SOLVING THE T2'13 CHALLENGE

by Alexander Polyakov

This year challenge begins with ZIP archive t213-challenge.zip containing disk image of the USB flash drive. 7-Zip opens the image as an archive just fine. All files in the archive have the same date and time: 26.10.2012 **13:37**. l33T! :)

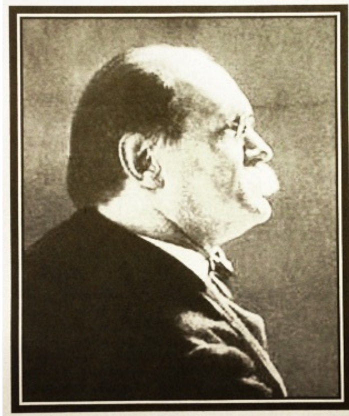
The filelist:

encryptor.c	Obfuscated C program. Resembles IOCCC contest entries.
file.mp3	Audio record of some low-rate data transfer.
news.txt	Posting to newsgroup.
photo1.jpg	Photo of two sheets of paper. On one sheet there is a photo of someone and four digits "1841". On the other sheet there is strange grid with numbers.
photo2.jpg	Again, photo of two sheets of paper and a tablet. There is a QR-code on the first sheet and a barcode on the second. The tablet partially obscures QR-code.
puzzle.html	The same strange grid with numbers which can be seen on the first photo.
vault/formatting.css vault/index.html vault/md5.js vault/vault.png	You enter some code on this page, then it redirects you somewhere.
data.torrent	This is a deleted file, so 7-Zip won't show it, but the file can be easily extracted by using any hex editor. FAT directory entry for this deleted file is located at offset 0x13f1c0 in apt.img. The file itself is located at 0x2c4000, size is 0xB05 bytes.

Looks like puzzles can be solved in any order.

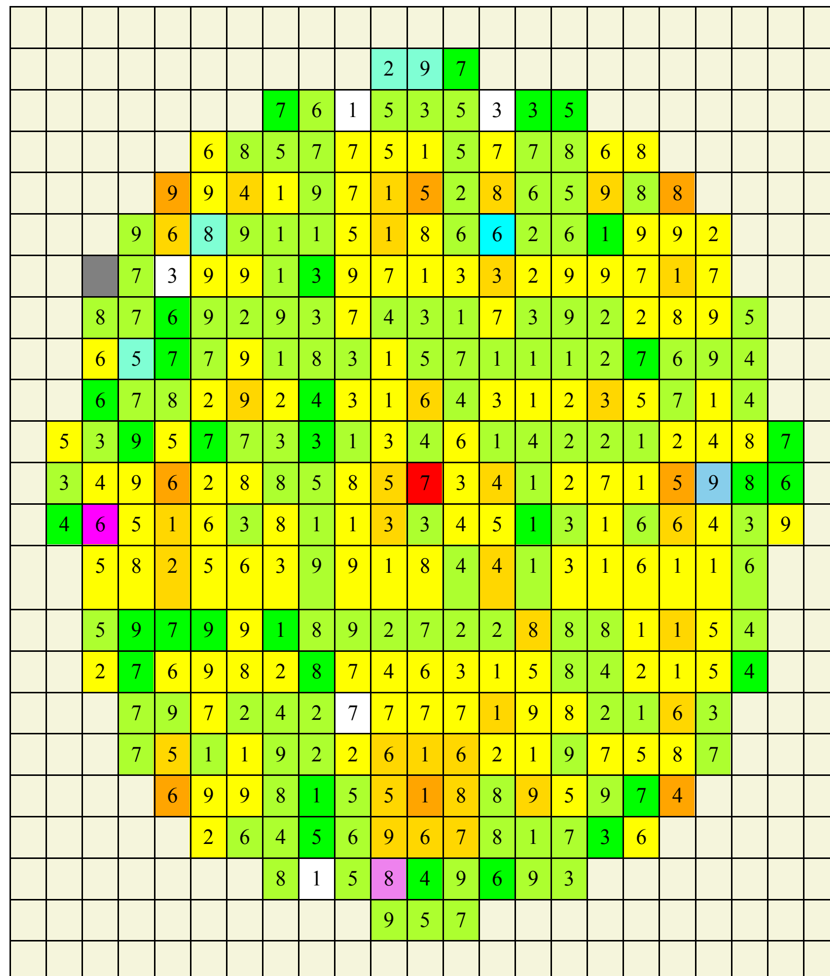
Level 1: photo01.jpg, puzzle.html, vault/*.*

Whenever there is a picture without a description, TinEye may be of use. (Hint: crop the picture before uploading - you'll get more results.)



Citing Wikipedia: "Samuel Loyd (January 30, **1841** - April 10, 1911)... was an American chess player, chess composer, **puzzle author**, and recreational mathematician." Further reading reveals that the cryptic grid is in fact one of Loyd's puzzles (but the numbers are, of course, different - so no copy-and-paste solution here!) It is called "Back from the Klondike", and it is a numerical maze. Note that vault/index.html also contains the word **"KLONDIKE"**.

