

\$ cat permute.cpp

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{ if (argc==1)
  { fprintf(stderr, "Need a permutation on the command line\n"); exit(1); }
  if ((argc-1)%8!=0)
  { fprintf(stderr, "Permutation lenght must be a multiple of 8\n"); exit(1); }
  int permlen=argc-1;
  int * perm = new int[permlen];
  int * used = new int[permlen+1];
  for (int i=0; i<=permlen; i+=1)
    used[i]=0;
  int zerobased=0, onebased=0;
  for (int i=0; i<permlen; i+=1)
  { int n=strtol(argv[i+1], NULL, 0);
    if (n==0) zerobased=1;
    if (n==permlen) onebased=1;
    if (n<0 || n>permlen)
      { fprintf(stderr, "Not a permutation: %d is out of range\n", n); exit(1); }
    perm[i]=n;
    if (used[n])
      { fprintf(stderr, "Not a permutation: %d appears twice\n", n); exit(1); }
    used[n]=1;
  if (zerobased && onebased)
  { fprintf(stderr, "Not a permutation 0 and %d can't both appear\n", permlen);
    exit(1); }
  if (onebased)
    for (int i=0; i<permlen; i+=1)
      perm[i]=-1;
  int blocksize=permlen/8;
  unsigned char * plain = new unsigned char[blocksize];
  unsigned char * cypher = new unsigned char[blocksize];
  while (1)
  { int readsome=0;
    for (int i=0; i<blocksize; i+=1)
    { int c=getchar();
      if (c==EOF)
      { if (i==0)
          break;
        c=0; }
      readsome=1;
      plain[i]=c;
      cypher[i]=0; }
    if (!readsome) break;
    int destbytenum=0, destbitnum=7;
    for (int i=0; i<permlen; i+=1)
    { int pos=perm[i];
      int srcbytenum=pos/8, srcbitnum=7-pos%8;
      if (plain[srcbytenum] & (1<<srcbitnum))
        cypher[destbytenum] |= 1<<destbitnum;
      destbitnum-=1;
      if (destbitnum==-1)
      { destbitnum=7;
        destbytenum+=1; } }
    for (int i=0; i<blocksize; i+=1)
      putchar(cypher[i]); } }
```

```

$ cat plain
One two three four five six
the cat sat on the mat.
the rain in Spain falls mainly in the plains.

$ permute 9 10 11 12 13 14 15 16 1 2 3 4 5 6 7 8 <plain
nO ewt ohter eofruf vi eis
xht eac tas tnot ehm ta
.ht earnii npSia nafll samniyli nht elpiasn

$ permute 17 18 19 20 21 22 23 24 9 10 11 12 13 14 15 16 1 2 3 4 5 6 7 8 <plain
enOwt t oerhf eruoif evxisht
c e tatasno ht m e.taht
r eniani pS niaaf sllam lnii yt n ehalpsni
$ permute 1 2 3 4 6 5 8 7 <plain
Omj x{o xdqjj iozq ifyj sftxdj cbx sbx om xdj nbx-xdj qbfm fm Spbfm iblls nbfrm lv fm xdj
plbfms-

```

```

$ set p = "6 11 2 9 16 8 5 12 15 1 7 10 4 14 3 13"
$ permute $p <plain >secret
$ cat secret
æµääžæ"à>äè·åš@-g-äl;"<àl2à
l:à
æ·AjHe,fäh:f-Hâ"e8lâ"è2âd*íf-ëHâ"à`lë²,f

$ invert $p
10 3 15 13 7 1 11 6 4 12 2 8 16 14 9 5

$ permute `invert $p` <secret
One two three four five six
the cat sat on the mat.
the rain in Spain falls mainly in the plains.

