# Method for Encoding and Decoding Arbitrary Computer Files in DNA Fragments

(Version 2.0: 21 August 2013)

Nick Goldman

EMBL-European Bioinformatics Institute, Hinxton, United Kingdom

## 1 Encoding

**1.1:** An arbitrary computer file is represented as a string $S_{\emptyset}$ of bytes (often interpreted as a number between $\emptyset$ and $2^8 - 1$, i.e. a value in the set $\{\emptyset \ldots 255\}$)*.

**1.2:** $S_{\emptyset}$ is encoded using a given Huffman code, converting it to base-3. This generates the string $S_1$ of characters in $\{\emptyset, 1, 2\}$, each such character called a 'trit'.

**1.3:** Write $len()$ for the function that computes the length (in characters) of a string, and define $n = len(S_1)$. Represent $n$ in base-3 and prepend $\emptyset$s to generate a string $S_2$ of trits such that $len(S_2) = 25$. Form the string concatenation $S_4 = S_2 \,.\, S_1 \,.\, S_3$, where $S_3$ is a string of at most 24 $\emptyset$s chosen so that $len(S_4)$ is an integer multiple of 25.

**1.4:** $S_4$ is converted to a DNA string $S_5$ of characters in $\{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ with no repeated nucleotides (nt) using the scheme illustrated in Figure 1. (The first trit of $S_4$ is coded using the 'A' row of the table. For each subsequent trit, characters are taken from the row defined by the previous character conversion.)

**1.5:** Define $N = len(S_5)$, and let $ID$ be a 2-trit string identifying the original file and unique within a given experiment (permitting mixing of DNA from upto $3^2 = 9$ different files $S_{\emptyset}$ in one experiment). Split $S_5$ into overlapping segments of length 1$\emptyset\emptyset$ nt, each offset from the previous by 25 nt. This means there will be $\frac{N}{25} - 3$ segments, conveniently indexed $i = \emptyset \ldots \frac{N}{25} - 4$; segment $i$ is denoted $F_i$ and contains (DNA) characters $25i \ldots 25i + 99$ of $S_5$.

Each segment $F_i$ is further processed as follows:

*$\emptyset$ is used throughout to represent the number zero, to avoid confusion with letters o and O.

| previous nt written | next trit to encode | | |
|---|---|---|---|
| | Ø | 1 | 2 |
| A | C | G | T |
| C | G | T | A |
| G | T | A | C |
| T | A | C | G |

**Figure 1: Base-3 to DNA encoding ensuring no repeated nucleotides.** For each trit $t$ to be encoded, select the row labelled with the previous nt used and the column labelled $t$ and encode using the nt in the corresponding table cell.

**1.6:** If $i$ is odd, reverse complement $F_i$.

**1.7:** A keystream method is used to add randomness to the pattern of bases of each segment $F_i$. The 1ØØ DNA characters of $F_i$ define 99 single-trit values by converting characters A, C, G, T to numbers Ø, 1, 2, 3; then forming the successive differences by subtracting (base-4) each such number from its successor ($d_2 = n_2 - n_1$, $d_3 = n_3 - n_2$ ... $d_{1ØØ} = n_{1ØØ} - n_{99}$); and then subtracting 1 from each of these numbers. (The fact there are no repeated nt in $F_i$ guarantees that the results of this process are valid trits.) To each of these 99 trits, add (base-3) the corresponding trit from the $j$th following random keystream, where $j = i \bmod 4$:

**keystream Ø:** ØØ2ØØØØ1Ø11Ø1Ø2111112122210ØØ11122221Ø1Ø1Ø212122ØØ2
2ØØØ2212Ø1Ø2Ø221ØØ2121121ØØØ212222Ø21211121122221

**keystream 1:** 2Ø2Ø2Ø12212121Ø12ØØØ12ØØ21Ø222112Ø2Ø222Ø2222222Ø22
ØØØ1221Ø12111Ø2212112Ø2Ø2Ø222112211122Ø2ØØ2121Ø22

**keystream 2:** 2212211Ø12ØØ22112Ø22ØØ11ØØ22221ØØØØØØ2Ø2ØØØ2112 1Ø2
1Ø2Ø1221ØØØ212Ø1Ø1Ø21Ø2Ø2ØØ2ØØØ1Ø1Ø2ØØ221211ØØ1ØØ

**keystream 3:** 1ØØ1221ØØØ111121ØØ12Ø21ØØ2ØØ111Ø22Ø1122122 1ØØ12
Ø122212ØØØØ2122ØØ22Ø122Ø22Ø11ØØØ1Ø21222211Ø222Ø2Ø

The 99 resulting trits give the 'randomized' version of $F_i$ by the reverse of the procedure described at the start of this step: add 1 to each to generate base-4