To solve the puzzle I wrote simple program utilizing breadth-first search (see Appendix A). Here's the output:



And the solution is...

Color of each square indicates how many days it will take to reach it: red - 0 days (i.e. starting square), orange - one day, gold - two days, etc. Note that there are five unreachable (white) squares on the map. They contain numbers 1,3,3,7 and 1. L33T!

The Javascript code in vault/index.html internally represents north as 0, north-east as 1, east as 2 etc. So the solution can be also represented as 72357023570.

Once the solution is entered, the script calculates MD5 hash of the string, i.e. MD5("72357023570") == "297c5fc3b5af4ca1bd4a3287b9bf5960", then drops last two digits, prepends "http://tinyurl.com/" and redirects to http://tinyurl.com/297c5fc3b5af4ca1bd4a3287b9bf59, which in turn redirects to https://www.dropbox.com/s/qw1acg19qgru4uz/c321553877c582edc9435f97f5bcd7e7.jpg .

So the first MD5 code is **c321553877c582edc9435f97f5bcd7e7**.

Level 2: encryptor.c, data.torrent

First I defeated **#define**-based tricks using the compiler itself (many C compilers can output the code after preprocessing without actually compiling anything).

(I used Borland C++ free command-line tools, so the command was **cpp32 -P encryptor.c**. But other compilers can do that too, e.g. **gcc -E -P** or **cl.exe /E**).

Then I made the program more readable by using indent and made some tweaks by hand.

The result:

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>

unsigned long int
r (unsigned long int p)
{
  return 1664525 * p + 1013904223;
}

void
t (unsigned long int *p, unsigned long int a, unsigned long int c)
{
  p[a ^ 1] += (p[a] << 4) + p[c] ^ p[a] + p[1] ^ (p[a] >> 5) + p[c + 1];
} int

main (void)
{
  unsigned long int k[10];
  k[5] = r (k[4] = r ((sleep (1), k[3] = r (k[2] = r ((k[1] = r (k[0] = r (0x083e5342)), time (0)))))));
  do
    {
      k[7] = k[6] = 0;
      if ((k[8] = read (0, k + 6, 8)) == 0)
        break;
      for (k[9] = k[1] = 0; k[9] < 32; k[9]++)
        {
          k[1] += k[0];
          t (k, 7, 2);
          t (k, 6, 4);
        }
      write (1, k + 6, 8);
    }
  while (k[8] == 8);
  fprintf (stderr, "Your key is %08x %08x %08x %08x.\n", k[2], k[3], k[4], k[5]);
  return 0;
}
```

This program generates random 16-byte key (2^{128} bits). Then it repeatedly does the following: reads 8 bytes from standard input, encrypts them and writes to standard output. Note that r (0x083e5342) == 0x9e3779b9. Google search for 9e3779b9 leads to Wikipedia page about Tiny Encryption Algorithm (TEA). Further analysis confirms that TEA is indeed used in encryptor.c .

data.torrent contains an URL which points to https://dl.dropboxusercontent.com/s/e0w96mphez7v7po/9370b0b2b3abe901f6287b26937e6b88.jpg.enc . The filename suggests that this is an encrypted JPEG picture.

A program that generates random key and encrypts a file... almost like one of those ransom trojans!

So I need to decrypt the file. But how? Isn't that TEA strong enough?

First, the key is derived from the 32-bit value returned by the time() function. So there are no more than 2^{32} keys (probably even less due to the fact that PRNG is used for key generation)!

Second, compare encrypted picture and the picture from the Level 1 (**c321553877c582edc9435f97f5bcd7e7.jpg**):

```
00000000:  9F F0 E6 C8-40 D8 E6 3E-CA F8 52 AF-0B 91 96 4A
00000010:  1F 7B C4 4A-4B C5 4C 9E-FE 39 D4 8B-C0 4C 8A F2
00000020:  08 5B 12 1F-F1 28 2A 08-30 D8 00 02-F7 F9 5C 21
00000030:  6A C6 3C D0-1E 97 39 AA-0D C4 14 9D-6F FF BB F0
00000040:  B5 9C DB 24-6A CA A7 52-CF BB 77 3F-6B B1 84 14
00000050:  EE 3F 2F 6D-9A 40 AD 0F-9A 88 58 4D-0A CC 0D 4D
00000060:  9A 88 58 4D-0A CC 0D 4D-9A 88 58 4D-0A CC 0D 4D
00000070:  9A 88 58 4D-0A CC 0D 4D-9A 88 58 4D-0A CC 0D 4D
00000080:  AF 7C BC F3-41 68 8D E4-1C DB 42 97-80 35 C9 0F
00000090:  AD 3C 86 A8-CC 85 09 2C-6E C7 F5 F8-22 FC AE AB
000000A0:  93 54 01 15-7A 67 E8 53-CA A4 6A F0-73 0A E7 C2
000000B0:  28 C9 F2 26-92 DB 97 66-75 B9 11 78-D7 F1 F2 03
000000C0:  1E A7 77 78-AC 5D F3 C5-17 1E 3B 9A-35 EC DE E4
000000D0:  5E CD 17 D5-7C A0 A3 FD-F7 59 E9 62-F7 46 CF 57
000000E0:  9A 88 58 4D-0A CC 0D 4D-9A 88 58 4D-0A CC 0D 4D
000000F0:  9A 88 58 4D-0A CC 0D 4D-9A 88 58 4D-0A CC 0D 4D
00000100:  9A 88 58 4D-0A CC 0D 4D-9A 88 58 4D-0A CC 0D 4D
00000110:  91 9D 5D AD-46 90 43 C7-E2 01 92 0A-65 37 3F F9
```

