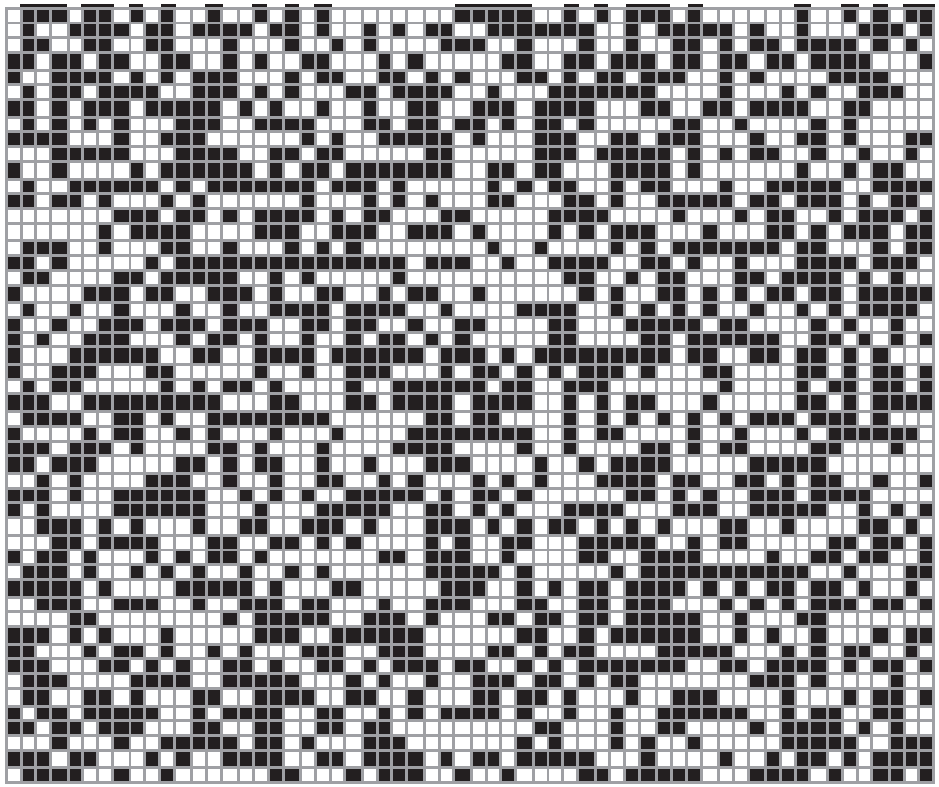
# Stream Ciphers



# Linear Congruential Generators

- A linear congruential generator produces a sequence of integers $x_i$ for $i = 1, 2, \ldots$ starting with the given initial (seed) value $x_0$ as

$$x_{i+1} = a \cdot x_i + b \pmod{n}$$

  where the multiplication and addition operation is performed modulo $n$, and therefore, $x_i \in \mathcal{Z}_n$
- This is a deterministic algorithm; the same $x_i$ value will always produce the same $x_{i+1}$ value, and the same seed $x_0$ will produce the same sequence $x_1, x_2, \ldots$
- There are only finitely many $x_i \in \mathcal{Z}_n$, and the sequence will repeat
- The period of the sequence is $w$ such that $x_{i+w} = x_i$

# Linear Congruential Generators

- For $(a, b, n) = (3, 4, 15)$, and $x_0 = 1$, we obtain the following sequence: $1, 7, 10, 4, 1, 7, 10, 4 \ldots$; the period is $w = 4$
- For $(a, b, n) = (3, 4, 15)$, and $x_0 = 2$, we obtain the following sequence: $2, 10, 4, 1, 7, 10, 4, 1, 7, \ldots$; the period is $w = 4$
- For $(a, b, n) = (3, 4, 17)$, and $x_0 = 1$, we obtain the following sequence: $1, 7, 8, 11, 3, 13, 9, 14, 12, 6, 5, 2, 10, 0, 4, 16, 1, 7, 8 \ldots$ the period is $w = 16$
- For $(a, b, n) = (3, 4, 17)$, and $x_0 = 15$, we obtain the following sequence: $15, 15, 15, \ldots$; the period is just $w = 1$
- For $(a, b, n) = (2, 4, 17)$, and $x_0 = 2$, we obtain the following sequence: $1, 6, 16, 2, 8, 3, 10, 7, 1, 6, 16, 2, \ldots$; the period is $w = 8$

