University of West Bohemia in Pilsen
Faculty of Applied Science

Semestral work:

SeedSeeker

Authors: Pavel Altmann, Jakub Kupčik,
Matěj Bartička, David Wimmer, Patrik Holub

Date: 1st July 2025

Obsah

1 Description

In programming, there is an often need to generate random values. The two most common cases are encryption and including randomness to introduce unpredictability. A very simple example would a slot machine. If someone could predict random values then he could abuse that knowledge and get rich.

In most cases the generated values are generated using a Pseudo-Random Number Generator (PRNG). This means that the random values aren't derived from true random events in our world, but from mathematical equations. These equations (or algorithms) should be built in such a way that they produce sequences of values that satisfy some random distribution. (Most commonly uniform distribution) They also share one fact: previous values define the following values. This also means that every such algorithm has a starting point, the seed.

From this seed all values follow, and if the seed could be deduced from the values generated then someone could "get rich"as mentioned. While there are know methods for some algorithms, the wasn't a tool that could apply these methods. This was our task.

We have created an application (SeedSeeker) that can attempt to reverse engineer the generator and it's seed from a sequence of it's outputs. This application was created in python and both generators and reversal methods are implemented as libraries for easily adding new content as more generators and reversal methods become available.

2 User manual

2.1 Download

There are two major ways to download SeedSeeker. For both them the source is SeedSeeker (`https://github.com/wren-projects/SeedSeeker`)

If the users don't have python installed or aren't comfortable with using python, they can download an already built binary in the Releases section on Github. Otherwise the users are free to download the repository and run the tool using python. The download and run process are also described on Github.

2.2 Running

Most of the information can be obtained from help page in the tool or the manpage for the tool.

There are 3 basic usages of the tool. First is to run a generator with the specified seed. How to specify seed for each generator will be explained in the Reversal Algorithms section. Then there is reversal mode which will attempt to reverse the sequence using each reversal algorithm. It will then print initial states for each reversal attempt that succeeded. This output can be used in the third usage case which is predicting future values. The program will take the initial state and generate user specified amount of future values.

There are however some caveats best explained beforehand. Most of the reversal algorithms do work for values generated with well defined generators. When the generator states are degenerated, then the outputs will most likely not match expected results. For example the tool cannot cope with repeating one single value. The program will outputs it's best attempt but if the user then uses it to predict future values, they might not match reality.

3 Reversal Algorithms

3.1 LCG

Let x_i be the sequence of LCG output. The sequence is given by the recursive formula:

$$x_{i+1} = (a \cdot x_i + c) \mod M$$

We first create a second sequence:

$$y_i = x_{i+1} - x_i$$

Note $y_{i+1} = a \cdot y_i \mod M$.

Let us denote

$$r_i = y_i \cdot y_{i+3} - y_{i+1} \cdot y_{i+2}$$

And note $r_i \equiv 0 \mod M$.

So to retrieve the value of modulo, we can just take an arbitrary number of r_is, calculate their GCD and estimate M based on that value.

Once we have modulo, we can just multiply r_{i+1} by the inverse of r_i in modulus M to retrieve a. Finally we trivially retrieve c.

3.2 ran3

The output sequence is simply the initial state. So we just consume 55 values and we know the full inital state.

3.3 Lagged Fibonacci

Let x_i be the sequence of outputs. We know, that $x_i = ((x_{i-r} + x_{i-s}) \mod M) + K$, where K is one iff $x_{i-1} > x_{i-r} + x_{i-s}$. We reverse this generator by bruteforce.

We try out all possible values of r and s (lower than some threshold). Then, for each of them, we calculate $x_i - x_{i-r} - x_{i-s}$. We know this is either $M - 1, M, M + 1$ or 0. By doing this multiple times, we pinpoint get the exact value of M.

3.4 Xoshiro256**

This generator uses a total of 256 bits of inner state broken up into 4 `uint_64` variables (stored in array 's').

For full description, see here.

Let us explore, how does the inner state change in one iteration:

$$[A, B, C, D] \rightarrow [A \oplus B \oplus D, A \oplus B \oplus C, A \oplus C \oplus (B << 17), \text{rotl}(B \oplus D)]$$

, where $\oplus$ denotes bitwise XOR.

Note that the output is always reversible into the second element of given state.

We can retrieve `s[1]` and `s[0]^s[1]^s[2]` from the first 2 outputs.

Let us denote t the state after one iteration. Note we know the value of `t[1]` and `t[2]`.

From 3th output, we can reconstruct `t[0]`.

4 Library Description