```
00000000:  FF D8 FF E0-00 10 4A 46-49 46 00 01-01 01 00 48
00000010:  00 48 00 00-FF ED 01 32-50 68 6F 74-6F 73 68 6F
00000020:  70 20 33 2E-30 00 38 42-49 4D 04 04-00 00 00 00
00000030:  00 FA 1C 01-5A 00 03 1B-25 47 1C 02-00 00 02 00
00000040:  02 1C 02 50-00 41 22 54-6F 6E 69 20-41 68 6F 6C
00000050:  61 20 20 20-20 20 20 20-20 20 20 20-20 20 20 20
00000060:  20 20 20 20-20 20 20 20-20 20 20 20-20 20 20 20
00000070:  20 20 20 20-20 20 20 20-20 20 20 20-20 20 20 20
00000080:  20 20 20 20-20 20 22 1C-02 3E 00 08-32 30 31 32
00000090:  30 38 30 31-1C 02 3F 00-06 31 37 34-30 31 37 1C
000000A0:  02 78 00 1F-4F 4C 59 4D-50 55 53 20-44 49 47 49
000000B0:  54 41 4C 20-43 41 4D 45-52 41 20 20-20 20 20 20
000000C0:  20 20 20 1C-02 37 00 08-32 30 31 32-30 38 30 31
000000D0:  1C 02 74 00-3F 54 6F 6E-69 20 41 68-6F 6C 61 20
000000E0:  20 20 20 20-20 20 20 20-20 20 20 20-20 20 20 20
000000F0:  20 20 20 20-20 20 20 20-20 20 20 20-20 20 20 20
00000100:  20 20 20 20-20 20 20 20-20 20 20 20-20 20 20 20
00000110:  20 20 20 20-1C 02 19 00-08 54 32 20-4D 6F 64 65
```

Assuming that both pictures are somewhat similar (made by the same author, using the same software...) it is very likely that **20 20 20 20 20 20 20 20** turned into **9A 88 58 4D 0A CC 0D 4D** after encryption!

All of this means that brute-force attack can be implemented.

Here's the code:

```
void bf (const unsigned long int startValue, const unsigned long int stopValue)
{
  unsigned long int i;
  unsigned long int k0, k1, k2, k3, k4, k5, k6, k7;

  i = startValue;
  k0 = r (0x083e5342);
  k1 = r (k0);
  while (1)
    {
      k2 = r (i);
      k3 = r (k2);
      k4 = r (k3);
      k5 = r (k4);

      k6 = 0x20202020;
      k7 = 0x20202020;

      if (0 == (i & 0xffffff))
        printf ("%.8lX\r", i);

      k1 = 0;

#define ROUND { k1 += k0; \
 k6 += (k7 << 4) + k2 ^ k7 + k1 ^ (k7 >> 5) + k3; \
 k7 += (k6 << 4) + k4 ^ k6 + k1 ^ (k6 >> 5) + k5; }


      ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND;
      ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND;
      ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND;
      ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND; ROUND;

      if (k6 == 0x4d58889a && k7 == 0x4d0dcc0a)
        {
          printf ("Bingo: %.8lX\n", i);
        }

      i++;
      if (i == stopValue)
        break;
    }
}
```

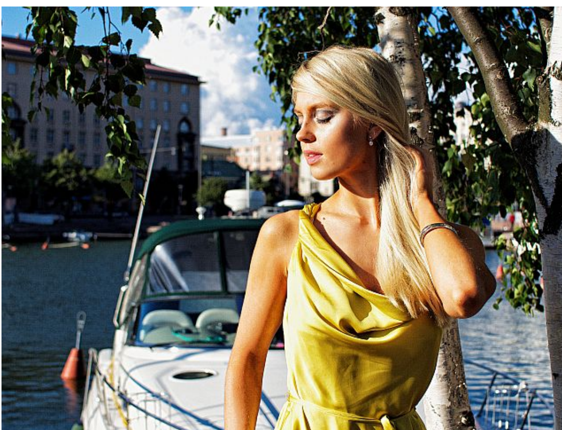
It is slightly optimized: function t() inlined, loop unrolled, array replaced with standalone variables. But the single most effective "optimization" would be simply running N copies of the program in parallel (where N is number of CPU cores). Each copy should bruteforce its own part of 2^{32} space, of course.

There are two time() values that give identical encryption keys: 0x508a67cc and 0xd08a67cc. 0x508a67cc == 1351247820, or, after conversion to human-readable format, "Fri, 26 Oct 2012 10:37:00 GMT"... or **"Fri, 26 Oct 2012 13:37:00 EEST"**. :)

And now look carefully at the data.torrent file again... :)