# Period of LCGs

- Theorem: Given a LCG with parameters $(a, b, p)$ such that $p$ is prime, the period $w$ is equal to the order of the element $a$ in the multiplicative group $\mathcal{Z}_p^*$ for all $x_0$ seed values except $x_0 = -(a-1)^{-1} \cdot b \bmod p$.
- Since the group order is $p-1$, the period $w$ is always a divisor of $p-1$. The maximum period occurs when $a$ is a primitive element, whose order is $p-1$.
- For $(a, b, n) = (3, 4, 17)$, the order of the group is equal 16, while the order of the element $a = 3 \bmod 17$ is found as 16 since

$$\{3^1, 3^2, 3^3, \ldots, 3^{16}\} = \{3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1\}$$

On the other hand, the "bad seed" value is $x_0 = -(a-1)^{-1} \cdot b \bmod 11$

$$x_0 = -(3-1)^{-1} \cdot 4 = -2^{-1} \cdot 4 = -2 = 15 \pmod{17}$$

# A Practical LCG

- Since our processors have fixed data length, it is a good idea to select a prime as large as the word size, since we will perform mod $p$ arithmetic
- It turns out that $2^{31} - 1 = 2,147,483,647$ is a prime number; furthermore, the smallest primitive element in $\mathcal{Z}_p$ for $p = 2^{31} - 1$ is found as $a = 7^5 = 16,807$
- Also, $a$ is fairly close to the square root of $p$, therefore, we have a good, practical, general-purpose LCG, given as

$$\begin{aligned} x_{i+1} &= a \cdot x_i \pmod{p} \\ p &= 2^{31} - 1 = 2,147,483,647 \\ a &= 7^5 = 16,807 \end{aligned}$$

Since $a$ is a primitive element, the period of LCG is $w = 2^{31} - 2$

# Cryptographic Strength of LCGs

- Does the LCG satisfy requirements R1 and R2?
- Analysis and experiments show that LCGs with large $p$ (such as the previous practical LCG) are (almost) acceptable as statistically random, but there are some deficiencies
- Unfortunately, the LCGs do not satisfy R2 since they are is highly predictable: Assuming $a$ and $p$ are known, given a single element $x_i$, any future element of the sequence can be computed as $x_{i+k} = a^k x_i \bmod n$
- Similarly, given $x_i$, any past element of the sequence can be computed as $x_{i-k} = a^{-k} x_i = (a^{-1})^k \bmod n$
- Inversion: the seed $x_0$ can be computed if any element $x_i$ of the sequence is known, by working back from $i$ down to 0

# Cryptographic Strength of LCGs

- In general, we need to assume that $a$ and $p$ are fixed parameters of the RNG and therefore they are not changeable, i.e., they are not part of the key ($x_0$, the seed) — they can be discovered by reverse engineering
- If we can bundle $a$ and $p$ with the seed $x_0$, then we can claim more security — it would be much harder to discover the key ($a$, $p$, and $x_0$) given a limited number of elements $x_i$ from the sequence $x_1, x_2, \ldots$
- Note that $x_{i+1} = a \cdot x_i \bmod p$ implies $x_{i+1} = a \cdot x_i + N \cdot p$ for some integer $N$; however, $N$ is different for every pair $(x_{i+1}, x_i)$, we have

$$x_{i+1} = a \cdot x_i + N_i \cdot p$$

and therefore, if we have $k$ pairs of the known elements $(x_j, x_k)$ then we will also have $k + 2$ unknowns, i.e., $a$, $p$, and $N_i$ for $i = 1, 2, \ldots, k$

