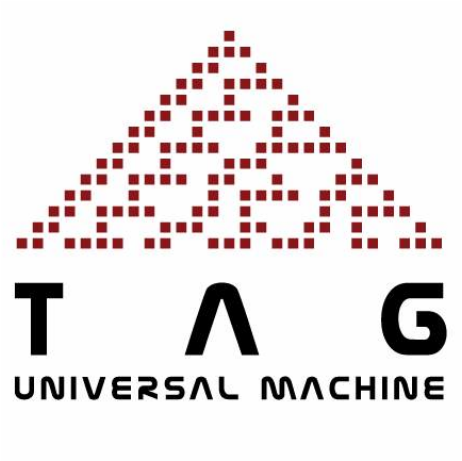

0.1 Digital Blockchain Introduction

Generating random information is a crucial activity for computers, since there are many computations that require certain probabilistic behaviors. Until recently, the conventional mechanism for generating random information has been to collect environmental entropy in the form of clock-jitter, temperature variations, and junk data. While adequate for generating small sequences of unpredictable information, it is an ad-hoc and non-repeatable mechanism, and as such cannot be engineered into digital systems. To gain predictability, software developers have resorted to feeding small bits of environmental entropy into complex mathematical equations that introduce repeatability to some extent, but quickly diverge from predictability after short runs due to the inherent inaccuracy in floating-point calculations. This goes to the heart of the reason why non-digital blockchain technologies need to maintain a centralized ledger - given identical software, two computers generating blocks with mathematical methods will eventually diverge due to environmental factors like CPU latency, temperature differences, or rounding effects in their disparate floating-point hardware. This explains why these technologies have had to develop baroque consensus mechanisms to determine truth, resulting in huge expenditure of energy and time replicating highly inefficient computations across many computers simply to generate and agree on the next block.

And by the same token, this is why these technologies cannot operate offline, because a node doing operations on its version of the blockchain would quickly diverge from the consensus ledger and be pruned, were it to operate for any significant time without communication with the rest of the nodes. A digital blockchain, however, does not require environmental entropy and is not dependent on any outside source for accurate generation of blocks. Additionally, because of the 128-bit address space it is operating under, any number of devices can branch off and operate indefinitely on their own, and if at any point they re-connect to some authority, all of their offline transactions can be seamlessly

merged into the larger order without losing any time-ordering information or validity.

This ability to function without environmental dependencies and the vast address space offers a significant advantage over non-digital blockchain technologies. It eliminates the need for baroque consensus mechanisms and reduces the energy and time spent on inefficient, irrelevant, and redundant computations. Additionally, a digital blockchain can operate offline, empowering nodes to perform operations independently without diverging from a local consensus ledger.

0.2 Technology Overview

MKRAND works by employing a cellular automaton called "Rule 30", which is detailed in the book A New Kind of Science. Unlike mathematical methods, it uses a mechanism that can be directly implemented digitally.

Here is a Cryptol specification:

```
/*

MKRAND - A Digital Random Bit Generator

The MIT License (MIT)

Copyright (c) 2014, TAG Universal Machine.

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

```
--

USAGE

Create a 128 bit seed:

seed = seedUnit

Once the seed is created, you may use it to generate a stream of random bits:

take `{100} (randBytes seed)

Here a string is encoded with seedUnit, using the deterministic random stream as a
one-time pad against which to XOR the string:

Encode:
randXOR seedUnit "Deus Ex Machina"
[0x28, 0x2b, 0x2c, 0xfa, 0x92, 0xca, 0xb3, 0xcb, 0xed, 0x50, 0xc2,v0x1b, 0x11, 0x0e, 0x70]

Decode:
:set ascii=on
randXOR seedUnit [0x28, 0x2b, 0x2c, 0xfa, 0x92, 0xca, 0xb3, 0xcb, 0xed, 0x50, 0xc2,0x1b, 0x
"Deus Ex Machina"

*/

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 ^ (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 ^ r | s <- src
| r <- randBytes seed
]