```
00000000:  64 31 33 3A-63 72 65 61-74 69 6F 6E-20 64 61 74  d13:creation dat
00000010:  65 69 31 33-35 31 32 34-37 38 32 30-65 34 3A 69  ei1351247820e4:i
00000020:  6E 66 6F 64-36 3A 6C 65-6E 67 74 68-69 38 33 34  nfod6:lengthi834
00000030:  34 34 34 30-65 34 3A 6E-61 6D 65 34-30 3A 39 33  4440e4:name40:93
00000040:  37 30 62 30-62 32 62 33-61 62 65 39-30 31 66 36  70b0b2b3abe901f6
00000050:  32 38 37 62-32 36 39 33-37 65 36 62-38 38 2E 6A  287b26937e6b88.j
00000060:  70 67 2E 65-6E 63 31 32-3A 70 69 65-63 65 20 6C  pg.enc12:piece l
```

The decrypted picture (the decryptor is in the Appendix B):



And its MD5 hash: **476d9247463dd91488fbd0d123e04ac1**

Level 3: file.mp3

This is an audio record of some low-speed data transfer. I searched for "decode audio record of data transfer" and found "Radio signal identification guide". And among many signal types mentioned there was one called APT (Automatic Picture Transmission). **A P T!** Maybe this is just a coincidence, but why not try?..

I downloaded and installed WXtoImg. WXtoImg can't open MP3 files, so I converted MP3 into WAV first (using MEncoder).

So WXtoImg decoded the picture.

At first sight it doesn't look very promising:

However, there is some regularity in it! Maybe I should simply adjust the dimensions?

I exported (File -> Save Raw Image as...) the image to a PNG file (PNG is lossless, thus better), converted it to .PPM file using IrfanView and started to experiment with width of the picture. Eventually I set the width to 1857 pixels and got this:

This is enough to read the URL: http://tinyurl.com/t2-sstv-hd. It redirects to https://www.dropbox.com/s/lmkr1uqkucn1dio/4e20c6c8d8b7473a2be84b12ab837bfd.jpg .

The MD5: **4e20c6c8d8b7473a2be84b12ab837bfd**

Level 4: news.txt

This is a posting to newsgroup. Header contains a string "MIME-Version: 1.0", but the message is in fact uuencoded. Decoders are readily available on the web (e.g. this site). The decoded message reads:

```
got mem dump of T's box @ Zetor. i can has pw hash dump? win pw == zip pw??? volatility ftw!

https://dl.dropboxusercontent.com/u/28851620/T/a444cf60f13382cb1c23336378126534948856 3a.zip
https://dl.dropboxusercontent.com/u/28851620/T/679f9a9737ecb42cc56a166f3e4830e225448df1.zip

-- APT
```

The first file (a444cf60f13382cb1c233363781265349488563a.zip) is a memory dump in *.vmem format (Google search query and the result):

```
 00000000:  53 FF 00 F0-53 FF 00 F0-53 FF 00 F0-53 FF 00 F0
 00000010:  53 FF 00 F0-53 FF 00 F0-53 FF 00 F0-53 FF 00 F0
 00000020:  A5 FE 00 F0-87 E9 00 F0-53 FF 00 F0-46 E7 00 F0
 00000030:  46 E7 00 F0-46 E7 00 F0-57 EF 00 F0-53 FF 00 F0
 00000040:  22 00 00 C0-4D F8 00 F0-41 F8 00 F0-FE E3 00 F0
 00000050:  39 E7 00 F0-59 F8 00 F0-2E E8 00 F0-D4 EF 00 F0
```

The second file (679f9a9737ecb42cc56a166f3e4830e225448df1.zip) is a password-protected ZIP archive with a JPEG picture inside (probably).

So... Download Volatility framework (you'll also need Python and PyCrypto toolkit). Learn how to use it (Google search query and the result). Dump hashes. Recover passwords from hashes (using e.g. ophcrack).

```
Hash:   81de36ec83691f0b22d3b69d51786748
Password:       T2INFOSEC

Hash:   aad3b435b51404eeaad3b435b51404ee
Password:       Empty password...

