncated the desired length l_r=5. Again, the hash is added bitwise modulo 3 to r_i to give the second part of the mi. For instance, $H(\tilde{m}_i \oplus_3 e(r_1)) \oplus_3 r_1$=12200. Therefore, e.g., m_1=00202110212101011222122200011222100221200220212221212011200 22122000020211021210101122221222000112221002212002202122212120112002212200.

The t_i are constructed by repeating each of the m_i exactly R=3 times. E.g., t_1=00202110212101011222-12220001122210022120022021222121201120022122000020211021210101122221222000112221002212002202122-22121201120022122000020211021210101122221222000112221002212002202122212120112002212200. Each of the t_i has length 3 66=198 which is l, as desired. Finally, each of these is represented in base four via the base-3 to DNA encoding ([10], **Table 1**) by avoiding homopolymers.

Overall, the result of the encoding part is a list of n=5 DNA segments, e.g., the first being DNA_1=CGCGCTCGCTGACTAGATGCTGCAC GTCTGCAGTATGATACATATCATGATCACTCAC-GCAGCACGTATATCTATCAGTCGAGCATCATGTACTCATGACGCAGCGTGCGCTGCAGCTGT-CTGTATGATGTACGCGCTCGCTGACTAGATGCTGCACGTCTGCAGTATGATACATATCATGATCACTCACGCAGCACG.

Decoding

Each of the n segments of the DN Ai is reconverted into their base-3 equivalents.

For example: DNA_1 becomes 00202110212101011222122200011222100221200220212221212011200221220000202110212-10101122221222000112221002212002202122212120112002212200002021102121010112222122200011222100221-12002202122212120112002212200 (which, if correct, is t_1). Each of these n recovered segments are then divided into R subcomponents of equal size. For instance, the first component gives the three subpieces 00202110212101011222122200011222100221200220212221212011200221220000 2110221200220212221212011200221220000; 00202110212101011222122-00011222100221200220212221212011200221220000; 00202110212101011222122200011222100221200220212221-12120112002212200: These should all be the same strings, repeated R=3 times. During decoding, a simple majority vote will compare each of the entries in these R subpieces and return the most likely entry for each individual digit. Here, since there were no errors, only one of these (same) substrings is computed. Doing this for all the n=5 recovered DNA strings yields the original strings $m_1,\ldots,m_5$.

This concludes the part of the method that utilizes repetition codes. However, this part may not detect all possible errors. The next steps illustrate how the additional components derived from cryptographic methods address this issue.

Each of the recovered blocks (which should be the m_i and which for simplicity are termed mi although it still must be confirmed that they are the same as the original m_i) is by itself validated for correctness, as detailed in STEP 5 as follows.

Let Y_i be the last l/R-l_r=61 digits for each of the recovered blocks m_i. For instance, the Y_i recovered from m_i is "0020211 0212101011222122200011222100221200220212221212011200022". The hash value for this gives "19e30bb67ee7486 df967389903d1487b". This allows for the recomputation of the r_1=02000. The expansion function applied to this yields $e(r_1)$=1120002022021210101202212012212121211020111122111 0-10120010021. Subsequently, $\tilde{m}_1$ is recovered as 2200212002222221021010011022210122111100211202111 11-1111110001.

The crucial step is done via recovering the ID_i and testing if they are of the correct form, i.e. with a desired number of 1's. For instance ID_1 is recovered as 02111111111110001. The first three digits yield ID_{ter}=021 and the next k=10 digits are crucial. If there is no error, the string will be 1111111111, i.e., the test string z of k=10 consecutive 1's. If this test is passed, the entire component m_i is therefore proven to be correct [5]. If all the m_i are validated then this gives the original message m along with a proof that the recovered m is the same as the original.

Error detection and correction

The internal repetition into R segments constitutes a classic repetition code with straight forward error correcting properties. A simple majority vote will identify less then $(R-1)/2$ substitution errors. Internal R fold repetition is achieved and is demonstrated above to illustrate the choice of the parameters. However, the strength of our method does not rely on this part and indeed one could achieve strong security even for R=1, due to the cryptography part integrated in STEP 4 and STEP 5.

