

One Way Hash

```
$ cat filehash.cpp
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char * argv[])
{ if (argc!=2)
  { printf("Give me a file name\n");
    exit(1); }
  FILE * fin=fopen(argv[1], "r");
  if (fin==NULL)
  { printf("Can't read '%s'\n", argv[1]);
    exit(1); }

const long long int multiplier=0x59BE60F376CA3D71LL;
const long long int divisor=0x4B8C64D76CBA135DLL;
long long int hash=0x2374A6FE07165785LL;

while (1)
{ int c=fgetc(fin);
  if (c==EOF) break;
  hash = hash * multiplier + c; }
if (hash<0) hash=-hash;
hash%=divisor;
printf("hash is %qd\n", hash); }
```

```
$ diff sample1.txt sample2.txt
16c16
< long-gone days at Valhalla State, and now sat glowering
---
> long-gone days at valhalla State, and now sat glowering
```

```
$ filehash sample1.txt
hash is 2817325362030363725
```

```
$ filehash sample2.txt
hash is 351733015509609875
```

```
$ wc sample1.txt
27      184     1083 sample1.txt
```

A Fingerprint for a file. Just record the hash for sensitive files, and you can tell if any illicit changes have been made. Of course it does not protect the file.

The hash does not need to be kept secret, as the file can not be reconstructed from it.

Modular Arithmetic

$$(A + B) \% n = (A\%n + B\%n) \% n$$

$$(A - B) \% n = (A\%n - B\%n) \% n$$

$$(A \times B) \% n = (A\%n \times B\%n) \% n$$

Multiplication table modulo 7

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Can see that $2 \times 4 = 1$, $3 \times 5 = 1$, $4 \times 2 = 1$ $5 \times 3 = 1$, $6 \times 6 = 1$,

Which means that multiplying by 4 is the same as dividing by 2,
multiplying by 5 is the same as dividing by 3, etc.

3 is the "modular inverse" of 5, modulo 7

4 is the "modular inverse" of 2, modulo 7

So modulo 7, division can be done meaningfully.

This always works out when the modulus is prime.

Multiplication table modulo 6

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

Can see that nothing $\times 4 = 1$,

Which means that division modulo 6 can not be done.

To The Power Of

Think of a binary number: 1001101. This is 77.

It tells us the $77 = 64 + 8 + 4 + 1$.

And makes it easy to work out anything to the power of 77.

$$\begin{aligned} A^{77} &= A^{64+8+4+1} \\ &= A^{64} \times A^8 \times A^4 \times A^1 \\ &= A^1 \times A^4 \times A^8 \times A^{64} \\ &= A \times (A^2)^2 \times ((A^2)^2)^2 \times (((((A^2)^2)^2)^2)^2)^2 \end{aligned}$$

Run a loop, looking at each digit of the exponent in turn, also squaring that value of A each time round. For any 1 in the binary for the exponent, multiply the answer so far by the A so far.

```
int power(int A, int B)
{ int answer=1;
  while (B>0)
  { if (B & 1)
      answer*=A;
    A*=A;
    B>>=1; }
  return answer; }
```

A large number to the power of a large number is a really huge number, and would be very difficult to compute. But if it is all done modulo N, the answer and all intermediate results will be less than N.

$(1263182^{3571532}) \% 1000000$ can be computed very quickly and easily, and only has six digits.

