Description of the CSAIL2019 Time Capsule Crypto-Puzzle

by Ronald L. Rivest

May 14, 2019

We describe a "refreshed" version of the LCS35 time-lock crypto puzzle, to extend the puzzle out to the year 2034 (since the original LCS35 puzzle was solved in 2019 by Bernard Fabrot).

See the description of the LCS35 puzzle given here:
http://people.csail.mit.edu/rivest/pubs.html#Riv99b
for explanation and details of the original puzzle.

The problem is to compute 2^(2^t) (mod n) for specified values of t and n. Here n is the product of two large primes, t is chosen to set the desired level of difficulty of the puzzle, and "^" denotes exponentiation. This overall structure is retained in the refreshed version of the puzzle.

Note that the puzzle can be solved by performing t successive squarings modulo n, beginning with the value 2. That is, set

$$W(0) = 2$$

$$W(i+1) = (W(i) \hat{} 2) \pmod{n} \quad \text{for } i=1, 2, \ldots$$

and compute W(t).

There is no known way to perform this computation more quickly than to perform the t squarings sequentially, unless the factorization of n is known.

In the original (LCS35) version of the puzzle, the number n was about 2048 bits long, and the number t was about 80 trillion. We estimated that the puzzle would take 35 years of continuous computation to solve, with the computer being replaced every year by the next fastest model available.

Since the puzzle has now been solved, however, we provide here "refreshed" values of n and t, that should restore the puzzle to having its intended difficulty (that is, it is estimated to require until about 2034 to solve).

One change is that the modulus n is now 3072 bits in length, instead of about 2048.

The new version of the puzzle also specifies that

$$t = 2 \hat{} 56 = 72057594037927936,$$

which is about 1000 times larger in magnitude than t for the original LCS35 puzzle.

While the full puzzle requires a solution for t = 2 ^ 56, CSAIL is also interested in solutions for t = 2 ^ k for 56/2 <= k < 56; these are called "milestone versions of the puzzle".

In order to allow the CSAIL director in the year 2034 (or whenever) to verify a submitted solution, we have arranged things so that solving

the puzzle also enables the solver to factor the modulus n, as described below.

Here is a smaller example of the puzzle.
Suppose n = 11*23 = 253, and t = 16 = 2^4. Then we can compute:

```
2^(2^0) = 2^1 = 2 (mod 253)
2^(2^1) = 2^2 = 4 (mod 253)
2^(2^2) = 4^2 = 16 (mod 253)
2^(2^3) = 16^2 = 3 (mod 253)
2^(2^4) = 3^2 = 9 (mod 253)
2^(2^5) = 9^2 = 81 (mod 253)
2^(2^6) = 81^2 = 236 (mod 253)
2^(2^7) = 236^2 = 36 (mod 253)
2^(2^8) = 36^2 = 31 (mod 253)
2^(2^9) = 31^2 = 202 (mod 253)
2^(2^10) = 202^2 = 71 (mod 253)
2^(2^11) = 71^2 = 234 (mod 253)
2^(2^12) = 234^2 = 108 (mod 253)
2^(2^13) = 108^2 = 26 (mod 253)
2^(2^14) = 26^2 = 170 (mod 253)
2^(2^15) = 170^2 = 58 (mod 253)
2^(2^16) = 58^2 = 75 (mod 253)
```

The values 2^(2^8) = 31 and 2^(2^4) = 9 are solutions to "milestone versions" of this toy puzzle, as 16 = 2^4, 8 = 2^3, and 4 = 2^2.

Thus, the "w" value computed for the puzzle is 75 (decimal), which is 4a (hex). If we have a "z" value for the puzzle of 13 (hex), then the "secret message" for the example is (4a xor 13) = 47 (hex). (The secret message should then be interpreted in ascii at 8 bits per character.)

Attached below is the Java code for generating the puzzle, and the actual refreshed puzzle itself.

Anyone who believes to have solved the puzzle, or to have computed a solution to a "milestone version" of the puzzle, should first check the CSAIL web site (or contact CSAIL) to see if the solution is new. If it is, then they should submit the solution to CSAIL. If they have solved the full version, then they should also submit the resultant English sentence along with the factorization of n and relevant solution notes to the Director of CSAIL. For a solution to a milestone version of the puzzle, please also submit an argument that the result is correct. (See the "small prime" of Shamir described in the original paper, for example, or use a more sophisticated verifiable delay function proof.)

Upon verification of the solution, the CSAIL Director will unseal the capsule at a ceremony set up for that purpose. If no solution is established by September 2033, then the CSAIL Director will unseal the capsule at CSAIL's 31st birthday celebration in 2034, or at suitable alternate event. In the absence of a CSAIL Director, the President of MIT will designate another official or individual.

