Digital Random Bit Generation Using Cellular Automaton

Enrique Flores

May 5, 2025

Background

Conventional random number generators (RNGs) for cryptographic and blockchain systems often rely on external sources of entropy, complex statistical mechanisms, or heavy computational protocols. These approaches introduce dependencies, increase power consumption, and create vulnerabilities, especially for embedded and mobile environments.

Blockchain systems similarly suffer from consensus-driven architectures that require continuous connectivity, massive computational resources, and synchronization across globally distributed nodes.

There exists a need for a method of digital random bit generation and decentralized ledger maintenance that is deterministic, lightweight, embeddable, and capable of operating without external validation.

Summary of the Invention

This invention provides a system and method for:

- Generating high-quality random bitstreams using a deterministic cellular automaton (specifically Rule 30) operating on a 128-bit initial seed.
- Utilizing these deterministic random streams to create a continuously evolving chain of 128-bit states, forming the basis for decentralized transaction validation and ledger evolution.
- Enabling decentralized ledgers that operate without consensus protocols, without reliance on external timestamps, counters, or synchronized clocks.
- Allowing nodes to synchronize, validate, and merge ledgers based solely on deterministic comparison to a shared initial seed, enabling seamless operation even in offline or partitioned network environments.

Nodes generate their local blockchain state entirely based on the evolution of their internal MKRAND generator. Upon receiving external transactions, nodes compare the incoming block against their own deterministic chain history. If a match is found, or can be generated through forward evolution, the node accepts and merges the transaction without requiring external consensus, permitting resilient synchronization between periodically connected nodes.

Detailed Description

Each node maintains:

- A shared initial seed value.
- A deterministic random bit generator based on Rule 30.
- A local ledger structure storing validated transactions.

Random Bit Generator (MKRAND)

The bit generator uses a one-dimensional cellular automaton based on Rule 30 to produce a deterministic, high-entropy bitstream. The generation process operates as follows:

1. Initialize a 128-bit state with a predefined seed.
2. Evolve the cellular automaton for 256 generations, forming a 2-dimensional field of 128 columns by 256 rows.
3. Extract the center column (i.e., the 64th or designated middle cell's evolution across all rows) as the output of one generation cycle, resulting in a 128-bit random value.
4. Use the extracted 128-bit output as the seed for the next generation cycle.
5. Repeat indefinitely to produce a deterministic but practically infinite chain of random 128-bit outputs.

This method enables the generation of a collision-free chain of random 128-bit numbers deterministically derived from the initial seed. Each new 128-bit output represents a unique and non-repeating point in the state space, ensuring high entropy and perfect reproducibility without external entropy sources.

Cryptol Specification

Below is the full Cryptol specification, which can be downloaded from `https://github.com/GaloisInc/cryptol/blob/master/examples/contrib/mkrand.cry`

```
// Digital Random Bit Generator (MKRAND) based on Rule 30
module MKRAND where

type Seg   = [0x80]
type Field = [0x80]Seg

/* Canonical seed - a segment with a single True bit in the center */
seedUnit:Seg
seedUnit = (0 :[63]) # (1:[1]) # (0:[64])


/*
 *  Field - Unfold an application of Rule 30 to a seed
 */
field: Seg -> [inf]Seg
field s = new
  where
    new = [s] # [ rule30 row | row <- new]
    rule30 r = [ a ^ (b || c) | a <- r >>> 1
                              | b <- r
                              | c <- r <<< 1
               ]


/* SHA30 - Use the input segment as the seed, generate two square fields,
 * keep the center column of the second.
 */
sha30: Seg -> Seg
sha30 s = take`{0x80} (drop`{0x80} [ r @ ((((length r) / 2)-1):[8]) | r <- field s])
```

```
/*
 * RAND − Seed XOR (SHA30 Seed)
 */
rands : Seg −> [inf]Seg
rands s = rest
  where
     rand p = p ˆ (sha30 p)
     rest = [rand s] # [rand x | x <− rest]


/* Break segments into bytes */
randBytes : Seg −> [inf][8]
randBytes s = groupBy‘{8} (join (rands s))


/* XOR a byte string into a random byte string, using the given seed */
randXOR : {n} Seg −>  String n −> String n
randXOR seed src = [s ˆ r | s <− src
                          | r <− randBytes seed
                   ]
```