One-Time-Pad Encryption 2003-11-25

\$ cat generatepad.cpp

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

void main(int argc, char *argv[])
{ if (argc!=2)
  { fprintf(stderr, "Need name of key file on command line\n");
    exit(1); }
  FILE *f=fopen(argv[1], "w");
  if (f==NULL)
  { fprintf(stderr, "Can't create key file '%s'\n", argv[1]);
    exit(1); }
  srandrandom();
  for (int i=0; i<1000; i+=1)
  { int r=random()&0xFF;
    fputc(r, f);
  }
  fclose(f); }
```

\$ cat onetimepad.cpp

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

typedef unsigned char byte;

byte key[1000];

void main(int argc, char *argv[])
{ if (argc!=2)
  { fprintf(stderr, "Need name of key file on command line\n");
    exit(1); }
  FILE *f=fopen(argv[1], "r");
  if (f==NULL)
  { fprintf(stderr, "Can't open key file '%s'\n", argv[1]);
    exit(1); }
  for (int i=0; 1; i+=1)
  { int c=fgetc(f);
    if (c==EOF) break;
    key[i]=c; }
  fclose(f);
  for (int i=0; 1; i+=1)
  { int c=getchar();
    if (c==EOF) break;
    putchar(c^key[i]); } }
```

```
$ generatepad pad1
```

```
$ more pad1
```

```
<F4><DE>'^Y1<CA><A8><A9><88><FB><9F>;H<E6><93><90><8D><F1>ESC<C0>
|e<A0>/<8C>0m<E0><86>K} {} <A4><94>Zn=^D<F6>8<A3>1<80><89><C4>^P^W
<B5>+<D7>2<90>wa^<A7><CE><FD>.^Yz<A9>C^<9D><8C>z<A1><82><B2>E
<B4>2<CE>xB<E5>.m<BC>`<FD>3<C1>^Z<DB><8F>^W <A9><91><B2><EC><AF>
<EF><89>;j+<BD>^<pqO><EA><91>$^X<FF><E0>x<FC>^T9^V<EF><C8>-
<F8>q<BE><AA>]m<99><E7><A8>^C^Rf <82><D7>o<C0><C1>^@<E4><D9><FF><C5>Q<FC>
^C[ctrl-C]
```

```
$ onetimepad pad1 <plain >secret
```

```
$ cat secret
```

```
»°B9E½C‰ü"í^-Æöýøf; |ÅýYêò#[JÅàz\pÖWÍôá;0zÔ_ù8ä<Ö"@"9çcMMüåÅäÓ)ØAî#Œ@Å@"]án³ê7yÅðÛ,ÜÁf
```

```
$ onetimepad pad1 <secret
```

```
One two three four five six
```

```
the cat sat on the mat.
```

```
the rain in Spain falls mainly in the plains.
```

```
$ cat realmessage.txt
```

```
Kill the president using a sharp stick on Tuesday night.
```

```
$ cat fakemessage.txt
```

```
Hey! Buy me a cake, and shampoo my car tomorrow, please.
```

```
$ onetimepad pad1 <realmessage.txt >encrypted
```

```
$ cat encrypted
```

```
¿·Ku¾ÀÌ`<í^;□÷ðä...;µ
```

```
ÍH¬QM"í*
```

```
V$™Vfeði·tvÍ
```

```
×à3
```

```
¹ [÷2-Å
```

```
$ onetimepad pad1 <encrypted
```

```
Kill the president using a sharp stick on Tuesday night.
```

```
$ onetimepad encrypted < fakemessage.txt >fakepad
```

```
$ onetimepad fakepad <encrypted
```

```
Hey! Buy me a cake, and shampoo my car tomorrow, please.
```

\$ cat xor.cpp

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

typedef unsigned char byte;

int hex(char c)
{ if (c>='0' && c<='9') return c-'0';
  if (c>='A' && c<='F') return c-'A'+10;
  if (c>='a' && c<='f') return c-'a'+10;
  fprintf(stderr, "%c is not a hexadecimal digit\n", c);
  exit(1); }