differences ($d'_2 \ldots d'_{1\emptyset\emptyset}$); then, starting with $n_1$ (unaltered from above), add (base-4) successive differences to generate 1ØØ successive values in {Ø,1,2,3} ($n_1$, $n_1 + d'_2$, $n_1 + d'_2 + d'_3 \ldots n_1 + d'_2 + d'_3 + \ldots + d'_{1\emptyset\emptyset}$); and then convert values Ø, 1, 2, 3 to nt A, C, G, T.

**1.8:** Let *i3* be the base-3 representation of $i$, appending enough leading Øs so that $len(i3) = 12$. Compute $P$ as the sum (mod 3) of the odd-positioned trits in $ID$ and *i3*, i.e. $ID_1 + i3_1 + i3_3 + i3_5 + i3_7 + i3_9 + i3_{11}$. ($P$ acts as a 'parity trit'—analogous to a parity bit—to check for errors in the encoded information about $ID$ and $i$.)

**1.9:** Form the indexing information string $IX = ID \,.\, i3 \,.\, P$ (comprising 2+12+1 = 15 trits). Append the DNA-encoded version of $IX$ to $F_i$ using the same strategy as at step 1.4 above, starting with the code table row defined by the last character of $F_i$, to give indexed segment $F'_i$.

**1.10:** Form $F''_i$ by prepending A or T and appending C or G to $F'_i$—choosing between A and T, and between C and G, randomly if possible but always such that there are no repeated nt. (This ensures that we can distinguish a DNA segment that has been reverse complemented during DNA sequencing from one that has not—the former will start with G|C and end with T|A; the latter will start A|T and end C|G.)

See Figure 2 for a schematic representation of the DNA-encoding of computer files.

**1.11:** The segments $F''_i$ are synthesized as actual DNA oligonucleotides and stored, and may be supplied for sequencing.

# 2 Example

**2.1:** As it is difficult to represent all possible bytes in this document, we use a simple example of a file comprising just 18 bytes that happen to be easily represented via ASCII codes (for clarity, we show spaces as ␣):