# Cryptographic Strength of LCGs

- Still, equations of the form $x_{i+1} = a \cdot x_i + N_i \cdot p$ can be solved using lattice reduction techniques, and therefore, we do not have strong assumptions of cryptographic strength
- There is also practical constraint in a LCG with all three parameters $(a, p, x_0)$ are considered as the key
- We know that $p$ has to be a prime and $a$ has to be a primitive element of the group, that means a key generation algorithm has needs to incorporate these properties and generate such keys
- On the other hand, in a LCG with fixed parameters $(a, p)$ we need not worry about key with special properties — the only key, the seed $x_0$, is just a random integer: any integer would be fine; also, since $b = 0$, the only "bad seed" is 0, and easy to avoid

# GLIBC `random()`

- The GNU C library's `random()` function is a LCG with three steps
- The first step is based on the prime modulus $p = 2^{31} - 1$ and the primitive element $a = 16,807$
- Given the seed value $s$, the first step computes 33 elements $x_1, x_2, \ldots, x_{33}$:

$$
\begin{aligned}
x_0 &= s \\
x_i &= a \cdot x_{i-1} \pmod{p} \quad \text{for} \quad i = 1, 2, \ldots, 30 \\
x_{31} &= x_0 \\
x_{32} &= x_1 \\
x_{33} &= x_2
\end{aligned}
$$

# GLIBC `random()`

- The second step is based on the addition operation mod $q = 2^{32}$
- In the second step, new $x_i$ values are computed for $i = 34, 35, \ldots, 343$

$$x_i = x_{i-3} + x_{i-31} \pmod{q} \quad \text{for} \quad i = 34, 35, \ldots, 343$$

- In the final step, the output values are generated using the previous mod $q$ addition operation and the logical right shift operation $(\cdot)_{rs}$ as follows

$$\begin{aligned} x_i &= x_{i-3} + x_{i-31} \pmod{q} \quad \text{for} \quad i \geq 344 \\ r_j &= (x_{j+344})_{rs} \quad \text{for} \quad i \geq 0 \end{aligned}$$

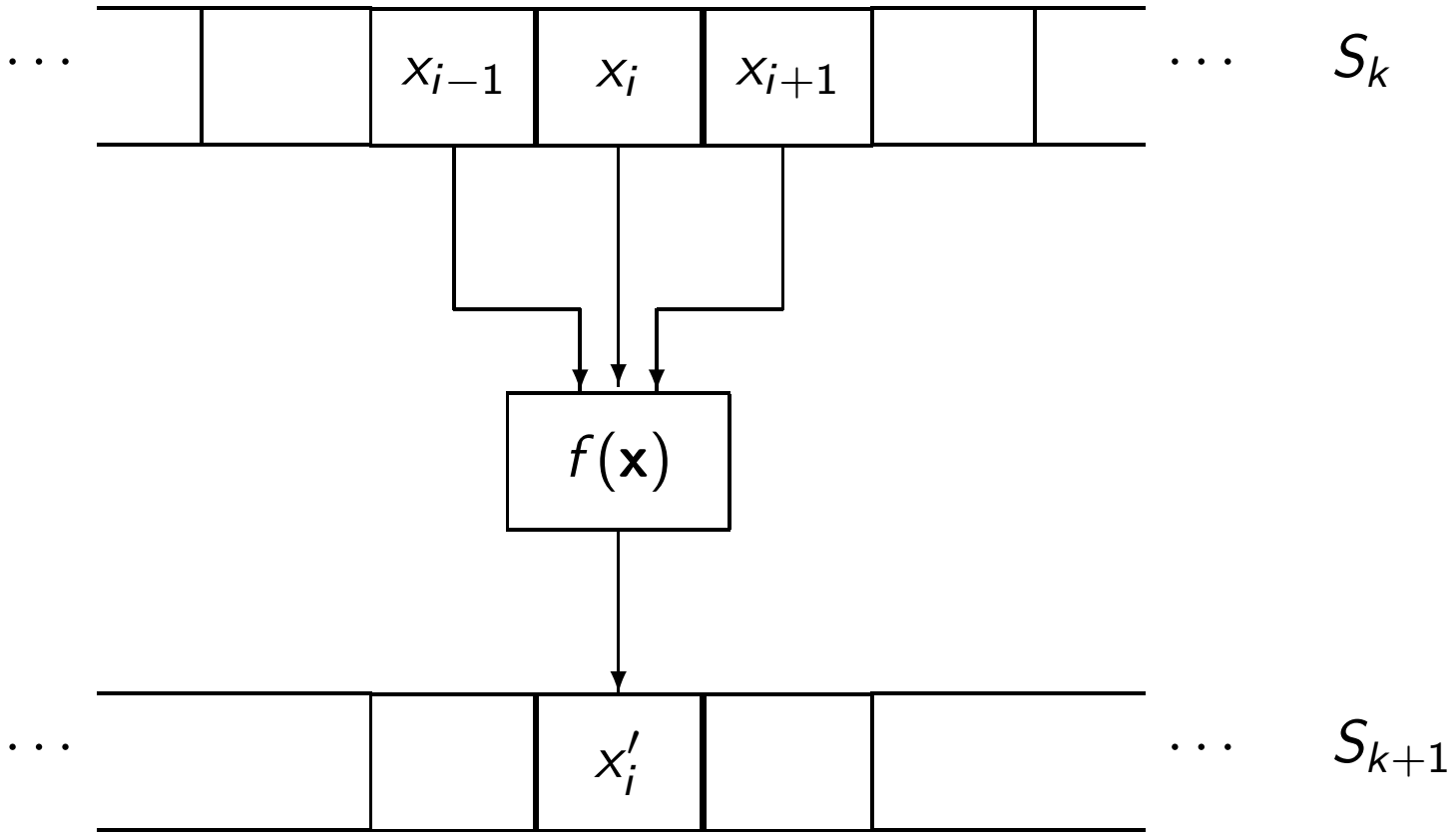
- Inversion: Two consecutive different moduli and the right shift make the inversion more difficult, however, since there are $2^{32}$ different seed values, exhaustive search is possible