$ cat invert.cpp
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{ if (argc==1)
  { fprintf(stderr, "Need a permutation on the command line\n"); exit(1); }
  int permlen=argc-1;
  int * perm = new int[permlen];
  int * used = new int[permlen+1];
  int * inv = new int[permlen];
  for (int i=0; i<=permlen; i+=1)
    used[i]=0;
  int zerobased=0, onebased=0;
  for (int i=0; i<permlen; i+=1)
  { int n=atol(argv[i+1]);
    if (n==0) zerobased=1;
    if (n==permlen) onebased=1;
    if (n<0 || n>permlen)
      { fprintf(stderr, "Not a permutation: %d is out of range\n", n); exit(1); }
    perm[i]=n;
    if (used[n])
      { fprintf(stderr, "Not a permutation: %d appears twice\n", n); exit(1); }
    used[n]=1;
  }
  if (zerobased && onebased)
    { fprintf(stderr, "Not a permutation 0 and %d can't both appear\n", permlen); exit(1); }
  if (onebased)
    for (int i=0; i<permlen; i+=1)
      perm[i]=-1;
  for (int i=0; i<permlen; i+=1)
    inv[perm[i]]=i;
  for (int i=0; i<permlen; i+=1)
  { if (i>0) printf(" ");
    printf("%d", inv[i] + (onebased ? 1 : 0));
  }
  printf("\n");
}

```

Bit Frequencies

Analysis of English language texts, counting the relative frequency of a 1 in each of the 8 bit positions. The results are completely consistent for the five texts tested. If the bits were permuted, the frequencies alone would be enough to reverse the permutation. Bit 0 is the Most significant bit, bit 7 the least.

J. M. Barrie, Peter Pan

```
$ bitfreq < peter.pan
Bit 0, freq 0.000000
Bit 1, freq 0.744555
Bit 2, freq 0.952047
Bit 3, freq 0.239387
Bit 4, freq 0.337803
Bit 5, freq 0.459206
Bit 6, freq 0.335522
Bit 7, freq 0.425811
```

John Buchan, The Thirty-Nine Steps

```
$ bitfreq </users/home/www/text/buchan/thirtyninesteps
Bit 0, freq 0.000000
Bit 1, freq 0.768272
Bit 2, freq 0.951484
Bit 3, freq 0.233395
Bit 4, freq 0.342851
Bit 5, freq 0.478468
Bit 6, freq 0.342052
Bit 7, freq 0.443030
```

Lewis Carroll, Alice in Wonderland

```
$ bitfreq < /users/home/www/text/carroll/alice
Bit 0, freq 0.000000
Bit 1, freq 0.732873
Bit 2, freq 0.944658
Bit 3, freq 0.232395
Bit 4, freq 0.331297
Bit 5, freq 0.462412
Bit 6, freq 0.321475
Bit 7, freq 0.435387
```

Arthur Conan-Doyle, The Memoirs of Sherlock Holmes

```
$ bitfreq < /users/home/www/text/conandoyle/memoirsholmes
Bit 0, freq 0.000010
Bit 1, freq 0.746443
Bit 2, freq 0.928032
Bit 3, freq 0.239549
Bit 4, freq 0.359350
Bit 5, freq 0.481908
Bit 6, freq 0.338191
Bit 7, freq 0.458845
```

Mark Twain, Huckleberry Finn

```
$ bitfreq < /users/home/www/text/twain/huckfinn
Bit 0, freq 0.000010
Bit 1, freq 0.727188
Bit 2, freq 0.924348
Bit 3, freq 0.228575
Bit 4, freq 0.364295
Bit 5, freq 0.497685
Bit 6, freq 0.338122
Bit 7, freq 0.457472
```

For any natural language text that has been permuted, the most probable permutation can easily be determined by analysing the bit frequencies in the result, and comparing the results with statistics similar to these:

```
$ permute 4 2 8 3 1 6 7 5 < peter.pan > peter.perm
```

```
$ crackperm peter.perm
```

Encryption key was: 4 2 8 3 1 6 7 5

Decryption key is: 5 2 4 1 8 6 7 3

Chapter 1 PETER BREAKS THROUGH

All children, except one, grow up. They soon know that they will grow up, and the way Wendy knew was this. One day when she was two years old she was playing in a garden, and she plucked another flower and ran with it to her mother. I suppose she must have looked rather delightful, for Mrs. Darling put her hand to her heart and cried,
ctrl-C