$$S_\emptyset = \texttt{Birney␣and␣Goldman}$$

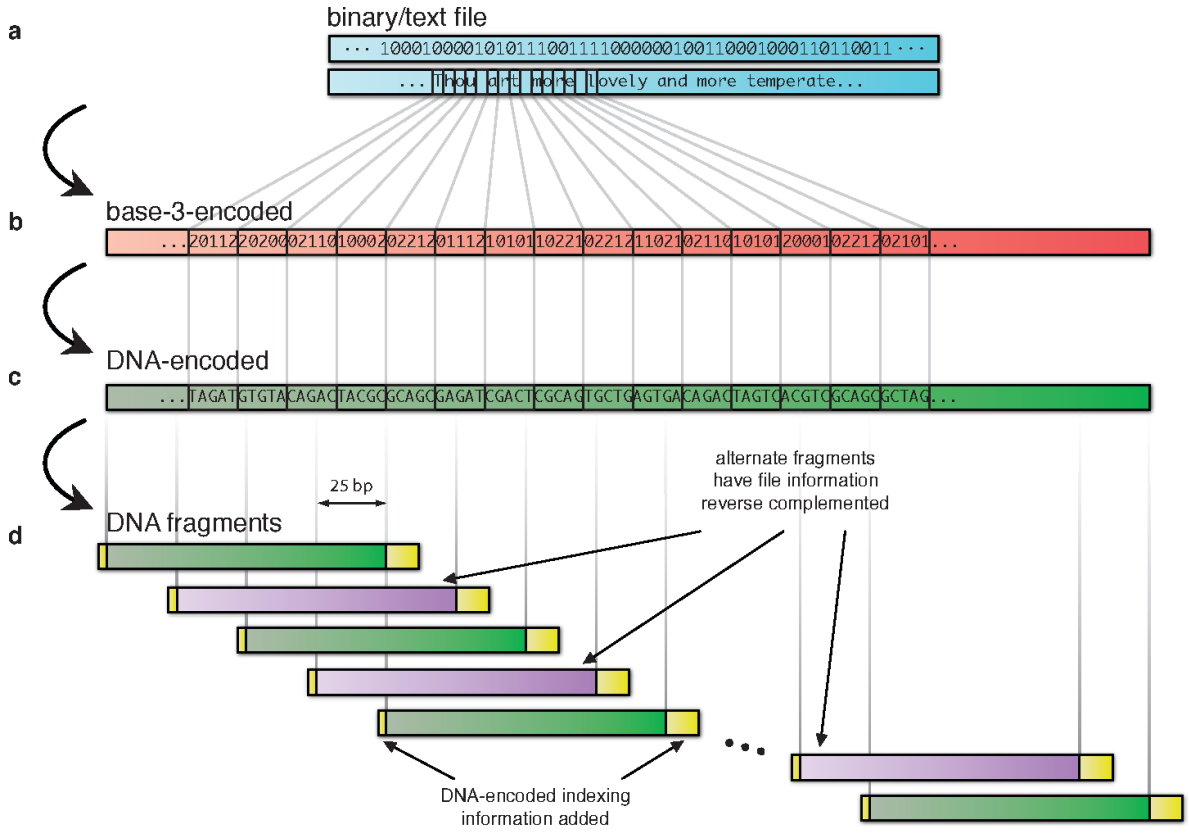


**Figure 2: Schematic of DNA-fragment-encoding of computer files.** The computer file (in any format, e.g. binary or text) shown in blue in (a) (step 1.1 above) is encoded in base-3 (red in b; 1.2–1.3) and then as DNA with no repeated nt (green in c; 1.4). This representation of the complete file is then split into overlapping segments (1.5), alternate segments reverse complemented (mauve; 1.6), and segment indexing and direction-determining information added (yellow; see d; 1.8–1.10). This diagram does not explicitly depict the 'keystream randomization' step (1.7 above).

**2.2:** Using the Huffman code defined in the example file `View_huff3.cd`, we convert the bytes of $S_\emptyset$ (shown above/below $S_1$ for illustrative purposes) into base-3[†]:

[†]Throughout what follows, spaces are not part of base-3 or DNA strings but are included to assist in 'reading' how strings have been derived in each step.

| | B | i | r | n | e | y | ␣ | a | n |
|---|---|---|---|---|---|---|---|---|---|
| $S_1$ = | 2Ø1ØØ | 2Ø21Ø | 1Ø1Ø1 | ØØØ21 | 2ØØØ1 | 222111 | Ø2212 | Ø1112 | ØØØ21 |
| | 221ØØ | Ø2212 | 222212 | Ø211Ø | Ø21Ø1 | 221ØØ | 11Ø21 | Ø1112 | ØØØ21 |
| | d | ␣ | G | o | l | d | m | a | n |

**2.3:** $n = len(S_1) = 92$, which is 1Ø1Ø2 in base-3. So:

$$S_2 = \text{ØØØØØØØØØØØØØØØØØØØØØ}\ \ \text{1Ø1Ø2 (length 25)}$$

$$S_3 = \text{ØØØØØØØØ (length 8)}$$

$$\begin{aligned} S_4 &= S_2 \,.\, S_1 \,.\, S_3 \\ &= \text{ØØØØØØØØØØØØØØØØØØØØØ1Ø1Ø2} \\ &\quad \text{2Ø1ØØ2Ø21Ø1Ø1Ø1ØØØ212ØØØ1222111Ø2212Ø1112ØØØ21} \\ &\quad \text{221ØØØ22122222212Ø211ØØ21Ø1221ØØ11Ø21Ø1112ØØØ21} \\ &\quad \text{ØØØØØØØØ (length } 25 + 92 + 8 = 125 = 5 \times 25\text{)} \end{aligned}$$

**2.4:** Using the DNA coding strategy and table shown at step 1.4 above, convert $S_4$ to DNA:

$S_5$ = CGTACGTACGTACGTACGTAGTCGCACTACACAGTCGACTACGCTGTACTGCAGAGTGCTGTC
TCACGTGATGACGTGCTGCATGATATCTACAGTCATCGTCTATCGAGATACGCTACGTACGT