If the size of the security parameter k is large enough then any category of error will be identified during decoding **(Table 3)**. In case of any error, the security check will not pass for a large enough k. For instance, if during decoding the second piece m_2 suffers from a deletion at position 3 (after majority decoding), then the decoding routine will recompute the following ID_2=0022101122012: It does not end in the desired $k \geq 5$ 1s and the error is detected.

According to [5], it does not matter what type and how many errors occur. Most importantly in order to obtain the desired string of k 1's, the entire message $\tilde{m}_i$ needs to be recomputed correctly; this is known as “all-or-nothing" security. In order to recover $\tilde{m}_i$, one must recover both parts of $\tilde{m}_i$ in their entirety. The first part is required to recover r_i from the second part, and r_i is required to recover $\tilde{m}_i$ from its first part. Since any changed bit of a cryptographic hash completely changes the result, the entire part of $\tilde{m}_i$ must both be

completely recovered. Above, *R=3* allows the correction of single substitution errors during majority voting. The extra security feature testing for the *k* 1's provides an additional feature that identifies any type of error that escaped detection, including substitution, deletion, or insertion errors. It gives a guarantee for those strings that are recovered in their entirety. The list of abbreviations is given in the **Table 4**.

Conclusion

A multistep algorithm is presented based on cryptographic features that increase precision and security. Two main ingredients are randomization and secure data integrity. The method is flexible and allows for increased levels of security, depending on demand, and identifies approaches to detect and correct errors. This is based on the number of DNA blocks required to be encoded within each generated sequence of DNA (as represented by *R*), or how many need to be sequenced (as represented by *S*). In contrast, the Illumina protocol [2] focuses on the average curve of the sequencing depth. In worst case scenarios, i.e., when certain nucleotide bases are consumed during the encoding process, algorithm will detect the loss of these data.

Our approach is applicable to *in vitro* and *in vivo* approaches of DNA storage and sequence analysis. Key for both is adequate randomness in the encoding algorithm. *In vitro* approaches require sufficient redundancy to prevent detrimental DNA sequence and structure. *In vivo* approaches must avoid the existence of the same or similar copies of DNA within a single cell. The required level of randomization is achieved by tools derived from cryptography that do not rely on knowledge or transmission of secret keys or randomization data. Another potential application of our algorithm is for the identification of genetically modified organ-isms. Effectiveness and accuracy require multiple copies of a trademark DNA label or "barcode" containing authentication and tracking information inside a single cell. This challenge can be efficiently addressed based on the randomization techniques described; each individual segment of DNA represents the same information, but is randomized before being encoded in DNA.

As the algorithm can be applied to validate and verify the correctness of the individually decoded pieces, without knowing the original data, a certificate can be generated that confirms that the obtained data are exactly like the original ones. This has applications to DNA

Table 3 Advanced error detection and correction via the use of cryptography. A clearly defined check routine identifies those reads that are error free. Here, these are exactly those for which the check string contains a desired number of 1's. This check will identify all types of errors and identify only those reads that are correct. Details of the implementation and a worked example for correct decoding is given in the example in the text.

Any error(s) corrected during majority decoding	Reconstructed m_2 (after majority voting) 0 1 0 1 1 0 1 1 1 1 . . . 0 0 2 0 2 1 1 2 1 0	Reconstructed check string "1111111111"
Example of substitution error	0 1 2 1 1 0 1 1 1 1 . . . 0 0 2 0 2 1 1 2 1 0	"2002202000"
Example of insertion error	0 1 2 0 1 1 0 1 1 1 1 . . . 0 0 2 0 2 1 1 2 1 1	"0110120120"
Example of deletion error	0 1 1 1 0 1 1 1 1 . . . 0 0 2 0 2 1 1 2 1 0	"0020122120"

Table 4: List of Abbreviations.