void main(int argc, char *argv[])
{ if (argc!=2)
  { fprintf(stderr, "Need hexadecimal key on command line\n");
    exit(1); }
  char * code=argv[1];
  int codelen=strlen(code);
  int keylen=(codelen+1)/2;
  byte *key=new byte[keylen];
  int start=0;
  if (codelen%2==1)
  { key[0]=hex(code[0]);
    start=1; }
  for (int i=start, j=start; j<codelen; i+=1, j+=2)
    key[i]=(hex(code[j])<<4)+hex(code[j+1]);
  for (int i=0, k=0; 1; i+=1, k+=1)
  { if (k>=keylen) k=0;
    int c=getchar();
    if (c==EOF) break;
    putchar(c^key[k]); } }
```

\$ cat plain

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

\$ xor 6E1AB5 <plain >cypher

\$ cat cypher

!tÐNnÂ:ÁhÐ :ÓoÇN|Ü•sídnÝ
 :Ön•{ÁNuÛNnÝ
 :Øn>dnÝ
 :ÇsÛNsÛNIÅsÛN|ÔvÆNwÔtÙ:Ü:Á•vÔtÆ@

```

$ cat xortwo.cpp

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int same(unsigned char a[], unsigned char b[], int max)
{ if (max>100) max=100;
  for (int i=0; i<max; i+=1)
    if (a[i]!=b[i])
      return 0;
  return 1; }

void main(int argc, char *argv[])
{ if (argc!=3)
  { fprintf(stderr, "Need two filenames on command line\n"); exit(1); }
  FILE *f1=fopen(argv[1], "r");
  if (f1==NULL)
  { fprintf(stderr, "could not open '%s'\n", argv[1]); exit(1); }
  FILE *f2=fopen(argv[2], "r");
  if (f2==NULL)
  { fprintf(stderr, "could not open '%s'\n", argv[2]); exit(1); }
  unsigned char codes[1009];
  int online=0, total=0;
  while (1)
  { int c1=fgetc(f1), c2=fgetc(f2);
    if (c1==EOF && c2==EOF) break;
    if (c1==EOF || c2==EOF)
    { fprintf(stderr, "\nThe two files are of different lengths\n"); exit(1); }
    printf("%02X", c1^c2);
    codes[total]=c1^c2;
    total+=1;
    if (total>1000) break;
    online+=2;
    if (online>60)
    { printf("\n");
      online=0; } }
  printf("\n\n");
  fclose(f1); fclose(f2);
  int keylen=0;
  for (int i=1; i<100; i+=1)
    if (same(codes, codes+i, total-i-1))
    { keylen=i;
      break; }
  if (keylen==0)
    printf("Could not find a key\n");
  else
  { printf("The key is ");
    for (int i=0; i<keylen; i+=1)
      printf("%02X", codes[i]);
    printf("\n"); } }

$ xortwo plain cypher
6E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E
1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1A
B56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB56E1AB5
6E1AB56E1A

```

The key is 6E1AB5

```
$ cat crackxor1.cpp
```

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

struct string
{ unsigned char * str;
  int len, size;
  string(void)
  { str=NULL; len=0; size=0; }
~string(void)
  { delete[] str; }
void grow(void)
  { int newsz=size*2;
    if (newsz<16) newsz=16;
    unsigned char * newstr = new unsigned char[newsz];
    for (int i=0; i<size; i+=1)
      newstr[i]=str[i];
    delete[] str;
    str=newstr;
    size=newsz; }
void add(unsigned char c)
  { if (len>=size) grow();
    str[len]=c;
    len+=1; }
unsigned char & operator[](int i)
  { return str[i]; } };

void main(void)
{ string s;
  while (1)
  { int c=getchar();
    if (c==EOF) break;
    s.add(c); }
double zerofreq[100], sumzf, totzf, maxzf=-1;
printf("          1/256=%10.6f %%\n", 100.0/256.0);
for (int keylen=2; keylen<25; keylen+=1)
{ int zeros=0, total=0;
  for (int i=0, j=keylen; j<s.len; i+=1, j+=1)
  { int c=s[i]^s[j];
    if (c==0)
      zeros+=1;
    total+=1; }
  double zf=100.0*zeros/total;
  zerofreq[keylen]=zf;
  sumzf+=zf;
  if (zf>maxzf) maxzf=zf;
  totzf+=1;
  printf("key length %2d gives%10.6f %% zeros\n", keylen, zf); }
int keylen=0;
for (int i=2; i<25; i+=1)
  if (zerofreq[i]>maxzf/2)
  { keylen=i;
    break; }
if (keylen==0)
  printf("I couldn't work out the key length\n");
else
  printf("It looks to me as though the key length is %d bytes\n", keylen); }
```

```
$ cat peter.pan
```