/* OTP example */

property otp_encdec =
randXOR seedUnit "Deus Ex Machina" == c
/\ randXOR seedUnit c == "Deus Ex Machina"
where c = [ 0x28, 0x2b, 0x2c, 0xfa
, 0x92, 0xca, 0xb3, 0xcb
, 0xed, 0x50, 0xc2, 0x1b
, 0x11, 0x0e, 0x70]

otp_involutive : {len} (fin len) => String len -> Bit
otp_involutive msg = randXOR seedUnit (randXOR seedUnit msg) == msg

property otp_involutive_32 msg = otp_involutive (msg : String 32)
```

The specification allows you to load the algorithm into cryptol to examine its properties. You will find it in examples/contrib/mkrand.cry or at `https://github.com/GaloisInc/cryptol/blob/master/examples/contrib/mkrand.cry`

0.3 C implementation

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              6C87401A18921096A5ABEE1B6F98B192                     128-bit Binary
1  -       SHA1          ed60c1ab-b0ca-34b6-a8c5-61482b32aeaa                       SHA1 Format
2  -     BINARY              ...10000110101100011011000011100                   Text Mode Binary
3  -   RESERVED
4  -       IPV4                                  165.144.212.67                     IPV4 Address
5  -       GUID         {94B63872-2B769783-A012-9CD27AE093E4}                 Globally Unique ID
6  -       IPV6       641e:349e:f60b:0a38:c8e4:b47d:ce1f:0f70                      IPV6 Address
7  -   RESERVED
8  -        PSI        [<:0B5E17A5689FC3671E1FC962976D8C6D:>]                   Time Fingerprint
9  -   RESERVED
10  -        INT                                     133409084            32-bit Unsigned Integer
11  -    UUID V4          ca5c7eca-6b16-4ede-a492-9c2f8885a1bf               Universally Unique ID
12  -     BASE64                          w80m4AMgPgMY_kY9dYMS0g                      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 4th 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. So effectively, an infinite size memory block is pointed to by each PSI Index.

0.4 Security and Encryption Applications

The benefit of using a cellular automaton, as opposed to gathering entropy from the computer's environment, is that it can be implemented with simple digital logic, without requiring an Arithmetic Logic Unit. Conventional random bit generators employ elaborate mathematical transformations in an attempt to generate unpredictable information. This is a very energy intensive process that does succeed in generating fairly unpredictable information, however at great cost and with some serious defects. The process being fairly ad-hoc actually doesn't produce a known amount and quality of entropy for a given time and effort, so if you look at the on-board RBG employed by modern CPU's for example, there are mechanisms that bind up for lack of sufficient environmental entropy, so the top-level software that employs this mechanism does not have a predictable bit-stream and in fact has to be prepared for low quality or even total lack of entropy. Modern exploits measure timing variation in CPU workload to reverse-engineer keys, by observing clock jitter and performance variations during key generation, these byproducts of math-based RBG's can even affect overall power consumption and recent exploits can extract keys by observing status LED's on devices involved in key generation. A separate issue comes to light when trying to use these techniques over large computational spaces, for example in simulations, it is important in the scientific process to be able to run a simulation many times while holding certain parameters steady, however this is not possible with math-based RBG's, because for a million+ particle count simulation using floating point math, the rounding process makes it very difficult if not impossible to re-run a simulation for a long period and arrive at the exact same end-condition every time, so the random bits can be categorized as random but not deterministic. Determinism in random bit streams allows

for them to be used in streaming encryption, as can be seen in the Cryptol spec, a practically infinite-size bit stream can be reliably encrypted in a streaming bit-for-bit manner with none of the issues that would come up when attempting to do this with floating-point RBG's.

0.5 Entropy - A Definition

In Claude Shannon's Mathematical Theory of Communication, entropy is defined as:

Entropy is defined by $H = -K \sum_{i=1}^{n} p_i \log p_i$ where K is a positive constant. To test the entropy produced by MKRAND, run it with the –profile argument.