# Cellular Automata

- A one-dimensional cellular automaton consists of a linearly connected array of $n$ cells, each of which takes the value of 0 or 1, and a boolean function $f(\mathbf{x})$ with $q$ variables
- The value of the cell $x_i$ is updated in parallel (synchronously) using this function in discrete time steps as $x_i' = f(\mathbf{x})$ for $i = 1, 2, \ldots, n$
- The boundary conditions are usually handled by taking the index values modulo $n$, i.e., the linearly connected array is actually a circular register
- The parameter $q$ is usually an odd integer, i.e., $q = 2r + 1$, where $r$ is often named the radius of the function $f(\mathbf{x})$; the new value of the $i$th cell is calculated using the value of the $i$th cell itself and the values of $r$ neighboring cells to the right and left of the $i$th cell

# Cellular Automata

The one-dimensional cellular automaton with $q = 3$.



# Cellular Automata

- Since there are $n$ cells, each of which takes the values of 0 or 1, there are $2^n$ possible state vectors
- Let $S_k$ denote the state vector at the automaton moves to the states $S_1, S_2, S_3$, etc., at time steps $k = 1, 2, 3$, etc
- The state vector $S_k$ takes values from the set of $n$-bit binary vectors as $k$ advances, and the state machine will eventually cycle, i.e., it will reach a state $S_{k+w}$ which was visited earlier $S_k = S_{k+w}$
- The period $w$ is a function of the initial state, the updating function, and the number of cells

# CA30 – A Random Updating Function

- Cellular automata are generally considered as discrete dynamical systems, or discrete approximations to partial differential equations modeling a variety of natural systems
- Wolfram proposed a random sequence generator based on the one-dimensional cellular automaton with $q = 3$ and the so-called CA30 updating function