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

^C[Ctrl-C]

```
$ xor 93E71BA573E08F15 <peter.pan >coded.pan
```

```
$ cat coded.pan
```

```
<B3><C7>;<85>S<C0><AF>5<B3><C7>;<E6>ESC<81><FF>a<F6><95>;<94>S<C0>
<AF>E<D6><B3>^<F7>S<A2><DD>P<D2><AC>H<85>'<A8><DD>Z<C6><A0>S<AF>y
<C0><AF>T<FF><8B>;<C6>ESC<89><E3>q<E1><82>u<89>S<85><F7>v<F6><97>
o<85>^<8E><EA>9<B3><80>i<CA>^D<C0><FA>e<BD><C7>;<F1>ESC<85><F6>5
<E0><88>t<CB>S<8B><E1>z<E4><C7>o<CD>^R<94><AF>a<FB><82>b<85>^D<89>
<E3>y<99><80>i<CA>^D<C0><FA>e<BF><C7>z<CB>^W<C0><FB>}<F6><C7>1<C4>
<C0><D8>p<FD><83>b<85>^X<8E><EA>b<B3><90>z<D6>S<94><E7>|<E0><C9>;
<85><<8E><EA>5<F7><86>b<85>^D<88><EA>{<B3><94>s<C0>S<97><EE>f<B3>
```

^C[Ctrl-C]

```
$ crackxor1 <coded.pan
```

key length	1/256	=	0.390625 %
key length	2	gives	0.888770 % zeros
key length	3	gives	0.386506 % zeros
key length	4	gives	0.732270 % zeros
key length	5	gives	0.379655 % zeros
key length	6	gives	0.884214 % zeros
key length	7	gives	0.066640 % zeros
key length	8	gives	7.264170 % zeros
key length	9	gives	0.073114 % zeros
key length	10	gives	0.818729 % zeros
key length	11	gives	0.426122 % zeros
key length	12	gives	0.586443 % zeros
key length	13	gives	0.411273 % zeros
key length	14	gives	0.857965 % zeros
key length	15	gives	0.069308 % zeros
key length	16	gives	7.161191 % zeros
key length	17	gives	0.073497 % zeros
key length	18	gives	0.839318 % zeros
key length	19	gives	0.400239 % zeros
key length	20	gives	0.634063 % zeros
key length	21	gives	0.405193 % zeros
key length	22	gives	0.838570 % zeros
key length	23	gives	0.077307 % zeros
key length	24	gives	7.068107 % zeros

It looks to me as though the key length is 8 bytes

```
$ cat crackxor2.cpp
```

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

struct string
{ unsigned char * str;
  int len, size;
  string(void)
  { str=NULL; len=0; size=0; }
~string(void)
  { delete[] str; }
void grow(void)
{ int newsz=size*2;
  if (newsz<16) newsz=16;
  unsigned char * newstr = new unsigned char[newsz];
  for (int i=0; i<size; i+=1)
    newstr[i]=str[i];
  delete[] str;
  str=newstr;
  size=newsz; }
void add(unsigned char c)
{ if (len>=size) grow();
  str[len]=c;
  len+=1; }
unsigned char & operator[](int i)
{ return str[i]; } };