Hash:   469c976ed23a70e0799d1f5c3b02f777
Password:       37-3CN6*WENQRF
```

Try "T2INFOSEC" as the archive password (no, it won't do!). Think again. Try "t2infosec". Yeah!

Or...

...simply pay attention and notice that the picture is password-protected, but its filename is not!

And the picture is just happens to be named after its MD5 hash... :)

The MD5: **e79d2f8834910399c34192a2f1f8fc0e**

Level 5: photo02.jpg

This is our QR-code:

Alas, it doesn't get recognized (neither by ZXing Decoder Online nor by app in my smartphone).

After reading Wikipedia, ISO/IEC 18004 and especially this tutorial (a good one!) I found out that:

- Each QR-code is made of "modules" (dark and light squares). One module contains one bit of information.
- This particular QR code is sized 29x29 (i.e. "version 3"). 243 modules (markup) + 31 (format/version info) + 567 (useful data and ECC - error-correction codes) = 29x29 = 841 modules.
- ECC are used to detect errors **and to correct them**. "Designer QR-codes" (in which part of the code is replaced with some artwork) are possible thanks to ECC.
- The more ECC QR-code contains (there are four levels - L, M, Q, H), the more error-tolerant it is, but the less space is there for useful data. Our code has ECC level M, i.e. up to 15% of 567 modules can be damaged/obscured, and the code still should be readable.
- But it isn't readable. Here's why: in this QR-code, a block of 13x13 = 169 modules (in the upper right corner) is obscured. Some of the missing info can be easily restored, because
  - 8x8 modules form finder pattern square,
  - 5 modules form part of timing pattern,
  - and 8 modules contain format info.

  169 - (8*8+5+8) = 92 modules. Or about 16.2% of 567. Pity.

Here's the QR-code with some data restored:

So I chose obvious but time-consuming path and wrote my own decoder (not an universal one, of course!; just the one that could help me read this particular QR-code).

```
Format and version info (15 bits): 001101110000101
ECC level: M
bitMask: 6
cntBadBits: 92

Type: 4
Size: 40 bytes
Decoded: ht???//tiny???.com/???????utionzo????mgz
```

This is easy! "http://tinyurl.com". But even adding just "http://" would be enough.

Now it should read without problems.

http://tinyurl.com/ihazsolutionzomgzomgz redirects to https://www.dropbox.com/s/qu3fu89hgrht516/60e327e0fac73eb6fa291bff84497c2a.jpg.

The MD5: **60e327e0fac73eb6fa291bff84497c2a**

And what about the barcode? Well, it is a decoy. I couldn't decode it automatically (because important information, such as start/stop zones, was removed from the barcode.)

So I decoded it using pen, paper and patience. It's a Code 128 barcode. Wikipedia article on Code 128 contains a nice helpful table with all possible combinations of stripes. The decoded message reads 1!Close but no cig :)

Appendix A. Program that solves the Klondike puzzle

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

#define NUMROWS 23
#define NUMCOLS 23
#define NUMPTS 325

const char maze[NUMROWS][NUMCOLS] = {
  {"......................."},
  {"..........297.........."},
  {".......761535335......."},
  {".....6857751577868....."},
  {"....994197152865988...."},
  {"...96891151866261992..."},
  {"...73991397133299717..."},
  {"..8769293743173922895.."},
  {"..6577918315711127694.."},
  {"..6782924316431235714.."},
  {".539577331346142212487."},
  {".349628858573412715986."},
  {".465163811334513166439."},
  {"..5825639918441316116.."},
  {"..5979918927228881154.."},
  {"..2769828746315842154.."},
  {"...79724277771982163..."},
  {"...75119226162197587..."},
  {"....699815518895974...."},
  {".....2645696781736....."},
  {".......815849693......."},
  {"..........957.........."},
  {"......................."},
};


int marks[NUMROWS][NUMCOLS];


#define MAXMOVES NUMPTS
#define NUMDIR 8
#define NEVER_VISITED 999
#define BEEN_THERE -1

typedef struct _Cell
{
  int row;
  int col;
}
Cell;

const Cell direction[NUMDIR] = {
  {-1, 0},
  {-1, 1},
  {0, 1},
  {1, 1},
  {1, 0},
  {1, -1},
  {0, -1},
  {-1, -1}
};


const Cell start = { 11, 11 };


char solution[MAXMOVES + 1];

int
IsDigit (char c)
{
  return ((c >= '0') && (c <= '9'));
}

int
ValidateRow (const int row)
{
  return ((row >= 0) && (row < NUMROWS));
}

int
ValidateColumn (const int col)
{
  return ((col >= 0) && (col < NUMCOLS));
}

int
ValidateCoordinates (Cell c)
{
  return (ValidateRow (c.row) && ValidateColumn (c.col));
}

void
CalculateDestination (const Cell curr, Cell * dest, int dir, int numSteps)
{
```

```
  dest->row = curr.row + numSteps * direction[dir].row;
  dest->col = curr.col + numSteps * direction[dir].col;
}


void
PrintSolution (void)
{
  Cell curr, dest;
  int i, j;

  static char colornames[2 + 12][16] = {
    "white", "beige",
    "red", "gold", "orange", "yellow", "greenyellow", "lime", "aquamarine",
    "aqua", "skyblue", "violet", "magenta", "gray"
  };

  printf ("<table style=" "\x22"
          "border-collapse:collapse;text-align:center;" "\x22>\n");

  for (i = 0; i < NUMROWS; i++)
    {
      printf ("<tr>");
      for (j = 0; j < NUMCOLS; j++)
        {
          unsigned long color;

          if (NEVER_VISITED == marks[i][j])
            {
              if (maze[i][j] == '.')
                color = 1;
              else
                color = 0;
            }
          else
            color = marks[i][j] + 2;

          switch (marks[i][j])
            {
            default:
              {
                char x[2];
                x[0] = maze[i][j];
                x[1] = 0;

                printf ("<td style=\x22"
                        "background-color:%s;color:black;width:1em;height:1em;"
                        "border:1px solid black;border-collapse:collapse;padding:5px;"
                        "\x22" ">%s</td>", colornames[color],
                        (maze[i][j] == '.') ? (" ") : (&x[0]));
                break;
              }
            }
        }
      printf ("</tr>\n");
    }
  printf ("</table>\n");

  printf ("<table>\n<tr>\n");
  curr = start;
  for (i = 0; NUMPTS; i++)
    {
      if (solution[i] < '0')
        break;

      printf ("<td style=\x22"
              "background-color:%s;color:black;width:1em;height:1em;"
              "border:1px solid black;border-collapse:collapse;padding:5px;"
              "\x22" ">%c</td>\n"
              "<td style=\x22"
              "background-color:white;color:black;width:1em;height:1em;"
              "border:1px solid black;border-collapse:collapse;padding:5px;"
              "\x22" ">%.2s</td>\n",
              colornames[i + 2],
              maze[curr.row][curr.col],
              &"N NEE SES SWW NW"[2 * (solution[i] - '0')]);

      CalculateDestination (curr, &dest, solution[i] - '0',
                            maze[curr.row][curr.col] - '0');


      curr = dest;
    }
  printf ("</tr></table>\n");

}


void
Tracer (const int nMove, const Cell curr, int setMarks)
{
  int i, k;

  if ((nMove < MAXMOVES) &&
      ValidateCoordinates (curr) && IsDigit (maze[curr.row][curr.col]))
    {;
    }
```

```
  else
    return;

  if (setMarks)
    {
      marks[curr.row][curr.col] = nMove;
    }

  for (i = 0; i < NUMDIR; i++)
    {
      Cell dest, penult;
      int numSteps;

      numSteps = maze[curr.row][curr.col] - '0';

      CalculateDestination (curr, &dest, i, numSteps);
      CalculateDestination (curr, &penult, i, numSteps - 1);
      if (!ValidateCoordinates (dest))
        continue;

      if (!setMarks)
        {
          solution[nMove] = i + '0';
        }

      if (!IsDigit (maze[penult.row][penult.col]))
        continue;

      if (!IsDigit (maze[dest.row][dest.col]))
        {
          if (setMarks)
            {
              marks[dest.row][dest.col] = nMove + 1;
              continue;
            }
          else
            {
              PrintSolution ();
            }
        }
      else
        {
          if (setMarks)
            {
              if (marks[dest.row][dest.col] == NEVER_VISITED)
                marks[dest.row][dest.col] = nMove + 1;
            }
          else
            {
              if (marks[dest.row][dest.col] != (nMove + 1))
                continue;       // been there - skip

              Tracer (nMove + 1, dest, setMarks);
            }
        }
    }

  if (!setMarks)
    solution[nMove] = 0;
}


main ()
{
  int i, j, nMove, cnt;

  for (i = 0; i < NUMROWS; i++)
    for (j = 0; j < NUMCOLS; j++)
      marks[i][j] = NEVER_VISITED;

  marks[start.row][start.col] = 0;
  memset (&solution[0], 0, sizeof (solution));

// do breadth-first search

  for (nMove = 0; nMove < NUMPTS; nMove++)
    {
      cnt = 0;
      for (i = 0; i < NUMROWS; i++)
        for (j = 0; j < NUMCOLS; j++)
          {
            if ('.' == maze[i][j])
              continue;

            if (nMove == marks[i][j])
              {
                Cell a;
                a.row = i;
                a.col = j;
                Tracer (nMove, a, 1);
                cnt++;
              }
          }

      if (0 == cnt)
        break;
    }
```

```
  Tracer (0, start, 0);

  return 0;
}
```