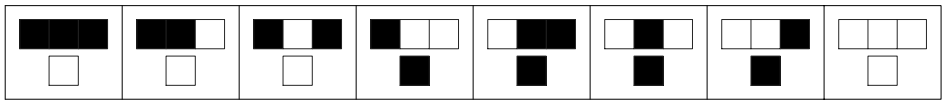
$$f(x_{i-1}, x_i, x_{i+1}) = x_{i-1} \oplus (x_i + x_{i+1})$$

  where $+$ is the boolean OR and $\oplus$ is the exclusive OR function
- The state vectors produced by this cellular automaton seem to have randomness properties, e.g., the time sequence values of the central cell shows no statistical regularities under the usual randomness tests

# CA30 – A Random Updating Function

- The truth table for the CA30 function $x_{i-1} \oplus (x_i + x_{i+1})$

| $x_{i-1}$ | $x_i$ | $x_{i+1}$ | $x_i'$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

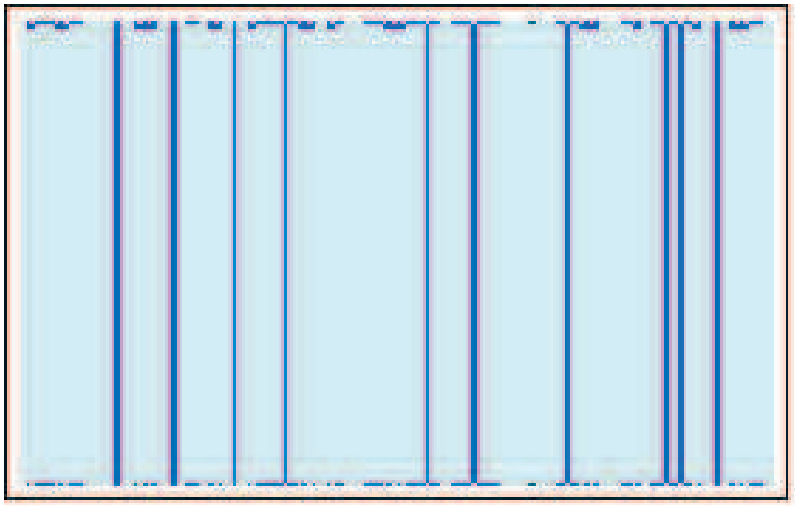
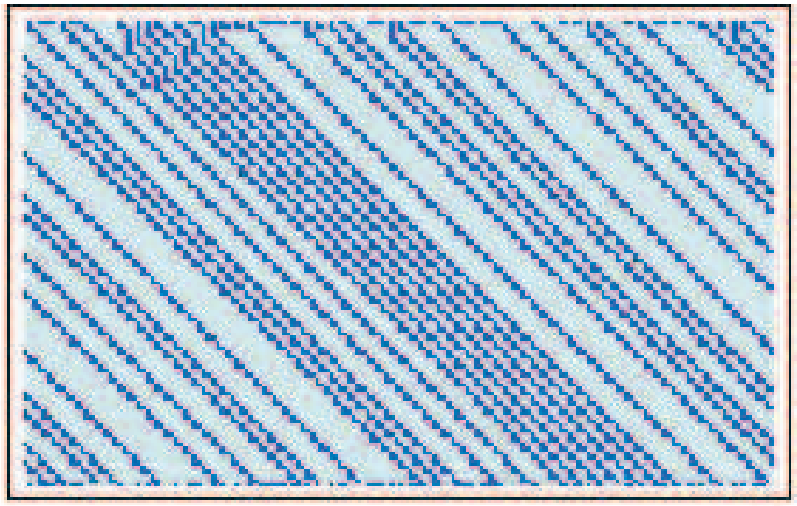
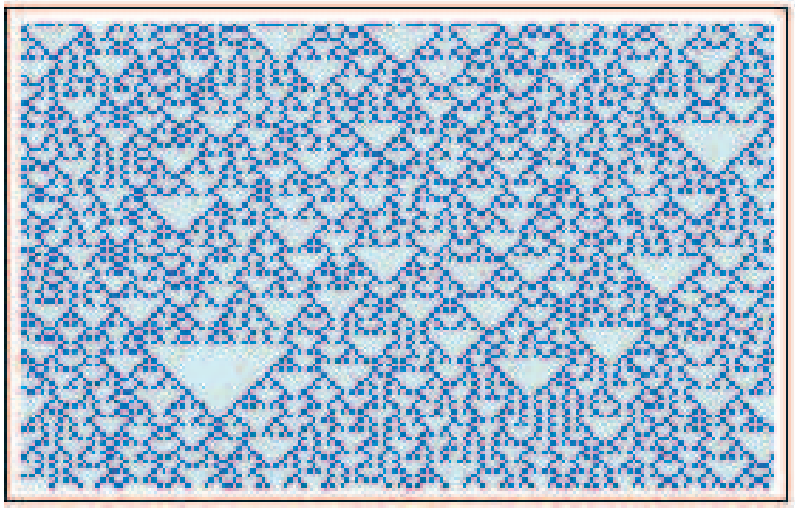
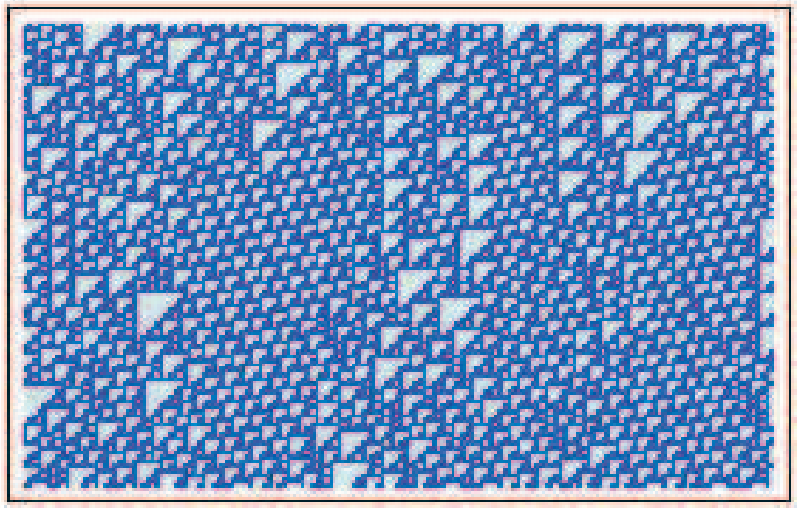