void main(int argc, char *argv[])
{ if (argc!=2)
  { fprintf(stderr, "Need key length on the command line\n");
    exit(1); }
  string s;
  while (1)
  { int c=getchar();
    if (c==EOF) break;
    s.add(c); }
  int keylen=atol(argv[1]);
  printf(" \\\0 \\\1 \\\23 \\\n spc e t s q x\n");
  for (int k=0; k<256; k+=1)
  { int counts[256];
    for (int i=0; i<256; i+=1) counts[i]=0;
    int total=0;
    for (int i=0; i<s.len; i+=keylen)
    { int c=s[i]^k;
      counts[c]+=1;
      total+=1; }
    printf("k=%02X:", k);
    printf(" %4.0f", 10000.0*counts[0]/total);
    printf(" %4.0f", 10000.0*counts[1]/total);
    printf(" %4.0f", 10000.0*counts[23]/total);
    printf(" %4.0f", 10000.0*counts['\n']/total);
    printf(" %4.0f", 10000.0*counts[' ']/total);
    printf(" %4.0f", 10000.0*counts['e']/total);
    printf(" %4.0f", 10000.0*counts['t']/total);
    printf(" %4.0f", 10000.0*counts['s']/total);
    printf(" %4.0f", 10000.0*counts['q']/total);
    printf(" %4.0f", 10000.0*counts['x']/total);
    if (counts[0]==0 && counts[1]==0 && counts[23]==0 && counts[' ']>total/10)
      printf(" YES ***");
    printf("\n"); } }
```

```
$ crackxor2 8 <coded.pan
```

	\0	\1	\23	\n	spc	e	t	s	q	x
k=00:	0	0	0	0	0	0	0	0	0	0
k=01:	0	0	0	0	0	0	0	0	0	0
k=02:	0	0	0	0	0	0	0	0	0	0
k=03:	0	0	0	0	0	0	0	0	0	0
k=04:	0	0	0	0	0	0	0	0	0	0
k=05:	0	0	0	0	0	0	0	0	0	0
k=06:	0	0	0	0	0	0	0	0	0	0
k=07:	0	0	0	0	0	0	0	0	0	0
k=08:	0	0	0	0	0	0	0	0	0	0
k=09:	0	0	0	0	0	0	0	0	0	0
k=0A:	0	0	0	0	0	0	0	0	0	0
k=0B:	0	0	0	0	0	0	0	0	0	0
....:
....:
k=8F:	0	0	0	0	0	171	524	542	161	340
k=90:	0	0	0	0	0	138	190	99	413	0
k=91:	0	0	0	0	111	163	57	7	468	4
k=92:	0	0	0	0	8	340	201	413	99	171
k=93:	0	0	0	251	1843	946	676	468	7	7 YES ***
k=94:	0	0	0	0	21	105	468	676	57	0
k=95:	0	0	0	0	0	140	413	201	190	0
k=96:	0	0	0	0	0	1	7	57	676	0
k=97:	0	0	0	0	0	566	99	190	201	0
k=98:	0	251	0	0	0	478	0	7	4	468
....:
....:
k=F9:	6	73	0	1	8	0	0	0	0	0
k=FA:	457	524	0	140	30	0	0	0	0	0
k=FB:	524	457	0	105	13	0	0	0	0	0
k=FC:	542	478	7	946	8	251	0	0	0	0
k=FD:	478	542	171	340	11	0	0	0	0	0
k=FE:	161	294	4	163	10	0	0	0	0	0
k=FF:	294	161	0	138	3	0	0	0	0	0