```
$ bitfreq <peter.pan
```

Bit 0 is MSB, 7 is LSB, proportion of 1's:

Bit 0, freq 0.000000

Bit 1, freq 0.744555

Bit 2, freq 0.952047

Bit 3, freq 0.239387

Bit 4, freq 0.337803

Bit 5, freq 0.459206

Bit 6, freq 0.335522

Bit 7, freq 0.425811

```
$ permute 4 2 8 3 1 6 7 5 <peter.pan >peter.perm
```

```
$ bitfreq <peter.perm
```

Bit 0 is MSB, 7 is LSB, proportion of 1's:

Bit 0, freq 0.239387

Bit 1, freq 0.744555

Bit 2, freq 0.425811

Bit 3, freq 0.952047

Bit 4, freq 0.000000

Bit 5, freq 0.459206

Bit 6, freq 0.335522

Bit 7, freq 0.337803

```
$ xor A5966A59 <peter.pan >peter.xor
```

```
$ bitfreq <peter.xor
```

Bit 0 is MSB, 7 is LSB, proportion of 1's:

Bit 0, freq 0.500000

Bit 1, freq 0.499585

Bit 2, freq 0.500202

Bit 3, freq 0.501706

Bit 4, freq 0.499608

Bit 5, freq 0.500659

Bit 6, freq 0.500632

Bit 7, freq 0.498503

XOR and Permutation alone are very insecure, but used together each can cover up the weaknesses of the other.

First experiment: XOR first (already done) then Permute.

```
$ permute 4 2 8 3 1 6 7 5 <peter.xor> peter.xorperm
```

```
$ crackperm peter.xorperm
```

Encryption key was: 3 4 1 8 5 2 6 7
Decryption key is: 3 6 1 2 5 7 8 4

```
<C8>m^V<B3><C8>m^V<B3><C8>m^V&<DA><FD>&<C3>^XY^V^S<C8>m^V<82>^Y^<C7><86><C8>x#bY<FA><A3><B3><B9>~#d9<B8>^E<B4><CF>m^V<9A>?^V'<DA><FF>D<E3><FC><BD>@<F1><C8><BD>$'X] f<B3>^^; <C6><F1><C8><B9>"e<m<E6><83><8E>m^V<C2><DA><BD><A4><B3>|<BB><C0><E5><C8><FB>@e<mf<A1>X^] ^V<C3><DA><BD><A4><B3><<FF>D<E1><CF><B9>"e<m<E6><83><8A>m<86><E5><98>mf
```

ctrl-C

It is not relevant that the results of the attempted decryption were unreadable, because it had also been exclusive-ored, so just reversing the permutation would not result in readable text. The important thing is that it worked out completely the wrong de-permutation key, so the results it got are just plain wrong.

Second experiment: Permute first (already done) then XOR.

```
$ xor A5966A59 <peter.perm> peter.permxor
```

```
$ crackxor <peter.permxor>
```

It looks to me as though the key length is 4 bytes

The key is E6040000

And here are the first 10 lines of the file:

```
S<82>zIS<82>zIS<82>z;^R<E2><BA><8D>7@z<E9>S<82>z<99>'V^N<9B>S<D0><A8>=#<F1><88>I<87><D3><A8>><A7><F4>+Z@<82>z9^V<C7>z+^R<E3>?^M<91><E6>=LS<E6><BB>+7B<BE>I4<C5>^^LS<E4><B8>.<B5><82><9E><89>T<82>z<9D>^R<E6><9B>I<B1><E5>^] ^NS<E1>=.<B5><82><BE3Fz<8D>^R<E6><9B>I<B5><E3>?^L@<E4><B8>.<B5><82><9E><89>V<82>^Z^N^W<82><BE7<82><9C>)<B2><82><8C>-^T<C6><9B>I0<C5>^^<AF>Sd^Z<AB>SF; (<B1><85>zI$<C5>^^I^W<E2><9B>I<B5><C3>^^^NS`;-Sd^Z<AB>SF<9C>.@c^^)<91>`z.^V<C6>z<AB>^R<E6>z<AF>3`z<89>^V<E2><9B>(^T<E4>z(^T<82>^ZI5<E2>
```

ctrl-C

And the results are just as bad (or good, depending on your point of view).

So DES, which interleaves 16 rounds of permutation and exclusive or, may be on to a good thing.