- The binary expansion of the integer 30 = (0001 1110)
- For $q = 3$, we have $2^{2^3} = 256$ different CA functions: CA0 .. CA255

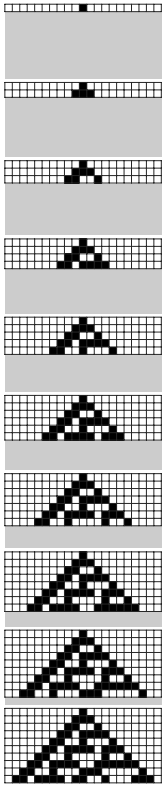
# 1-Dimensional CA Functions

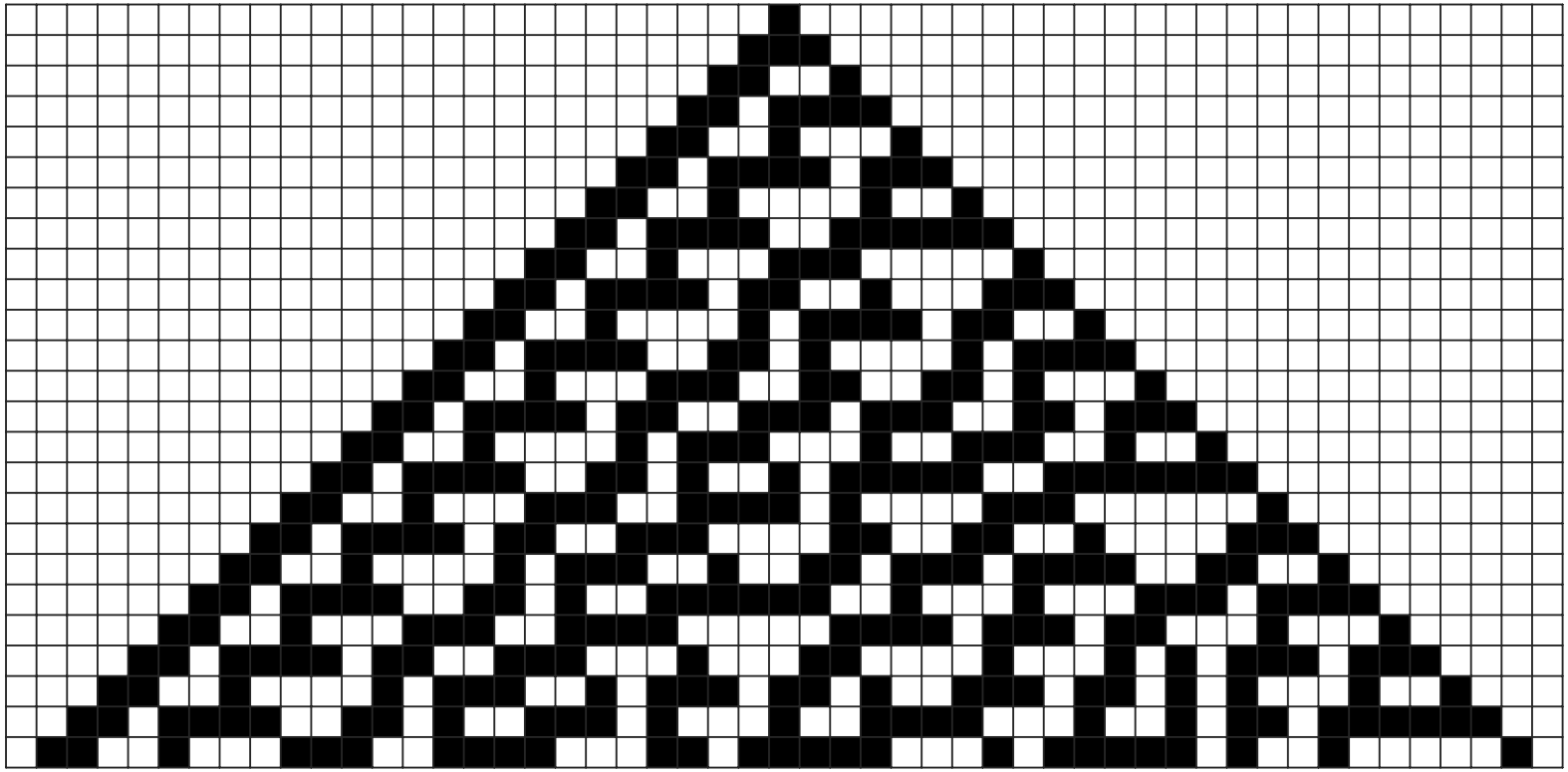
- Wolfram analyzed the behaviors of all them, starting from different initial conditions
- There seems to be 4 classes of behaviors
  - Class I Homogeneous: Everything eventually dies (or eventually lives forever). Some initial transient behavior usually precedes this final state
  - Class II Periodic: Perhaps after some initial transients, the pattern repeats itself exactly, in space (horizontally), in time (vertically), or both
  - Class III Chaotic: Patterns grow in a chaotic fashion: short-lived islands of order and sensitivity to initial conditions
  - Class IV Complex: Patterns grow in a complicated way, with both local stable behavior (acting as memory) and long-range correlations (acting to transmit data)