```
$ crackxor < coded.pan
```

It looks to me as though the key length is 8 bytes

Key byte 0 is 93
Key byte 1 is E7
Key byte 2 is 1B
Key byte 3 is A5
Key byte 4 is 73
Key byte 5 is E0
Key byte 6 is 8F
Key byte 7 is 15

The key is 93E71BA573E08F15

And here are the first 10 lines of the file:

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, "Oh, why can't you remain like this for ever!" This was all that passed between them on the subject, but henceforth Wendy knew that she must grow up. You always know after you are two. Two is the

Experiments supporting XOR cracking

Terms

“Plaintext”: Human readable.

“Key”: Secret pattern used to encrypt or decrypt or both.

“Ciphertext”: The result of encrypting plaintext.

Assume plaintext is the sequence of characters ($p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, \dots$) and makes sense in some natural language, so the distribution of characters is definitely not even. (e.g. E more common than Q).

Key is the sequence of bytes (k_1, k_2, k_3, k_4) with no discernible pattern, so each k_i is essentially a sequence of 8 random bits.

So using XOR encryption, the ciphertext is ($p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, p_{10} \oplus k_2, \dots$)

Ciphertext XORed with itself delayed by one byte:

$$(p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, p_{10} \oplus k_2, \dots)$$

$$\oplus (0, p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, p_{10} \oplus k_2, \dots)$$

$$= (p_1 \oplus k_1, p_2 \oplus k_2 \oplus p_1 \oplus k_1, p_3 \oplus k_3 \oplus p_2 \oplus k_2, p_4 \oplus k_4 \oplus p_3 \oplus k_3, p_5 \oplus k_1 \oplus p_4 \oplus k_4, p_6 \oplus k_2 \oplus p_5 \oplus k_1, p_7 \oplus k_3 \oplus p_6 \oplus k_2, \dots)$$

which is nothing special.

Because the k_i are completely random, anything XORed with them is also completely random, so the result is just a string of randomly distributed bytes.

Ciphertext XORed with itself delayed by two bytes:

$$(p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, p_{10} \oplus k_2, \dots)$$

$$\oplus (0, 0, 0, p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, p_{10} \oplus k_2, \dots)$$

$$= (p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3 \oplus p_1 \oplus k_1, p_4 \oplus k_4 \oplus p_2 \oplus k_2, p_5 \oplus k_1 \oplus p_3 \oplus k_3, p_6 \oplus k_2 \oplus p_4 \oplus k_4, p_7 \oplus k_3 \oplus p_5 \oplus k_1, \dots)$$

the same as above.

Ciphertext XORed with itself delayed by three bytes:

$$(p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, p_{10} \oplus k_2, \dots)$$

$$\oplus (0, 0, 0, 0, p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, \dots)$$

$$= (p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4 \oplus p_1 \oplus k_1, p_5 \oplus k_1 \oplus p_2 \oplus k_2, p_6 \oplus k_2 \oplus p_3 \oplus k_3, p_7 \oplus k_3 \oplus p_4 \oplus k_4, p_8 \oplus k_4 \oplus p_5 \oplus k_1, \dots)$$

the same as above.

Ciphertext XORed with itself delayed by four bytes:

$$(p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, p_9 \oplus k_1, p_{10} \oplus k_2, \dots)$$

$$\oplus (0, 0, 0, 0, 0, p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1, p_6 \oplus k_2, p_7 \oplus k_3, p_8 \oplus k_4, \dots)$$

$$= (p_1 \oplus k_1, p_2 \oplus k_2, p_3 \oplus k_3, p_4 \oplus k_4, p_5 \oplus k_1 \oplus p_1 \oplus k_1, p_6 \oplus k_2 \oplus p_2 \oplus k_2, p_7 \oplus k_3 \oplus p_3 \oplus k_3, p_8 \oplus k_4 \oplus p_4 \oplus k_4, \dots)$$

$$= (\dots, \dots, \dots, \dots, p_5 \oplus p_1, p_6 \oplus p_2, p_7 \oplus p_3, p_8 \oplus p_4, p_9 \oplus p_5, p_{10} \oplus p_6, \dots)$$

Which is something else.

Because the $k_i \oplus k_i$ components cancel themselves out, the result is just the exclusive OR of plaintext characters with other plaintext characters. Although they are still incomprehensible, the result bytes will no longer be randomly distributed. The probability that two p_i are equal will be much higher than the probability that some p_i is equal to a random k_i .

