RC4

Solves the problems that make XOR encryption easily crackable by producing a random stream of bytes (based on the key of course), and exclusive-oring the bytes of the message with those random bytes. RC4 is essentially a random number generator; the random numbers are the real encryption key, and the user's key is the seed to the random number generator. I have seen no reports of anyone having made any progress towards cracking RC4.

```
#include <stdio.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;
  /* etc etc etc */ }

byte * stringtohex(char *code)
{ int codelen=strlen(code);
  int keylen=(codelen+1)/2;
  byte *key=new byte[keylen+1];
  key[0]=keylen;
  key+=1;
  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]);
  return key-1; }

int S[256], rc4i, rc4j;

void swap(int & x, int & y)
{ int t=x; x=y; y=t; }

int reset(byte key[])
{ int len=key[0];
  key+=1;
  int K[256];
  for (int i=0, j=0; i<256; i+=1, j+=1)
  { S[i]=i;
    if (j>=len) j=0;
    K[i]=key[j]; }
  int j=0;
  for (int i=0; i<256; i+=1)
  { j=(j+S[i]+K[i])%256;
    swap(S[i], S[j]); }
  rc4i=0;
  rc4j=0; }

byte rc4(byte c)
{ rc4i=(rc4i+1)%256;
  rc4j=(rc4j+S[rc4i])%256;
  swap(S[rc4i], S[rc4j]);
  int t=(S[rc4i]+S[rc4j])%256;
  return c^S[t]; }

void main(int argc, char *argv[])
{ if (argc!=2)
  { fprintf(stderr, "Need hexadecimal key on command line\n");
    exit(1); }
  byte * key = stringtohex(argv[1]);
  reset(key);
  while (1)
  { int c=getchar();
    if (c==EOF) break;
    putchar(rc4(c)); } }
```

The array S contains a permutation of all possible 8-bit values. The reset function uses the user's encryption key to initially jiggle the S array around. The user's key can be any length up to 256 bytes (or 2048 bits, which is pretty good). Every character of plaintext is exclusive-ored with a byte from S, and that byte is then used to jiggle the S array even more.

This is a **symmetric algorithm**, meaning that encryption and decryption a both performed by the same algorithm using the same key. Convenient in some ways (when encrypting files, you only have to remember one password), but it means that you need a different password for every person you will ever want to have secure communications with.

It is also extremely fast.

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

$ rc4 74A5BE10F871 <plain >secret

$ cat secret
çk(uã'¾l/;eûU¸õ)€‡âžâ¦
YtÆø0öšßç®Ò^núƒ±%—»vdiÛÿMŠNÀF′"Ä=¾□fl4™Ã—'ÿäL£BÿU„ƒËÕ&6šõZóÇü!×î

$ rc4 74A5BE10F871 <secret
One two three four five six
the cat sat on the mat.
the rain in Spain falls mainly in the plains.

$ rc4 74A5BE10F872 < secret                    // just one bit of the key wrong, and it isn't even close.
Žžs‡ù!
      E□8Có□ÔÑ‰µüka»[l¡
íÆ¶èD$                          ‰Húm":1‰S2îØHÀ@äq**°†§•k³ŠAÊÃ'öä'DP˜;>Û[Lu$Fi*XËf·êíÔ4ù¥Ä

$ rc4 823B2E20385CC <peter.pan >secret
$ ls -l peter.pan secret
-rw-------  1 stephen  wheel  262612 Nov 25 03:07 peter.pan
-rw-------  1 stephen  wheel  262612 Dec  1 15:12 secret              // same length
```