**2.5:** $N = len(S_5) = 125$, and we choose (e.g.) $ID = 12$. $S_5$ will be split into overlapping segments $F_i$ of length 1ØØ nt for $i \in \{Ø \ldots \frac{125}{25} - 4\}$, i.e. $i \in \{Ø, 1\}$. With overlapping parts underlined for illustration, $F_Ø$ and $F_1$ are:

$F_Ø$ = CGTACGTACGTACGTACGTAGTCGC ACTACACAGTCGACTACGCTGTACT
GCAGAGTGCTGTCTCACGTGATGAC GTGCTGCATGATATCTACAGTCATC

$F_1$ = ACTACACAGTCGACTACGCTGTACT GCAGAGTGCTGTCTCACGTGATGAC

GTGCTGCATGATATCTACAGTCATC GTCTATCGAGATACGCTACGTACGT

**2.6:** Only $i = 1$ is odd, so $F_1$ is reverse complemented:

$F_1$ = ACGTACGTAGCGTATCTCGATAGACGATGACTGTAGATATCATGCAGCAC

GTCATCACGTGAGACAGCACTCTGCAGTACAGCGTAGTCGACTGTGTAGT

**2.7:** Fragments $F_\emptyset$ and $F_1$ are randomized using keystreams $\emptyset$ and 1 (step 1.7), respectively. Full details of the example are omitted, but after keystream randomization these become:

$F_\emptyset$ = CGTGTACGACTCGACAGAGATGCAGAGACTAGACACTCTCTATACGCATC

TCAGACACTCGTAGTCTACTGACATCGCACGAGAGAGAGCAGCGCACTCA

$F_1$ = ATATATAGCGTGATAGCGTCAGTCGCATCGCACACGAGCTCTCTCTAGCA

TAGCGTCTAGATGCGACGAGCGAGCTCATGTGATCTGCGTACTCTCTACA

**2.8:** For $i = \emptyset$, $i3 = \emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset$ (length 12) and the sum (mod 3) of the odd-positioned trits of $ID$ and $i3$ is $P = 1 + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset \pmod 3 = 1$. For $i = 1$, $i3 = \emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset 1$ and $P = 1 + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset \pmod 3 = 1$.

**2.9:** For $i = \emptyset$, $IX = ID\ .\ i3\ .\ P = 12\ \emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\ 1$; for $i = 1$, $IX = 12\ \emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset 1\ 1$. So:

$F'_\emptyset$ = CGTGTACGACTCGACAGAGATGCAGAGACTAGACACTCTCTATACGCATC

TCAGACACTCGTAGTCTACTGACATCGCACGAGAGAGAGCAGCGCACTCA

GC GTACGTACGTAC T (length $1\emptyset\emptyset + 15 = 115$)

$F'_1$ = ATATATAGCGTGATAGCGTCAGTCGCATCGCACACGAGCTCTCTCTAGCA

TAGCGTCTAGATGCGACGAGCGAGCTCATGTGATCTGCGTACTCTCTACA

GC GTACGTACGTAG A (length 115)

**2.10:** Prepend A|T and append C|G (note that in this example we have three random choices, but no choice at the start of $F''_1$):

$F''_{\emptyset}$ = A CGTGTACGACTCGACAGAGATGCAGAGACTAGACACTCTCTATACGCATC
TCAGACACTCGTAGTCTACTGACATCGCACGAGAGAGAGCAGCGCACTCA
GCGTACGTACGTACT G (length $1 + 115 + 1 = 117$)

$F''_1$ = T ATATATAGCGTGATAGCGTCAGTCGCATCGCACACGAGCTCTCTCTAGCA
TAGCGTCTAGATGCGACGAGCGAGCTCATGTGATCTGCGTACTCTCTACA
GCGTACGTACGTAGA C (length 117)

# 3 Decoding

Decoding is simply the reverse of encoding, starting with sequenced DNA fragments $F''_i$ of length 117 nt. Reverse complementation during the DNA sequencing procedure (e.g. during PCR reactions) can be identified for subsequent reversal by observing whether fragments start with A|T and end with C|G, or start G|C and end T|A. With these two 'orientation' nt removed, the remaining 115 nt of each segment can be split into the first 1ØØ 'message' nt and the remaining 15 'indexing' nt. The indexing nt can be decoded to determine the file identifier *ID* and the position index *i3* and hence $i$, and errors may be detected by testing the parity trit $P$. The value of $i$ permits the identification of the correct random keystream, the randomizing effect of which can be removed by applying the keystream encoding (step 1.7) a second time, but *subtracting* the keystream trits. The file identifier and position index then permit reconstruction of the DNA-encoded files, which can then be converted to base-3 using the reverse of the encoding table in step 1.4 above and then to the original bytes using the given Huffman code. Precise details of the decoding procedure are left as an exercise for the reader.