Short form	Full form
l_0	Largest length of segments chosen that are amenable to manipulation
l	The actual block length of each sequence that is sequenced. $l \leq l_0$ is chosen according to some technical requirements depending on the required number of repetitions within each block
R	The number of repetitions of the same sequence inside each sequenced block. R is chosen such that R/l is an integer and according to some additional technical requirements detailed below to ensure a required hamming distance
l_{id}	Length of the identifier for each message block
l_r	Length of the input string r
ID_i	Concatenation of the file identifier and the internal counter
S	Sequencing depth. A Parameter that identifies the number of sequences to be recovered
M	Original message
$M_1 \| \| M_2$	Concatenation of strings M_1 and M_2
$\tilde{M}$	M appended and padded with additional 0's, resp., the message length, to allow unique decoding. The length of $\tilde{M}$ is a multiple of $\frac{l}{R}-l_{id}$ (basic scheme) or of $\frac{l}{R}-l_{id}-l_r$ (enhanced scheme)
$\tilde{M}_i$	The equal length blocks within $\tilde{M}$
$mi, \tilde{m}_i$	The individual message blocks with their identifiers, after incorporation of the respective cryptographic steps
r, r_i	Pseudorandom string of length l_r
$e(r_i)$	A cryptographic expansion function that takes a short random string r_i and outputs a longer random string $e(r_i)$
$\tilde{m}_i \oplus_3 e(r_i) \| \| r_i$	Random string r_i of length l_r chosen from data m and direct summed $\oplus_3$ modulo 3 with m. Each of these has length $\frac{l}{R}$
$H(M)$	Hash value of M
k	Security parameter underlying the cryptographic OAEP scheme
z	Last k digits in $\tilde{m}_i$. These k strings are checked for a specific pattern. Used in STEP 5 to ensure authenticity of the individual segments

steganography, sequence alignment, and fast identification of correct reads in next generation sequencing.

Acknowledgments

This work was supported by NSF-40243.

References

1 IACR (2015) International Association for Cryptologic Research.

2 Illumina (2016) An introduction to next-generation sequencing technology.

3 Ailenberg M, Rotstein O (2009) An improved Huffman coding method for archiving text, images, and music characters in DNA. Biotechniques 47: 747-754.

4 Andrews S (2010) Fast QC: A quality control tool for high throughput sequence data. Babraham bioinformatics.

5 Bellare M, Rogaway P (1995) Optimal asymmetric encryption. EUROCRYPT 950: 92.

6 Buchmann J (2004) Introduction to Cryptography. Springer Verlag.

7 Church GM, Gao Y, Kosuri S (2012) Next-generation digital information storage in DNA. Science 337: 1628.

8 Cox JP (2001) Long-term data storage in DNA. Trends Biotechnol 19: 247-250.

9 Gibson DG, Benders GA, Andrews-Pfannkoch C, Denisova EA, Baden-Tillson H, et al. (2008) Complete chemical synthesis, assembly, and cloning of a mycoplasma genitalium genome. Science 319: 1215.

10 Goldman N, Bertone P, Chen S, Dessimoz C, LeProust EM, et al. (2013) Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. Nature 494: 77-80.

11 Haughton D, Balado F (2013) BioCode: two biologically compatible Algorithms for embedding data in non-coding and coding regions of DNA. BMC Bioinformatics 14: 121.

12 Heider D, Barnekow A (2007) DNA-based watermarks using the DNA-Crypt algorithm. BMC Bioinformatics 8: 176.

13 Kiltz E, OeNeill A, Smith A (2010) Instantiability of rsa-oaep under chosen-plaintext attack. CRYPTO 2010: 295.

14 Leier A, Richter C, Banzhaf W, Rauhe H (2000) Cryptography with DNA binary strands. Biosystems 57: 13-22.

15 Li H, Durbin R (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics 25: 1754-1760.

16 Liss M, Daubert D, Brunner K, Kliche K, Hammes U, et al. (2012) Embedding permanent watermarks in synthetic genes. PLoS One 7: e42465.

17 Mollin RA (2010) An Introduction to Cryptography. CRC Press.

18 Roman S (1997) Introduction to Coding and Information Theory. Springer.

19 Schneier B (1996) Applied Cryptography. 2nd John Wiley and Sons, Inc.

20 Shimanovsky B, Feng J, Potkonjak M (2003) Hiding data in DNA. In Information Hiding.

21 Smith GC, Fiddes CC, Hawkins JP, Cox JP (2003) Some possible codes for encrypting data in DNA. Biotechnol Lett 25: 1125-1130.

22 Stinson DR (2006) Cryptography: Theory and Practice. CRC press.

23 Yachie N, Ohashi Y, Tomita M (2008) Stabilizing synthetic data in the DNA of living organisms. Systems and Synthetic Biology.

24 Yachie N, Sekiyama K, Sugahara J, Ohashi Y, Tomita M (2007) Alignment-based approach for durable data storage into living organisms. Biotechnol Prog 23: 501-505.