C Implementation of MKRAND

To examine the C implementation of MKRAND, download the code from:
`https://github.com/taguniversalmachine/MKRAND-1`
Then build it with:

```
make -f src/Makefile.simple
```

Here is the help screen you will see when executing `./mkrand -h`:

```
MKRAND - A Digital Random Bit Generator (DEBUG)
Copyright (c) 2013 TAG Universal Machine.

USAGE: mkrand [-s seed] [-f format] [-n blocks] [-o filename] [--profile] [--verbose]
Formats:
 0  -       Pure              434C928FE5F7C925A068054DD044E718                     128-bit Binary
 1  -       SHA1           f357d594-9faa-32b2-a460-043a2e5c783c                        SHA1 Format
 2  -     BINARY              ...00000010000011110100101100111                   Text Mode Binary
 3  -   RESERVED
 4  -       IPV4                                  96.168.68.21                       IPV4 Address
 5  -       GUID         {2B3F9801-95F8D686-A17F-71D58932F596}                 Globally Unique ID
 6  -       IPV6       e8be:1db8:8291:da30:2019:603e:47b8:925f                       IPV6 Address
 7  -   RESERVED
 8  -        PSI        [<:15D53A16B74B0C9E0EDC423576F85241:>]                   Time Fingerprint
 9  -   RESERVED
10  -        INT                                    2766294806            32-bit Unsigned Integer
11  -    UUID V4          b710cd46-9e45-4f53-ab79-f207c15a2511              Universally Unique ID
12  -     BASE64                        e08BJk8OH4U_nscNjAATXA                      Text Encoding
13  -   RESERVED
```

Here is an example of some entropy produced by this utility:

```
% mkrand -f8 -n10
[<:2F9BF1E4E23B8BFF29856910FF5AD61E:>]
[<:047A9A0D46538466A56A8FAE05081D1D:>]
[<:29206E26929395565560B0B3176E5552:>]
[<:5BDF7D0166F684FBD7015B30AA1CE90E:>]
[<:B29581E95CDEDE5E087A90EC4BD657ED:>]
[<:6ACB637019D91F7FF41EE8ED082C919C:>]
[<:C444C4CF3335E645C77F790EFB5E9D55:>]
[<:1900F73EA129AF9A50EC5FE4C49BBE9D:>]
[<:4419BCB82F1077EA0EC3DC69477E94DD:>]
[<:53F99033150119CBA9D67525AD1B726E:>]
```

Each block, referred to as a PSI index, identifies a unique point in space and time with a 128-bit digital number. The blocks are formed into chains; each block is computed from the previous one.

Machine-like precision means any chain can be perfectly regenerated from any block.

Here is an example of a new chain created from the fourth-from-last block in the above chain:

```
% mkrand -f8 -n10 -s "[<:C444C4CF3335E645C77F790EFB5E9D55:>]"
[<:1900F73EA129AF9A50EC5FE4C49BBE9D:>]
[<:4419BCB82F1077EA0EC3DC69477E94DD:>]
[<:53F99033150119CBA9D67525AD1B726E:>]
[<:8C55EEDA7C6208DD126EA61D279141F6:>]
[<:C5F1C39C36DBA146889444E89E963DE5:>]
[<:640F97F111D89F79FD745E04A2EA8C24:>]
[<:551852E493924494BA17EF08F8447D49:>]
[<:FFE976344A6CC5D2AB6E1E479ABA9EF3:>]
[<:2DE82C85AC76F91C394693723D62A005:>]
[<:C0F70220F54BA33DF4916CC29CFFD631:>]
```

As you can see, the utility was able to regenerate the chain from that given block, and could take over the generation for practically infinite periods. Effectively, an infinite size memory block is pointed to by each PSI Index.