```
% mkrand --profile
139.17 kbps  Entropy (H): 1.00000
```

0.6 Conventional Blockchain technologies

In the article *Blockchain and our planet: why such high energy use?* [1], the energy consumption of blockchain technologies is detailed in the section **Endless Growth and Burden-shifting**:

In other words: computers joining the mining do not change the functionality of a blockchain, but only increase its energy use. The law of diminishing returns plays a big role here. Once a blockchain network reaches a critical number of nodes, security already meets a base requirement. But mining is lucrative, and more and more people own cryptocurrency. As a result, the Bitcoin blockchain alone currently uses 204,5 TWh of electricity per year, comparable to the power consumption of Thailand. And it's the amount of energy used itself that is the problem, not the source of that energy. Many miners are switching to renewable energy sources. But this simply moves the problem elsewhere. We don't yet have enough renewable energy production to cover all of our activities. So, if mining uses up renewable energy, that just increases the non-renewable energy used on other activities.

0.7 A Digital Economy at Any Scale

Economies are energy distribution networks. Currently, economics operates at the human level, with the exchange of currencies, but current implementations of crypto-currencies have been unable to penetrate the computational space, where they would be most useful. All of the non-digital blockchain implementations require access to massive bandwidth and processing power to operate, so they remain the plaything of speculators with excess money and time. To be actually useful, a crypto-currency must be usable by computers themselves, without human intervention, and they must be able to operate in the Internet of Things space, where all future growth is.

Imagine a scenario with three low-power devices operating in a remote region with only sporadic access to the network. One device has a solar panel, and a way to store the generated electrons, while the other two devices are electrically connected to

[1] https://pre-sustainability.com/articles/blockchain-and-our-planet-why-such-high-energy-use/

the energy producer and consume electricity for their own work. Currently, there is no efficient way to measure and meter the exchange of value between these devices, engineers must design the producer for the worst-case power draw, and the consumers for the highest efficiency possible, but the software has very little visibility into the electrical economy of the set of three devices. With a digital blockchain implemented on all three devices, a simple economy can quickly be activated by consensus among the devices.

The producer device announces to its clients that it can generate, say, 200 milliAmps per second. Once that announcement goes out, the two clients can then say something like "I, Device A, need 100 mA continuously", and "I, Device B, require 400 mA periodically for approximately two seconds". The producer device then does a quick computation and says "Ok, in order to service both clients, I must feed Device A half my budget, and accumulate electrons for when Device B creates demand, so that both have their needs satisfied without overloading my circuits."

The producer device then announces, "Ok, I am selling blocks of 100 mA/seconds of electricity for 1 digital block each". Now, the client devices know they must generate digital blocks in proportion to their energy usage, and device A can easily pay for its needs because each block only costs it 1 mA/microsecond to compute, and Device B knows it must accumulate 8 blocks and have them ready to pay for when its time comes to consume two seconds of electricity at 400 mA. Now, electricity and blocks flow through the network in a self-regulating manner and without complex arbitration. A device that is malfunctioning and consuming electricity outside of normal bounds will quickly go "bankrupt" and electrical energy will cease flowing to it. Perhaps a free-tier of energy is periodically available for it to wake up and attempt self-repair or produce work at a slower pace without bringing down the rest of the system.

Now say you have thousands of these devices in close proximity, the digital blocks can flow from one group to another without any complex arbitration or currency conversion, and the aggregate field of devices can then export the excess to larger economies, perhaps together they have hundreds of watt-hours of surplus energy, or equivalently, hundreds of megabytes of surplus blocks, this is now enough to sell back to the power company, where the electrical energy or digital blocks can be converted into dollars or back into electricity for the power company to use in the larger economy or some other network. All of these transactions and changes in scale operate in the same way and with the same low power.

The reason the above scenario is possible is because this technology provides a perfect hashing function, which will shortly be explained, but essentially allows these devices to conduct economic transactions amongst each other without access to the internet, and yet if at some point they do gain connection, all the transactions can then flow out to the wider economy without concern that some transaction outside of their knowledge is in conflict with any transaction that was made during the time they were offline. This is why Bitcoin and all the public-ledger crypto-currencies cannot operate offline, because they must post their transactions in a reasonably quick manner to the public ledger otherwise the transaction sequence will be orphaned and pruned by the mechanism, if a device goes online years later with a transaction done while offline, it is unlikely the public ledger blockchain would spend too much energy trying to re-incorporate it onto the ledger, it would involve backtracking and notifying every client on the planet and it is just not feasible, so it will just be considered a junk transaction.

0.8 Collision-free Hashing

Hashing is a computer science term for mapping one number system onto another, and is a very common operation to do. Let's say you have a potato farm and are sorting the harvested potatoes for shipment to the distributor. You have a large number of potatoes, say hundreds of thousands, and a small number of grades or categories that they are sorted into, so for instance a large potato with no blemishes would be Grade A, a smaller one with a couple bruises might be Grade B, and so on. The hashing function in this case is an observation of important features of the potato, such as weight, approximate dimensions, surface quality, color, etc. A hash collision happens when two different potatoes have similar enough characteristics that they both get sorted into the same grade. This is perfectly fine and part of the process of categorization. There are instances, however, when your system requires that each potato go into its own distinct bin and it would be a costly mistake to have two different potatoes go into the same bin. An example of the need for a perfect hashing function is a bank, who takes input data about a client, such as their name, social security number, driver's license, address and maps it into a unique identity which then is linked to a unique account number. It would be a costly mistake to have two customers with similar names linked to the same account, at that point their funds would be intermingled and it would be a painful and slow exercise in untangling the assets from each other and putting them into the correct bin. A perfect hash function always maps the same criteria into the same bin, and further, will never map two different inputs, no matter how similar, into the same bin. So let's expand the potato example to grains of sand now. We have a beach somewhere and we want to take an electron micrograph of each and every grain of sand and store it under the visible criteria that identifies each grain, and furthermore, it would be disastrous to have two grains of sand map to the same image. So like in the potato example, we take some surface identifiers of the grains of sand and map them into a set of bins, being our available memory to store the picture. We could take for instance the GPS location of the sand grain down to the square millimeter, its depth down to the millimeter, its weight down to the microgram, its color, its temperature, eventually we would have very precise location and identifier data looking something like this:

```
Grain X
Location: Palma Mallorca Spain @38.5015772,3.1388132,8.29z
Depth: 11.34 mm
Weight: 0.0647989 gram
Color: \#C2B280
Temperature: 114.34 degrees Celsius
```

Micrograph: Figure 1

To figure out how much information we have, we take the Log 2 of all the numeric data and add it up, so for instance, the first component of the GPS location, 38.5015772, yields 5.267 bits, the second component yields 1.6502 bits, and the third yields 3.0514 bits, so to encode the position we require 9.9686 bits, but since we only have whole bits in computers, we need to round up to 10 bits. Similarly, we need 3.5 bits for the depth, 4 bits for the weight, 24 bits for the color, and 3.5 bits for the temperature, for a grand total of 45 bits to uniquely identify the grain of sand. Now say we want to store its micrograph which is 13,813 bytes, into some place in

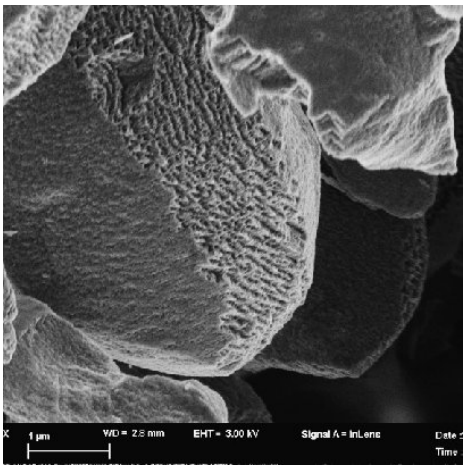

Figure 1: Grain X

memory, so that when we give the computer the above identifier data, we get the correct picture and not any other. We want to map the entire beach and we want to be able to quickly access the already ingested data while the mapping process is underway. A conventional computer with a conventional non-digital hashing function would start out fine at first, but once a few million grains of sand were input, it would start to struggle with memory issues, and at 8 billion grains of sand per cubic meter, it would probably fall over before mapping the entire beach, and retrieval times of the ingested data would be very slow as it would have to move terabytes of data in and out of memory to look for the requested images, whereas a memory controller using this technology can index into the correct block in microseconds, the block being any digitally addressable asset, either physical memory, hard disk block or an IPV6 address out on the internet, without the need to pull up unrelated data in a painful and expensive tree search like a database. A perfect hashing function implemented digitally would map the data as fast as it can be ingested onto an address space large enough to prevent any collisions from happening, it would also map the criteria as fast as it can be queried and return the memory location of the associated image, and that happens to be exactly what this technology does, it maps 128 bits of arbitrary data into and out of a perfectly uniform 128-bit address space.

128 bits is a big number. From the Medium article `shorturl.at/aBFY9`. Most people know that 128 is a BIG NUMBER but don't comprehend exactly how big it is. Outside of a few disciplines such as cryptography and physics, most people will never come across a number that large. Most cryptographic algorithms deal with numbers that are 128 bits or larger. A 128-bit number has 2^{128} possible values, but how big is 2^{128}?! A 128-bit number has 2^{128} possible values. That means $2 \times 2 \times 2 \times \ldots \times 2$ (128 times).

We can also write it as: $2^{64} \times 2^{64}$ or as $2^{32} \times 2^{32} \times 2^{32} \times 2^{32}$.

It's important to understand that 2^{128} is not twice as big as 2^{64}; it is 2^{64} times as big. If you take 2^{64} and double it, you only get 2^{65}. Let's see what these numbers look like when we write them out as numbers:

$2^{32} = 4,294,967,296$.

$2^{64} = 18,446,744,073,709,551,616$.

$2^{128} = 340,282,366,920,938,463,463,374,607,431,768,211,456$.

2^{32} is only about 4.3 billion, which is a little less than the number of people alive today.

2^{64} is about 18.4 quintillion, which is completely outside normal human experience.

2^{128} is 340 undecillion, and I had to look that up because I had no idea what the number is called.