# 1-Dimensional CA Functions

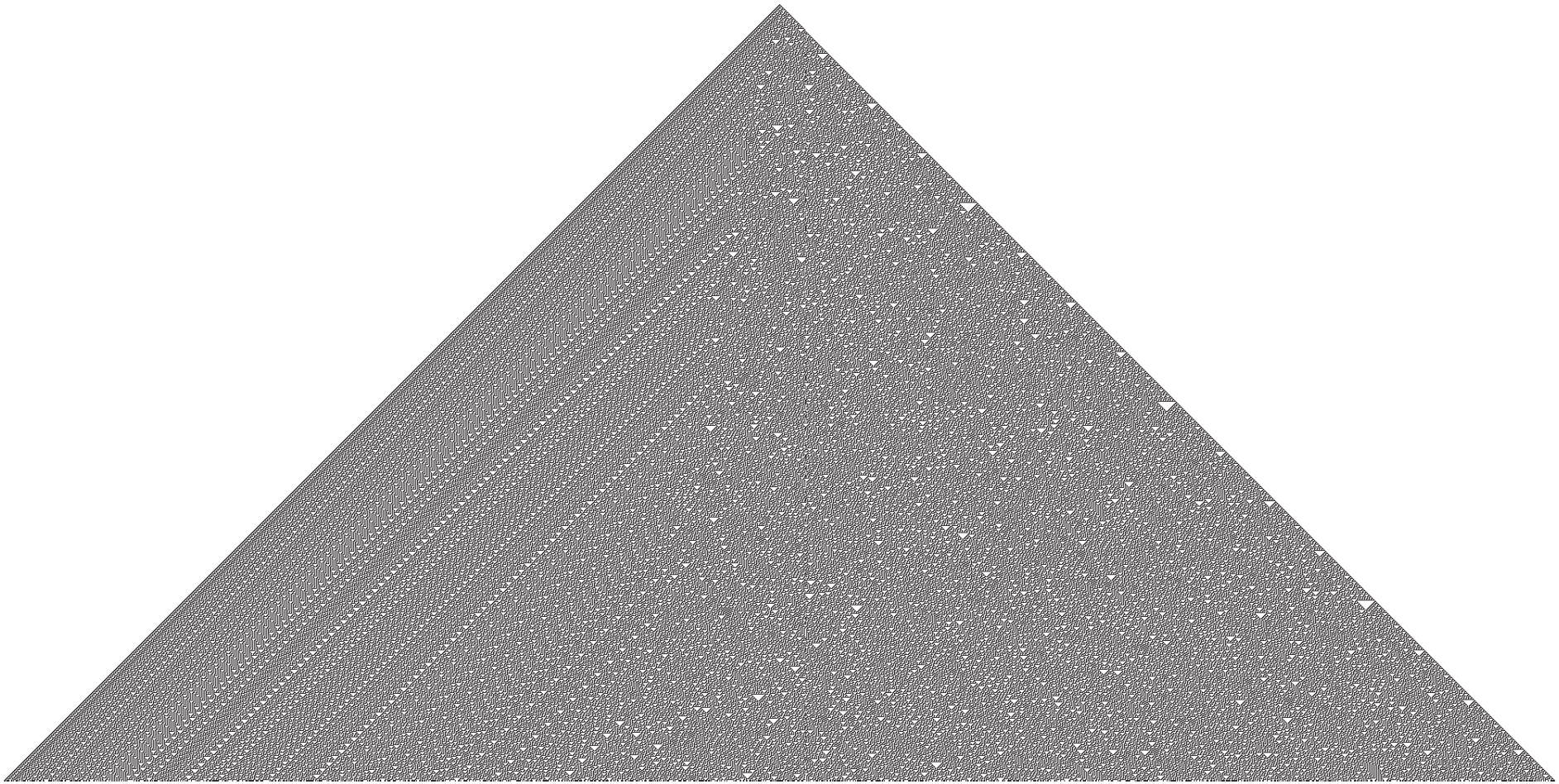





# CA30 Behavior





# CA30 Behavior



# Security of CA-based RNG

- CA30 and many other CA functions satisfy R1
- Question: Do they satisfy R2?
- In order to use such a generator for cryptographic purposes, we must also ensure that the seed value (the initial state vector $S_0$) is difficult to construct given a sequence of state vectors
- It was claimed that this problem is in the class NP – no systematic algorithm for its solution for an "unbounded" $n$
- However, in order to use CA as a RNG in a stream cipher, we need select a suitable updating function and a large $n$ (several hundred bits)
- We may not have cryptographic strength for some updating functions and for small values of $n$