Appendix B. TEA decryptor

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>

#ifdef _WIN32
#include <io.h>
#include <fcntl.h>
# define SET_BINARY_MODE(handle) setmode(handle, O_BINARY)
#else
# define SET_BINARY_MODE(handle) ((void)0)
#endif

unsigned long int
r (unsigned long int p)
{
  return 1664525 * p + 1013904223;
}

void
t (unsigned long int *p, unsigned long int a, unsigned long int c)
{
  p[a ^ 1] -= (p[a] << 4) + p[c] ^ p[a] + p[1] ^ (p[a] >> 5) + p[c + 1];
} int

main (void)
{
  unsigned long int k[10];

SET_BINARY_MODE(0);
SET_BINARY_MODE(1);

  k[5] = r (k[4] = r ((0, k[3] = r (k[2] = r ((k[1] = r (k[0] = r (0x083e5342)), 1351247820 ))))));
  do
    {
      k[7] = k[6] = 0;
      if ((k[8] = read (0, k + 6, 8)) == 0)
        break;
      for (k[9] = 0, k[1] = 0x6526b0d9; k[9] < 32; k[9]++)
        {
          k[1] -= k[0];
          t (k, 6, 4);
          t (k, 7, 2);
        }
      write (1, k + 6, 8);
    }
  while (k[8] == 8);
  fprintf (stderr, "Your key is %08x %08x %08x %08x.\n", k[2], k[3], k[4], k[5]);
  return 0;
}

