CELLULAR AUTOMATON BASED RANDOM NUMBER GENERATION

-Varul Srivastava

Abstract : *We explore an interesting way of generating random numbers, using the Wolfram's Rule 150 $(10010110)_2$ of Cellular Automaton. We also discuss how to remove user bias in passwords to prevent dictionary attacks to some extent.*

Cellular Automaton – An Introduction

Cellular Automaton is a system that uses some fixed rules and previous state of a system to calculate next generation/state of the system.

G(e+1) = F(G(e),r)

Here, $e+1^{th}$ generation is obrained from transformation rule r on e^{th} generation. Let us consider our example (rule 150) and solve few generations to get an idea of the rule.

Rule F : For the i^{th} bit of the $e+1^{th}$ generation we combine the $i-1^{th}$, i^{th} and $i+1^{th}$ bit of the e^{th} generation to form a 3 bit number j (0-7) and select the corresponding j^{th} bit of r (which is an 8 bit number.

Initial Generation G(0) : 11100101
Rule r : 10010110
G(1) = F(11100101,10010110)
G(1) = 11001100

This explains the progression of generation, and if we combine generations to a matrix M

$$M = \begin{bmatrix} G(0) \\ G(1) \\ . \\ . \\ G(n) \end{bmatrix}$$

This is a 2-D matrix and we can generate byte stream by reading bits **column-wise.**

Reading Bits from Matrix

We clearly see that there are multiple ways of reading byte streams, but with varying levels of privacy. Let us analyse the following ways :

1.Horizontal Generation of Stream: This is the poorest method possible, as after reading stream from first k (number of columns) bits, one is able to predict all other bits.

2.Vertical Generation of Stream: This is moderatley secure means, given we generate extra bits (unused in generating streams) so that prediction of generations is difficult. However, it is still possible to read first grid and predict next grid by optimising brute force to small base exponential time.

3. Diagonal Generation of Stream: This is also moderately secure, as it requires reading the entire matrix and then predict the next matrix, and security can be increased by creating additional bit streams similar to previous example.

4. Random Generation of Stream: This method is also under study. It is more efficient if the random number generator is salted with the password

(x_next) = (x * hash(pswd)* prime1 + prime2) % MOD.

is better than,

(x_next) = (x * prime1 + prime2) % MOD.

This is the most secure way that I have come up with. This way of stream generation gurantees large-exponent exponential complexity of brute force attacks.

Salting of Password – Removal of user bias

The potential of password is to have 256 values per character, but the inherent bias of user to use 'a' or 'e' over '^' or even 'q' is removed by a function that removes the inherent bias. The function is :

ch_new = (ch*Prime1 + Prime2) ^(prev_char)% 256