Good luck!

```
================================================================================
=
/*    Program to create "Time-Lock Puzzle" for LCS35 Time Capsule
      Ronald L. Rivest
      March 30, 1999 (revised May 14, 2019 for CSAIL 2019 puzzle)
 */

import java.io.*;
import java.util.Random;
import java.math.BigInteger;
import java.lang.Math;

public class TimeLockPuzzle {

      public TimeLockPuzzle () {}

      static BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));

    static public void main(String args[]) throws IOException {
        System.out.println("Creating CSAIL2019 Time Capsule Crypto-
Puzzle...");
        CreatePuzzle();
        System.out.println("Puzzle Created.");
    }

    final static BigInteger ONE = new BigInteger("1");
    final static BigInteger TWO = new BigInteger("2");

    static public void CreatePuzzle() throws IOException {
        // Compute count of squarings to do each year
        // BigInteger squaringsPerSecond =
            // new BigInteger("3000"); // in 1999
        // System.out.println("Assumed number of squarings/second (now) = "+
        //               squaringsPerSecond);
        // BigInteger secondsPerYear = new BigInteger("31536000");
        // BigInteger s = secondsPerYear.multiply(squaringsPerSecond); //
first year
        // System.out.println("Squarings (first year) = "+s);
        // int years = 15;               // was 35 for orignal LCS 35 puzzle
        // double Moore =
            // 1.58740105197;  // = 2^(2/3) Moore's Law constant per year
(used for LCS35 puzzle)
            // 1.1000;            // Estimate from Simon Peffers
            // (Note that this is also in code in a few lines as a constant.)
        // Method used for computing t for LCS35
        // BigInteger t = new BigInteger("0");
        // for (int i = 1;i<=years;i++)
        //    {   // do s squarings in year i
        //        t = t.add(s);
        //        // apply Moore's Law to get number of squarings to do the
next year
        //        s = s.multiply(new BigInteger("11000")).divide(new
BigInteger("10000"));
        //    }

        // Method for 2019 puzzle: set t to 2**56
        int log2_t = 56;
```

```
BigInteger t = (new BigInteger("1")).shiftLeft(log2_t);
System.out.println("Squarings (total)= "  + t);

// Now generate RSA parameters
int primelength = 1536;
System.out.println("Using "+primelength+"-bit primes.");
BigInteger twoPower = (new BigInteger("1")).shiftLeft(primelength);

String pseed = getString("large random integer for prime p seed");
BigInteger prand = new BigInteger(pseed);

String qseed = getString("large random integer for prime q seed");
BigInteger qrand = new BigInteger(qseed);

System.out.println("Computing...");

BigInteger FIVE = new BigInteger("5");
BigInteger p = new BigInteger("7");
BigInteger q = new BigInteger("11");
BigInteger n = new BigInteger("77");
BigInteger max_n = (new BigInteger("1").shiftLeft(2*primelength));
BigInteger min_n = (new BigInteger("1").shiftLeft(2*primelength-1));
while (n.compareTo(min_n)==-1 || n.compareTo(max_n)==1)
    {
      // Note that 5 has maximal order modulo 2^k (See Knuth)
      prand = prand.add(ONE);
      p = getNextPrime(FIVE.modPow(prand,twoPower));
      System.out.println("p = "+p);

      qrand = qrand.add(ONE);
      q = getNextPrime(FIVE.modPow(qrand,twoPower));
      System.out.println("q = "+q);

      n = p.multiply(q);
      System.out.println("n = "+n);
    }

BigInteger pm1 = p.subtract(ONE);
BigInteger qm1 = q.subtract(ONE);
BigInteger phi = pm1.multiply(qm1);
System.out.println("phi = "+phi);

// Now generate final puzzle value w
BigInteger u = TWO.modPow(t,phi);
BigInteger w = TWO.modPow(u,n);
System.out.println("w (hex) = "+w.toString(16));

// Obtain and encrypt the secret message
// Include seed for p as a check
StringBuffer sgen = new StringBuffer(getString("string for secret"));
sgen = sgen.append(" (seed value b for p = "+prand.toString()+")");
System.out.println("Puzzle secret = "+sgen);
BigInteger secret = getBigIntegerFromStringBuffer(sgen);
if (secret.compareTo(n) > 0)
    { System.out.println("Secret too large!"); return; }
BigInteger z = secret.xor(w);
System.out.println("z(hex) = "+z.toString(16));
```

```
        // Write output to a file
        PrintWriter pw = new PrintWriter(new FileWriter("puzzleoutput.txt"));
        pw.println("Crypto-Puzzle for CSAIL 2019 Time Capsule.");
        pw.println("Created by Ronald L. Rivest. May 14, 2019.");
pw.println();
        pw.println("(Successor to LCS 35 Crypto-Puzzle.)"); pw.println();
        pw.println("Puzzle parameters (all in decimal):"); pw.println();
        pw.print("n = "); printBigInteger(n,pw); pw.println();
        pw.print("t = "); printBigInteger(t,pw);
        pw.print("  = 2 ** "); pw.print(log2_t); pw.println(); pw.println();
        pw.print("z = "); printBigInteger(z,pw); pw.println();
        pw.println("To solve the puzzle, first compute w = 2^(2^t) (mod