;
```

Appendix C. QR-code tool

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

#define SQSIZE 29
#define NUMDATABITS 567

#define LIGHT_MODULE 0x20
#define DARK_MODULE 'x'


char z[SQSIZE][SQSIZE] = {
  "xxxxxxx xx  x x ????? xxxxxxx",
  "x     x x xx    ????? x     x",
  "x xxx x x x xxx ????? x xxx x",
  "x xxx x   x  x x????? x xxx x",
  "x xxx x x x  xx ????? x xxx x",
  "x     x   xx  x ????? x     x",
  "xxxxxxx x x x x x x x xxxxxxx",
  "         x xxx x?????        ",
  "x  xxxxxx x   xx?????x  x xxx",
  "x xxx     x x xx?????????????",
  "x x x xxxxxxxx x?????????????",
  "    x    xxxx x ?????????????",
  "  x  xx  x      ?????????????",
  "   x x xxx xx  xxxx   xxxx xx",
  "  x xxx xxxxx  x   xx  xxxx x",
  " x xx  xx  xx xx  xx  x   x x",
  "  xxx xx x xx x    xxx       ",
  "xxxx x xx x xx x x x    xxxx ",
  "xxx xxxxxxx x  xxx x   x    x",
  "xxx     x   x xx   x  xx xx  ",
  "xxxx xx xx xxxx  x  xxxxxxxx ",
  "        xx xx xx    x   xx   ",
  "xxxxxxx xxxxx xx x xx x xx   ",
  "x     x xxx   xx xxxx   xx  x",
  "x xxx x x  x xx x  xxxxxx  x ",
  "x xxx x xx  x  xxxx  x    x x",
  "x xxx x   xxx xx  x xx xxx xx",
  "x     x   x   xx      xxxxx x",
  "xxxxxxx xxxxx      xx xx     "
};



const int FormatTrack[15 * 2] = {
  0, 8, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 2, 0, 1, 0, 0, -1, 0, -2, 0, -1, 0, -1,
  0, -1, 0, -1, 0, -1
};

const int DataTrack[NUMDATABITS*2] = {
28, 28, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0,
-1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1,
0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1,
1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, 0, -1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1,
1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1,
0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0,
-1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 0, -1, 0, -1, -1, 1, 0,
-1, -1, 1, 0, -1, -1, 1, 0, -1, -6, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1,
0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1,
1, 0, -1, -1, 1, 0, -1, 0, -1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1,
1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1,
0, -1, 6, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 0, -1, 0, -1, -1, 1,
0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 0, -1, 0, -1, 0, -1, 0, -1, 0, -1, 1, 0,
-1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1,
0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1,
1, 0, -1, -2, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1,
-1, 1, 0, -1, 0, -1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1,
1, 1, 0, -1, 2, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1,
1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1,
0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0,
-1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 0, -1, 0, -1, -1, 1, 0,
-1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1,
0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1,
1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1,
-1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -2, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0,
-1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, 0, -1, 0, -1, 1, 1, 0, -1, 1, 1, 0,
-1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 2, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1,
1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1,
1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1,
0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0,
-1, 0, -1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1,
0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1,
1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1,
-1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -2, 1, 0,
-1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, 0, -1,
0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 2, 1, 0,
-1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1,
1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1,
1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1,
0, -1, 1, 1, 0, -1, 1, 1, 0, -1, -8, -1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1,
1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1,
-1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, 0, -2, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1,
```

```
1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1,
1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 0, -1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1,
1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1,
-1, 1, 0, -1, -1, 1, 0, -1, -1, 1, 0, -1, 0, -1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1,
1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1, 1,
1, 0, -1, 1, 1, 0, -1, 1, 1, 0, -1
};


unsigned long int eccLevel, bitMask;

char bits[NUMDATABITS + 1];

void
PrintQRCode (void)
{
  int i, j;
  printf ("<table style=" "\x22"
          "border-collapse:collapse;text-align:center;\x22" ">");

  for (i = 0; i < SQSIZE; i++)
    {