Equal things exclusive-ored together produce zero, so the last result (when the delay is equal to the key length) will have an unusually high proportion of zeros in it. Just keep trying out different numbers of bytes of delay; when there is an inordinate number of zeros in the result, you have discovered the key length.

\$ freqtest 4

We are pretending that there are only 4 letters
in the alphabet, A (ascii code 0) to D (ascii code 3)

Relative frequency of A = .49

Relative frequency of B = .49

Relative frequency of C = .01

Relative frequency of D = 0.010000

Two random strings from the language XORed:

BAAAABBABAABBABAAAABBBBBBABBABAAABCABAABAABABAABADAABBBAABBBBA...
BBAABABAABABABBBAAABBAABBAABBADABABDBBBABBBBBBABABBBABABBBAAA...

0?00??????00??0?00000?????0???0?0???00?0?00?0?000?00?????00?00?00?0...

(Question-marks represent any non-zero result)

Proportion of 0s in result string = 0.467000

One random string from the language and one evenly distributed string

ABABABBACBBABAAABBAABBAAAAABBBBABBAAABAAABABAABBBDADABBBABAAABA...
DBCCBCDCDCBACDDADACCDABDACCABDACBDDACBDADDCCDCBDBDCBBCBDA...

?0??????00???0?????0?0?0?000?0?0?0?0?0?0?????00?????0?0?????00...

(Question-marks represent any non-zero result)

Proportion of 0s in result string = 0.248000

(i.e. approximately 0.25, exactly what would be expected from a random distribution of values)

Table of Exclusive-Or results and probabilities for random strings from the language

	p=0.49 A = 00	p=0.49 B = 01	p=0.01 C = 10	p=0.01 D = 11
p=0.49 A = 00	p=0.2401 00	p=0.2401 01	p=0.0049 10	p=0.0049 11
p=0.49 B = 01	p=0.2401 01	p=0.2401 00	p=0.0049 11	p=0.0049 10
p=0.01 C = 10	p=0.0049 10	p=0.0049 11	p=0.0001 00	p=0.0001 01
p=0.01 D = 11	p=0.0049 11	p=0.0049 10	p=0.0001 01	p=0.0001 00

Total probability of 00 being the result of an exclusive or =

$$0.2401+0.2401+0.0001+0.0001 = 0.4804 \\ \text{pure random} = 0.2500$$

Total probability of 01 being the result of an exclusive or =

$$0.2401+0.2401+0.0001+0.0001 = 0.4804$$

Total probability of 10 being the result of an exclusive or =

$$0.0049+0.0049+0.0049+0.0049 = 0.0196$$

Total probability of 11 being the result of an exclusive or =

$$0.0049+0.0049+0.0049+0.0049 = 0.0196$$

\$ freqtest 6

We are pretending that there are only 6 letters in the alphabet, A (ascii code 0) to F (ascii code 5)

Relative frequency of A = .42

Relative frequency of B = .42

Relative frequency of C = .07

Relative frequency of D = .07

Relative frequency of E = .01

Relative frequency of F = 0.010000

Two random strings from the language XORed:

ABBBABBBABDAAABBBBBCABA
FDBAADB BBBB BABB AFBAAAABBBBAABAAAABABA
D BCA ABC AA . . .

BBBFABBAEAABBCBABBACBCABBAABDBAABABAABAAADDCAADDDBABAACBAABADBBAAABB... .

?00?0000??0??00?0?0?????0?0?0?0?0?????0?0?00?0??00?0?00?00000?00?????...

(Question-marks represent any non-zero result)

Proportion of 0s in result string = 0.342000

One random string from the language and one evenly distributed string

BAAAAAAAAAAADBBCABAABAABFBDAABABCBCABAADAAABCBCBABADDDBDABABABAACAAEC.....

FADADEDEDCEDEFGCBADDAEAECCDEEADEEBCCFAFGDABACAAACACEFADDCDAEAFFEDAAADDAAEF.

(Question-marks represent any non-zero result)

Proportion of 0s in result string = 0.170000

(i.e. approximately 0.1666, exactly what would be expected from a random distribution of values)

Table of Exclusive-Or results and probabilities for random strings from the language

	p=0.42 A = 000	p=0.42 B = 001	p=0.07 C = 010	p=0.07 D = 011	p=0.01 E = 100	p=0.01 F = 101
p=0.42 A = 000	p=0.1764 000	p=0.1764 001	p=0.0294 010	p=0.0294 011	p=0.0042 100	p=0.0042 101
p=0.42 B = 001	p=0.1764 001	p=0.1764 000	p=0.0294 011	p=0.0294 010	p=0.0042 101	p=0.0042 100
p=0.07 C = 010	p=0.0294 010	p=0.0294 011	p=0.0049 000	p=0.0049 001	p=0.0007 110	p=0.0007 111
p=0.07 D = 011	p=0.0294 011	p=0.0294 010	p=0.0049 001	p=0.0049 000	p=0.0007 111	p=0.0007 110
p=0.01 E = 100	p=0.0042 100	p=0.0042 101	p=0.0007 110	p=0.0007 111	p=0.0001 000	p=0.0001 001
p=0.01 F = 101	p=0.0042 101	p=0.0042 100	p=0.0007 111	p=0.0007 110	p=0.0001 001	p=0.0001 000

Total probability of 000 being the result of an exclusive or =

$$0.1764+0.1764+0.0049+0.0049+0.0001+0.0001 = 0.3628$$

pure random = 0.1667