n).");
        pw.println("Then exclusive-or the result with z.");
        pw.println("(Right-justify the two strings first.)");
        pw.println();
        pw.println("The result is the secret message (8 bits per
character),");
        pw.println("including information that will allow you to factor n.");
        pw.println("(The extra information is a seed value b, such that ");
        pw.println("5^b (mod 2^1536) is just below a prime factor of n.)");
        pw.println(" ");
        pw.close();

        // Wait for input carriage return to pause before closing
        System.in.read();
    }

    static String getString(String what) throws IOException {
        // This routine is essentially a prompted "readLine"
        StringBuffer s = new StringBuffer();
        System.out.println("Enter "+what+" followed by a carriage return:");
        for (int i = 0;i<1000;i++)
           { int c = System.in.read();
             if (c<0 || c == '\n') break;
             if (c != '\r') // note: ignore cr before newline
               s = s.append((char)c);
           }
        return(s.toString());
    }

    static BigInteger getBigIntegerFromStringBuffer(StringBuffer s)
    throws IOException {
        // Base-256 interpretation of the given string
        BigInteger randbi = new BigInteger("0");
        for (int i = 0;i<s.length();i++)
            {   int c = s.charAt(i);
                randbi = randbi.shiftLeft(8).add(new
BigInteger(Integer.toString(c)));
             }
        System.out.println("Value of string entered (hex) =
"+randbi.toString(16));
        return randbi;
    }

    static void printBigInteger (BigInteger x, PrintWriter pw) {
        String s = x.toString();
        int charsPerLine = 60;
```

```
        for (int i = 0;i<s.length();i+=charsPerLine)
        {   if (i!=0) { pw.println(); pw.print("    "); }

pw.print(s.substring(i,java.lang.Math.min(i+charsPerLine,s.length())));
        }
        pw.println();
    }

    static BigInteger getNextPrime(BigInteger startvalue)
    {
        BigInteger p = startvalue;
        if (!p.and(ONE).equals(ONE))   p = p.add(ONE);
        while (!p.isProbablePrime(40)) p = p.add(TWO);
        return(p);
    }

}// end of TimeLockPuzzle class
================================================================================
=
Crypto-Puzzle for CSAIL 2019 Time Capsule.
Created by Ronald L. Rivest. May 14, 2019.

(Successor to LCS 35 Crypto-Puzzle.)

Puzzle parameters (all in decimal):

n = 474809754727201286617503413061677388505126074492005644486710
    619636071042455814765425270760494101231177589201256757906462
    053687463338505591900116762157771031136607205702942170513568
    430393481139013793780209643316395921689235118482669118001605
    519886679653623008552320068354906699567215583904228295559156
    849460306111329203904475384384648480711222838920423958171293
    110891982025021858635204389730623887202537819314111150742631
    144461349873631561421830476173554162699783903651772800068839
    401561061817976886834207039510014762029561669583444089424114
    790556556780829814902466852704523965014586209290411941287400
    776304104231428760477287686129441766402083279620913558718182
    645823558000382582372423580085016028485080973720098370355217
    935469186387604444337782243983407931357802908565807857573129
    024477859561522947241132683150266742576852000637175296327429
    629450606318225806436204878833839252826635151130492184785475
    064219269454112506587397 7

t = 72057594037927936
  = 2 ** 56

z = 281925200789646054484626895242259073034267440683167652674260
    444591975889902849810168128895728145609121926279767832267892
    542066973188291684718744670094846726572899884983417621775255
    327210450830507578704233244224354792555227248423668532198826
    815434660322678303055198560726336871038927854530931214528139
    373062870256369865359829985298135548898569521934607434544000
    441455075616167756392020564099520298066717731321210733579077
    301206361429295031940789176281366976549232723977044834008951
    835696482863894992874097176920270620602720510686448265399942
    665576829947343992975331586905362399981263293987821422331239
    630446286999665633015654768670005653277971729589550900484148
    746877649612226774676201443379399666029879354208782911706544
```

```
    158175417730799622034258201586627351071911874884702195535013
    165691782737205797563610954623717953897626420548504301835206
    050378592457192577621469438707103550461448491697592874506496
    05141583555374634939174l2

To solve the puzzle, first compute w = 2^(2^t) (mod n).
Then exclusive-or the result with z.
(Right-justify the two strings first.)

The result is the secret message (8 bits per character),
including information that will allow you to factor n.
(The extra information is a seed value b, such that
5^b (mod 2^1536) is just below a prime factor of n.)
==========================================================================
=
```