      printf ("<tr>");
      for (j = 0; j < SQSIZE; j++)
        {
          static const char colornames[3][12] = { "white", "black", "red" };
          int color;

          switch (z[i][j])
            {
            case LIGHT_MODULE:
              color = 0;
              break;
            case DARK_MODULE:
              color = 1;
              break;
            default:
              color = 2;
              break;
            }

          printf ("<td style=\x22"
                  "background-color:%s;color:white;width:1em;height:1em;"
                  "border:0px solid black;border-collapse:collapse;padding:0px;\x22></td>",
                  colornames[color]);
        }
      printf ("</tr>\n");
    }

  printf ("</table><br><br><br>");
}



void
ReadFormatInfo (void)
{
  unsigned long int i, j, row, col;
  unsigned long int bit;
  unsigned char formatInfo[16];

  row = 0;
  col = 0;

  memset (&formatInfo[0], 0, sizeof (formatInfo));

  for (i = 0; i < (sizeof (FormatTrack) / sizeof (FormatTrack[0])); i += 2)
    {
      row += FormatTrack[i];
      col += FormatTrack[i + 1];

      j = 14 - (i / 2);

      bit = (z[row][col] == DARK_MODULE);

      if ("101010000010010"[j] == '1')
        bit ^= 1;

      formatInfo[j] = bit + '0';
    }

  printf ("Format and version info (15 bits): ");
  printf ("%s\n", formatInfo);

  eccLevel = ('1' == formatInfo[0]);
  eccLevel <<= 1;
  eccLevel |= ('1' == formatInfo[1]);

  printf ("ECC level: %c\n", "MLHQ"[eccLevel]);

  bitMask = ('1' == formatInfo[2]);
  bitMask <<= 1;
  bitMask |= ('1' == formatInfo[3]);
  bitMask <<= 1;
  bitMask |= ('1' == formatInfo[4]);
  printf ("bitMask: %d\n", bitMask);
```

```
}


int
FlipBit (const int maskId, const row, const col)
{
  int flag;

  switch (maskId)
    {
    case 0: flag = (0 == ((row + col) % 2)); break;
    case 1: flag = (0 == (row % 2)); break;
    case 2: flag = (0 == (col % 3)); break;
    case 3: flag = (0 == ((row + col) % 3)); break;
    case 4: flag = (0 == (((row / 2) + (col / 3)) % 2)); break;
    case 5: flag = (0 == (((row * col) % 2) + ((row * col) % 3))); break;
    case 6: flag = (0 == ((((row * col) % 2) + ((row * col) % 3)) % 2)); break;
    case 7: flag = (0 == ((((row + col) % 2) + ((row * col) % 3)) % 2)); break;
    default: flag = 0; break;
    }

  return flag;
}


void
ReadBits (const int maskId)
{
  unsigned long int i, row, col, bit, cntBits, cntBadBits;

  cntBits = 0;
  cntBadBits = 0;
  row = 0;
  col = 0;

  for (i = 0; i < (sizeof (DataTrack) / sizeof (DataTrack[0])); i += 2)
    {
      row += DataTrack[i];
      col += DataTrack[i + 1];

      bit = (LIGHT_MODULE == z[row][col]) ? 0 : ((DARK_MODULE == z[row][col]) ? 1 : 2);

      if (FlipBit( maskId, row, col))
        bit ^= 1;

      bits[cntBits] = (bit < 2) ? ('0' + bit) : ('?');
      if ('?' == bits[cntBits])
        cntBadBits++;

      cntBits++;
    }

  bits[cntBits] = 0;

  printf ("\nRead %d bad modules.\n\n", cntBadBits);
}



void
WriteBits (const int maskId)
{
  unsigned long int i, row, col, bit, cntBadBits;

  cntBadBits = 0;
  row = 0;
  col = 0;

  for (i = 0; i < (sizeof (DataTrack) / sizeof (DataTrack[0])); i += 2)
    {
      row += DataTrack[i];
      col += DataTrack[i + 1];

      switch (bits[i / 2])
        {
        case '0': bit = 0; break;
        case '1': bit = 1; break;
        default:  bit = 2; break;
        }

      if (FlipBit( maskId, row, col))
        bit ^= 1;

      switch (bit)
        {
        case 0: z[row][col] = LIGHT_MODULE; break;
        case 1: z[row][col] = DARK_MODULE; break;
        default: z[row][col] = '?'; cntBadBits++; break;
        }
    }

  printf ("\nWrote %d bad modules.\n\n", cntBadBits);
}
```

```
unsigned char
ReadChar (const int startPos, const int numBits)
{
  unsigned char result;
  int i;

  result = 0;
  for (i = 0; i < numBits; i++)
    {
      switch (bits[startPos + i])
        {
        case '0':
        case '1':
          result <<= 1;
          result |= (bits[startPos + i] & 1);
          break;
        default:                // '?'
          return '?';
        }
    }

  return result;
}



void
WriteChar (const int startPos, const int numBits, const unsigned char c)
{
  int i;
  unsigned int bitMask = 0x80;

  for (i = 0; i < numBits; i++)
    {
      bits[startPos + i] = '0' + !!(c & bitMask);
      bitMask >>= 1;
    }
}


void
main ()
{
  int type;
  int numBytes;
  int i, j;
  char url[128];

  printf( "<!--\n" );

  ReadFormatInfo ();
  memset (&bits[0], 0, sizeof (bits));
  ReadBits (bitMask);

  type = ReadChar (0, 4);
  numBytes = ReadChar (4, 8);
  printf ("Type: %d\n", type);
  printf ("Size: %d bytes\n", numBytes);

  for (i = 0; i < numBytes; i++)
    url[i] = ReadChar (12 + i * 8, 8);
  url[i] = 0;

  printf ("Decoded URL: %s\n", url);


  // add guessed chars

  memcpy (&url[0], "http://tinyurl.com/???????utionzo????mgz", 40);

  for (i = 0; i < numBytes; i++)
    {
      if ('?' != url[i])
        WriteChar (12 + i * 8, 8, url[i]);
    }

  WriteBits(bitMask);
  printf( "-->\n" );

  PrintQRCode ();
}
```