So the 128-bit number space is inconceivably large, large enough that we can say with confidence that mapping some input data uniformly into that space, there is zero possibility of collision simply because the probability is so small as to require continuous hashing for longer than the known age of the universe for a collision to be an even measurable possibility. Let's imagine a simple 128-bit counter.

Let's assume the counter takes 1 millisecond (1 ms) to increment by 1 count. The maximum 128-bit integer is 2^{128} - 1. Now, let's calculate the total time required to count through that space:

Total Time = Number of Increments × Time per Increment

Number of Increments = MAXINT + 1 (since counting starts from 0)

Time per Increment = 1 ms = 0.001 seconds

Now, plug in the values and calculate:

Total Time = (2^{128} - 1 + 1) × 0.001 seconds

Total Time = (2^{128}) × 0.001 seconds

Total Time ≈ 3.4028237×10^{26} seconds

This is an extremely large number! To put it into perspective, it is about 10^{18} times longer than the current age of the universe (estimated to be around 13.8 billion years). So, counting up to the MAXINT with a 1 ms increment rate is practically impossible with current technology and the age of the universe.

0.9 Centralized blockchain - Metcalfe's Law inverted

A consensus ledger in this digital form is not global, but application-specific. Any group of devices operating off of a common seed block effectively has an infinite blockchain or address space all to itself. Let's say for example you have a smart home, and within it you have a smart fridge, and a smart lawn mower, among other devices. You have configured your fridge to keep track of your food inventory, make shopping lists, and message you with shopping reminders, as well as having access to your bank information to reconcile your purchases. A routine activity of the refrigerator is to maintain a current shopping list, and when it detects purchases on your bank card that match items on the list, to remove the items and also update a spending report that is always available for when it comes time to pay bills. Now say you are mowing your lawn with your smart lawn mower and detects it is time for an oil change. It turns on its check engine indicator, and also messages the home via the refrigerator to add a couple quarts of oil to your shopping list. You finish mowing your lawn and turn off the lawn mower. The request for an oil purchase is received by the refrigerator and added to your shopping list. Eventually a few days later you go shopping and notice the oil on your list, so you pick it up. When you get home, the smart fridge detects the oil purchase and queues up a notification for the smart mower, whcih is currently off and unreachable. A few days later, you mow your lawn again,

and the lawn mower receives its message from earlier in the week about its request being fulfilled, and now it can let you know that it's time to change the oil and that it's available in the garage. This kind of round-trip economic activity is only possible if the devices can be offline for days or weeks at a time without requiring continuous back and forth communication. In addition, all these transactions take place entirely within the home, there is no need for your devices to receive notifications about your neighbor's purchases, or to store that information in their limited memory. Centralized ledgers require all participants to store all transactions worldwide onto their local memory, as well as do work for others for free by participating in concensus work for completely unrelated transactions. Also, because every device can compute its own blockchain given a seed block, devices can reason about time-ordered events without having to consult a central database.

Repeating the block generation example from above, adapted for this smart home example:

```
% mkrand -f8 -n10 -s "[<:C444C4CF3335E645C77F790EFB5E9D55:>]"
[<:1900F73EA129AF9A50EC5FE4C49BBE9D:>]          Eggs
[<:4419BCB82F1077EA0EC3DC69477E94DD:>]          Milk
[<:53F99033150119CBA9D67525AD1B726E:>]          Bread
[<:8C55EEDA7C6208DD126EA61D279141F6:>]          Lawn mower oil
[<:C5F1C39C36DBA146889444E89E963DE5:>]          Cheese
[<:640F97F111D89F79FD745E04A2EA8C24:>]          Dishwasher detergent
[<:551852E493924494BA17EF08F8447D49:>]          Soda
[<:FFE976344A6CC5D2AB6E1E479ABA9EF3:>]          Spark plugs
[<:2DE82C85AC76F91C394693723D62A005:>]          Bread
[<:C0F70220F54BA33DF4916CC29CFFD631:>]          Salt
```

We can see this particular blockchain was initialized with

```
[<:C444C4CF3335E645C77F790EFB5E9D55:>]
```

a seed block which every device in the smart home is given when it is registered onto the home network. All devices with that seed block can generate the same blockchain at any point in time without any further communication with any other device. This means the device can reason about time-order without needing to download or access the entire ledger. In the example of the lawn mower, in its limited internal memory it knows the seed block, and it needs to generate a block to request the oil, say the last time it was online it had seen the last transaction of 53F99 Bread being purchased, so it can assume the next block 8C55EE.. can be used for its oil purchase, so it uses that to "pay" for its purchase and the refrigerator adds it to its ledger which is more complete because it contains all sorts of household items not related to lawn care. In the event that block was used already by the time the oil purchase request is made, it is a simple matter for the fridge to remap its own ledger since it has more information than the lawn mower and it is also the authority for the ledger, so local knowledge is employed to easily fix local collisions. The mower then goes offine for a few days, and wakes up and sniffs the 55185.. soda purchase, updating its local ledger with that being the last known used block, then if it needs to emit a purchase request for spark plugs, it just generates the next block, in this case FFE97.. and so on. In this way, the local electronic economy of the smart devices can be seen as operating with a common coupon book and accidental duplicate use of a block due to missing some transactions is easily detected and fixed by the local authority, in this case the fridge, with local

knowledge and heuristics. Also the same mechanism can be used to reason about time ordering - any device on the network that sees the oil purchase of 8C55E.. and then later sees a purchase FFE97.. for spark plugs can easily compute which order came first, since it can just generate a few blocks forward from the oil purchase to find the spark plug block, so even if it receives the messages out of order, it can compute the strict order on its own, given the associated blocks. This does away with error-prone comparisons of floating-point wall-clock timestamps and all the complexity of time zones and clock drift.

While Metcalfe's Law suggests that the value of a network increases with the number of participants, non-digital blockchain networks may face challenges as their size and activity grow, because their resource usage and latency increase with the number of participants, leading to potential economic disincentives.

One key factor contributing to this inverse relationship is the energy consumption associated with certain blockchain networks, particularly those that use proof-of-work (PoW) consensus mechanisms like Bitcoin. As the number of participants (nodes) increases and more people try to post transactions to the blockchain, the competition to validate transactions and add blocks intensifies. This, in turn, drives up the computational power required for mining, leading to higher energy consumption and costs for the participants. As more nodes join and compete for block rewards, the energy expended by existing nodes can become an economic dis-incentive. The cost of electricity and specialized hardware needed for mining can outweigh the rewards for some participants, especially in areas with high electricity prices. This dynamic has led to concerns about the environmental impact and long-term sustainability of non-digital blockchain networks that heavily rely on PoW consensus. Proof-of-stake (PoS) consensus mechanisms waste less electricity than PoW, but suffer from the same inverse network dynamics, because they still need back-and-forth communication between participating devices and blockchain nodes - a large part of the wasted electricity is eliminated, but does nothing about the wasted network bandwidth or duplicated storage necessary for all nodes to hold a complete copy of the ledger. While technically these systems could be said to be de-centralized due to there not being one "master" node, the ledger itself and its consensus mechanism is most definitely a centralized data structure that must be interacted with in order to participate in that given blockchain.

0.10 Platform Compatibility

Currently MKRAND compiles on Linux, OS X, and embedded devices with a C compiler. It should compile on Windows using the appropriate tools for that platform but has not been tested. Digital implementations employ standard tools and technologies.