```
$ cat frequency.cpp // measuring relative frequencies of characters in text
#include <stdio.h>
#include <stdlib.h>

char *charname(int c)
{ static char s[10];
  if (c=='\n') return "\n";
  if (c<' ')
  { sprintf(s, "ctrl-%c", c+64);
    return s; }
  if (c==' ') return "space";
  if (c<127)
  { s[0]=c;
    s[1]=0;
    return s; }
  if (c==127) return "DEL";
  sprintf(s, "[%d]", c);
  return s; }

void main(void)
{ int count[256], ordered[256], taken[256];
  for (int i=0; i<256; i+=1)
  { count[i]=0;
    taken[i]=0; }
  int total=0;
  while (1)
  { int c=getchar();
    if (c==EOF) break;
    count[c]+=1;
    total+=1; }
  for (int i=0; i<256; i+=1)
  { int bigpos=-1, bigcnt=-1;
    for (int j=0; j<256; j+=1)
      if (!taken[j] && count[j]>bigcnt)
      { bigcnt=count[j];
        bigpos=j; }
    ordered[i]=bigpos;
    taken[bigpos]=1; }
  double sumsq=0.0;
  for (int i=0; i<256; i+=1)
  { int j=ordered[i];
    if (count[j]>0)
    { double relfreq=count[j]/(double)total;
      sumsq+=relfreq*relfreq;
      printf("%6s: %.6f\n", charname(j), relfreq); } }
  printf("Prob. of two random chars being the same = %.6f\n", sumsq);
  printf("1/256 = %.6f\n", 1.0/256.0); }
```

\$ frequency <peter.pan

space: 0.186214
e: 0.095506
t: 0.067362
a: 0.056471
o: 0.053265
h: 0.053074
n: 0.048296
i: 0.046060
s: 0.044423
r: 0.040307
d: 0.034199
l: 0.030315
\n: 0.025281
u: 0.020486
w: 0.019165
y: 0.017166
g: 0.015407
m: 0.015323
c: 0.015011
,: 0.014603
f: 0.013541
": 0.011111
b: 0.010944
.: 0.010582
p: 0.009809
k: 0.007532
v: 0.005803
I: 0.002917
T: 0.002742
W: 0.002239
' : 0.002132
P: 0.001828
H: 0.001778
S: 0.001565
;: 0.001477
-: 0.001371
?: 0.001211
M: 0.001207
N: 0.001116
A: 0.001001
!: 0.000887
x: 0.000777
O: 0.000765
J: 0.000743
D: 0.000735
B: 0.000708
q: 0.000655
j: 0.000564
.....
.....
9: 0.000004
Z: 0.000004
^: 0.000004

Prob. of two random chars being the same = 0.070724

1/256 = 0.003906