Blockchain Operation

Transactions are:

- Locally generated and validated against the node's internal deterministic chain, derived from the shared seed block.
- Sequenced based on the evolution of the deterministic chain itself, without reliance on external timestamps or counters.
- Validated upon receipt from another node by matching the received block against the locally generated chain.
- Nodes can either search backward through their chain to find a matching block, or evolve their chain forward until a match is generated.
- This allows nodes to synchronize, fall behind, or catch up without any loss of continuity or ordering, enabling seamless operation between periodically connected or offline nodes without requiring consensus mechanisms.

Advantages

- Eliminates need for external randomness sources.
- Reduces power and bandwidth consumption.
- Enables true offline operation of distributed ledgers.
- Improves scalability and resilience in mobile and IoT contexts.

Comparison to Existing Blockchain Technologies

This section presents a comparative analysis between the MKRAND-based blockchain system and conventional blockchain technologies such as Bitcoin and Ethereum, in terms of energy efficiency and transaction speed.

Step 1: Estimate MKRAND Energy per Block

In MKRAND:

- Each evolution step updates 128 bits based on Rule 30 logic.
- The field evolves for 256 generations to produce a new block.

Thus, total operations per block:

$$128 \times 256 = 32,768 \text{ operations}$$

Assuming each logic operation requires approximately 0.1 picojoules (pJ) (based on modern ASIC/FPGA switching energy), the total energy per block is:

$$32,768 \times 0.1\,\text{pJ} = 3,276.8\,\text{pJ} = 3.28\,\text{nJ}$$

Therefore, each MKRAND block requires approximately **3.3 nanojoules**.

Step 2: Estimate Bitcoin Energy Usage

Bitcoin mining requires massive computational effort for proof-of-work:

- Average energy usage per Bitcoin transaction: between 500 kWh and 1,000 kWh.
- Using 500 kWh as conservative estimate:

$$500 \times 3.6 \times 10^6 = 1.8 \times 10^9 \text{ joules}$$

Thus, each Bitcoin transaction consumes approximately **1.8 billion joules**.

Step 3: Energy Efficiency Ratio (Bitcoin vs MKRAND)

$$\frac{1.8 \times 10^9}{3.3 \times 10^{-9}} \approx 5.45 \times 10^{17}$$

MKRAND is approximately **545 quadrillion times more energy efficient** than Bitcoin.

Step 4: Estimate Ethereum Energy Usage

After transitioning to Proof-of-Stake:

- Average energy usage per Ethereum transaction: approximately 0.03 kWh.

$$0.03 \times 3.6 \times 10^6 = 108,000 \text{ joules}$$

Thus, each Ethereum transaction consumes approximately **108,000 joules**.
Energy Efficiency Ratio (Ethereum vs MKRAND):

$$\frac{108,000}{3.3 \times 10^{-9}} \approx 3.27 \times 10^{13}$$

MKRAND is approximately **32 trillion times more energy efficient** than Ethereum.

Step 5: Transaction Speed Comparison

- MKRAND: 256 steps at 200 MHz clock rate:

$$\frac{256}{200 \times 10^6} = 1.28 \times 10^{-6} \text{ seconds}$$

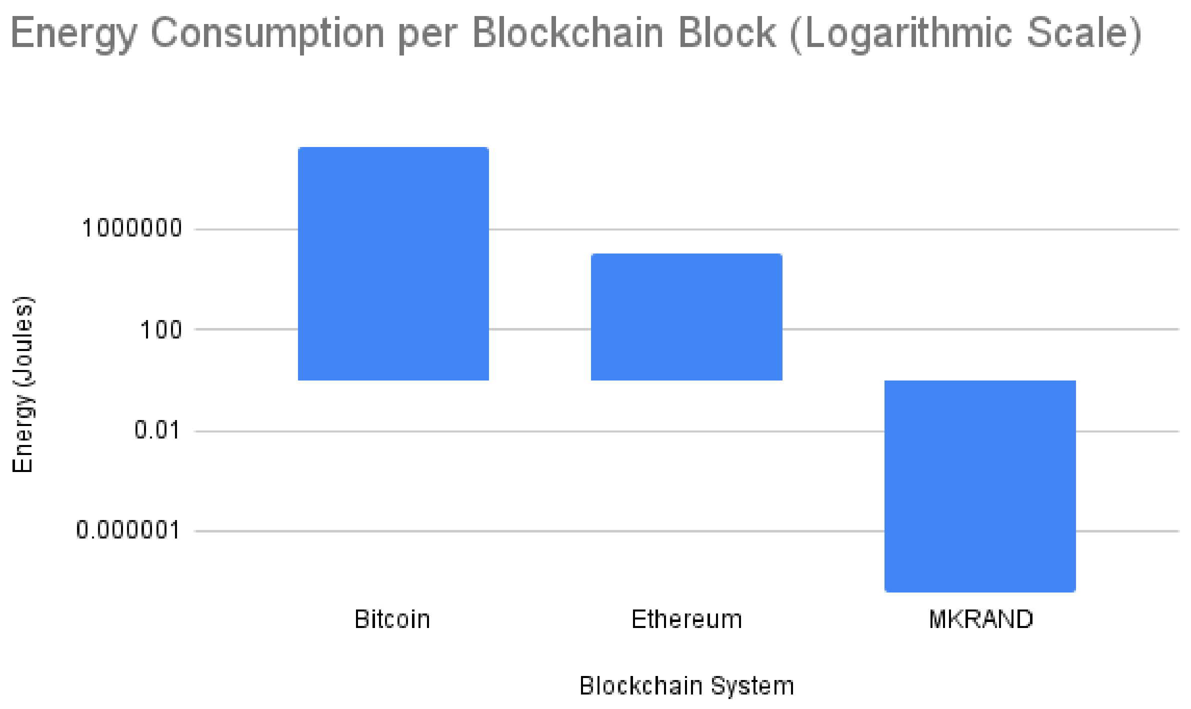
Approximately **1.3 microseconds** per block.

- Bitcoin: 10 minutes per block.
- Ethereum: 12 seconds per block.

Thus, MKRAND achieves between **10 million** and **500 million times faster block generation**.

Summary

System	**Energy per Block**	**Block Time**	**Relative Efficiency**
Bitcoin	∼1.8 billion joules	10 minutes	545 quadrillion ×
Ethereum	∼108,000 joules	12 seconds	32 trillion ×
MKRAND	∼3.3 nanojoules	1.3 microseconds	N/A



Conclusion

The MKRAND blockchain system demonstrates orders of magnitude improvement in both energy efficiency and transaction speed compared to conventional blockchains. By eliminating consensus mechanisms, mining, and operating on minimal digital logic without operating system overhead, MKRAND enables scalable and sustainable decentralized computing for embedded, mobile, and offline applications.

Provisional Claims

1. A method for generating deterministic random bitstreams comprising:
    - initializing a one-dimensional cellular automaton with a 128-bit seed;
    - evolving the automaton for 256 generations to produce a 2D field;
    - extracting the center column of the bottom half of the field as a 128-bit output;
    - feeding the output back as the seed for the next generation cycle.
2. The method of claim 1, wherein the cellular automaton follows Rule 30 evolution rules.
3. A system comprising:
    - a hardware or software implementation of the method of claim 1; and
    - a ledger or state system synchronized entirely via the deterministic bitstream.
4. The system of claim 3, wherein the hardware implementation uses FPGA, ASIC, or embedded micro-controller devices.