Errors introduced during DNA synthesis, storage or sequencing could lead to various artefacts, particularly nt insertion, deletion or substitution. Recovery of information from fragments with such errors may be possible (details left as exercise)—we have not found this to be necessary due to the large numbers of perfectly-sequenced fragments available via high-throughput sequencing.

# 4 Version history

**Version 2.0:** Updated to reflect changes to representation of file length (step 1.3 above) and introduction of keystream encoding (step 1.7 above), with corresponding changes elsewhere.

**Version 1.0:** First complete, finalized description, as published with the *Nature* paper.

**Versions 0.1–0.4:** Preliminary versions of original description.

# 5 Keystream randomization code

As a further explanation of the keystream randomization of step 1.7 above, the following Perl subroutine implementing it is included:

```
sub encrypt {   # subroutine to perform keystream encoding/decoding
                # call it using: $newDNA=&encrypt($oldDNA,$keystream,1); (to set $newDNA to be $oldDNA encoded
                   using $keystream)
                #               $oldDNA=&encrypt($newDNA,$keystream,0); (to set $oldDNA to be $newDNA decoded using
                   $keystream)

       my ($DNA,$keystream,$add,$i,$L,@bases,@kstrits); # keep everything local!

# The following lines are copied from code that calls this subroutine,
# as we expect keystreams to be defined there and passed into the subroutine.
# They are included here as a record of the 4 keystreams in use from late-July 2013,
# but commented-out as they are not needed in this implementation.

# The first 4 of the following are random keystreams generated on 29vii13 using
# http://www.random.org/integers/?num=99&min=0&max=2&col=1&base=10&format=html&rnd=new
# and with an arbitrary 0 added to the start (it is not used).
# The 5th one is all 0s --- this might be useful as it is the case of *not* using a keystream.
#@ks=("000200001011010211111212221001112222101010212122202200022120102022100212112100021222202121111211222221",
#      "020202012212121012000120021022211202022202222222022000122101211102221121202020222112211122020021210 22",
#      "022122110120022112022001100222210000002020002112102102012210002120101021020200200010102002212110010 0",
#      "010012210001111210012021002001110220112212210010012012221200002122002201220220110001021222211022202 0",
#      "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000");

       ($DNA,$keystream,$add)=@_;                    # DNA string to encrypt; keystream to do it with;
                                                    # $add=any 'true' value for *adding* keystream (encrypting),
```

```
                                                  'false' for subtracting (decrypting)
$L=length($keystream);
if (length($DNA)!=$L) {die "\&encrypt error";}

$DNA =~ tr/acgtACGT/01230123/;                  # convert A,C,G,T to 0,1,2,3 (ignore case)
@bases=split(//,$DNA);                         # split to arrays for ease of processing
@kstrits=split(//,$keystream);

foreach (1..$L-1) {                            # compute differences
       $i=$L-$_;                                  # (note: this is performed right to left
       $bases[$i]=($bases[$i]-$bases[$i-1]) % 4;  #        in order to do it in-place)
       }

foreach (1..$L-1) {                            # subtract 1 to recover base-3 trits
       $bases[$_]=($bases[$_]-1) % 3;
       }

foreach (1..$L-1) {                            # add/subtract keystream trits as appropriate
       $bases[$_]=($add) ? ($bases[$_]+$kstrits[$_]) % 3 : ($bases[$_]-$kstrits[$_]) % 3;
       }

foreach (1..$L-1) {                            # add 1 to generate base-4 differences
       $bases[$_]=($bases[$_]+1) % 4;
       }

foreach (1..$L-1) {                            # add successively to generate 0,1,2,3 coding
       $bases[$_]=($bases[$_]+$bases[$_-1]) % 4;
       }

$DNA=join("",@bases);                          # return to string format
```

```
        $DNA =~ tr/0123/ACGT/;                          # convert 0,1,2,3 to A,C,G,T

        $DNA;                                           # return result
        }

1; # need to end with a true value if subroutine to be used via 'require'
```