Cryptographic Puzzles and Complex Systems

Rade Vuckovac
Rade.Vuckovac@griffithuni.edu.au

A puzzle lies behind password authentication (PA) and blockchain proof of work (PoW). A cryptographic hash function is commonly used to implement them. The potential problem with secure hash functions is their complexity and rigidity. We explore the use of complex systems constructs such as a cellular automaton (CA) to provide puzzle functionality. The analysis shows that computational irreducibility and sensitivity to initial state phenomena are enough to create simple puzzle systems that can be used for PA and PoW. Moreover, we present puzzle schemata using CA and *n*-body problems.

Keywords: password authentication; proof of work; provable security

1. Introduction

Our fundamental proposal is to profit from how difficult it is to predict the behavior of complex systems. An analogous challenge is found in mathematics when attempting to define randomness and in computing when producing randomness. For example, a mathematical perspective on randomness can be summarized as follows.

> Although the concept of randomness is ubiquitous, it turns out to be difficult to generate a truly random sequence of events. The need for "pseudorandomness" in various parts of modern science, ranging from numerical simulation to cryptography, has challenged our limited understanding of this issue and our mathematical resources. [1]

Furthermore, number theorists provide insights into how number theory can contribute to the issue of randomness [2]:

- There is a dichotomy in number theory. Either we have a closed analytical formula for a given problem, or there is great difficulty and the problem shows random behavior.
- The outcome of this situation can go in two directions. Solving a problem brings a more profound understanding of a process behind the question. Otherwise, we are faced with the fact that some explicit arithmetical or Diophantine problems show randomness, and that can have considerable practical value.

Although the issue of randomness is important to our puzzle proposal, we will use other peculiarities of chaos theory and cellular automata (CAs) to construct password authentication (PA) and blockchain proof of work (PoW) puzzles.

1.1 Password Authentication and Proof of Work Puzzles

PA and PoW puzzles have the same foundation. Both use a problem where a solution is relatively hard to find (i.e., compute). When a solution is found, the verification process that confirms the correct solution is quite easy (computable in practice).

1.1.1 The Password Authentication Puzzle

In early days, passwords were kept as a plain text in some form of a table. The user typed a password, and it was compared to the table entry. If the stored and typed password were the same, some rights were granted to the user. This scenario is not very secure because access to the password table compromises the whole security scenario. The initial solution for this problem is to store a hash of the password instead. Now, when the user enters a password, the password is hashed. The resulting hash is compared with the hash stored in the table. If identical, the user gains the desired access. In this case, having access to the hash table does not reveal passwords because hash functions are hard to invert. However, the hash function itself is not a secret. The salt, a publicly known string, is concatenated to the password, preventing the building of a list of known hashes. Therefore, password authentication hashing computes the hash value h using two arguments to the hash function hash$(\cdot, \cdot)$ as

$$h = \text{hash}(\text{password}, \text{salt}), \tag{1}$$

where password and salt are two inputs to the hash function. An attacker is anyone who tries to find the corresponding password knowing h, the hash function and the salt. Note that the two-argument hash function typically is just the composition of catenation followed by a one-argument hash function.

1.1.2 The Proof of Work Puzzle

The PoW concept [3, 4] was introduced to generate a computationally hard question, easily verifiable. The first use of PoW was to prevent email spamming [4]. The sender must solve some puzzle before sending an email. The task consists of repeatedly hashing publicly available data (such as recipient email address, date) plus a string of random characters until an acceptable hash result is found. The acceptable hash has some predefined pattern (e.g., the value 0 for the first 20 bits). Those who want to provide evidence of work must vary the random part, called the nonce, until they produce a hash that meets the constraint of the pattern. For instance, in a hashcash system

[5], the chances of finding a pattern with 20 zeros is one in a million. In the email application, the recipient checks the solution (the proposed nonce) quickly (since the recipient only needs to evaluate the hash function once), and only accepts emails with confirmed nonce. The PoW puzzle is no obstacle for the genuine sender. Sending an email to multiple recipients becomes a burden because too many puzzles result in impractical computational cost. From the perspective of the sender, the problem has the form of the following equation:

$$p = \text{hash}(n, d) \tag{2}$$

where p is patterned hash value, hash is a hash function, n is the nonce (random string) and d is the public data.

1.1.3 General Puzzle

Both puzzles could be considered the same. From the perspective of the PA attacker and the PoW sender the problem has the form

$$t = \text{hash}(v, d) \tag{3}$$

where t is a target hash value (corresponding to h and p in PA and PoW, respectively), hash is a hash function, v is a variable part that attackers must compute (password or nonce, respectively) and d is public data (salt, an email address or a blockchain transaction ledger).

1.2 Cryptographic Hash Function

Today, schemata for PA/PoW commonly use a secure hash function. For example, every information technology user performs several password authentications daily. Blockchain applications are even larger users of hash functions. Bitcoin's (one of the popular blockchain technologies) energy footprint for hash computation (PoW) is in the neighborhood of a small country's energy consumption (between Greece and Switzerland [6]). The potential shortcomings concerning the use of hash functions are:

1. Secure hash functions are complex and not easy to develop. For example, the latest adoption of the secure hash function SHA-3 by NIST (US National Institute of Standards and Technology) was in 2015 [7]. The whole SHA-3 project started in 2007 and lasted for eight years. SHA-2 and SHA-3 are the only secure NIST recommendations. NIST emphasizes the recommendation without absolute security in the case of SHA-3. Bitcoin came into existence in 2009 and consequently had to use the older SHA-256 algorithm.
2. Typically, the security of the hash functions is argued using heuristic security. It is based on the notion that a particular algorithm is secure if it resists all current cryptanalyses. That is in sharp contrast to provable security, which relies on a computationally difficult problem (e.g., factoring).

3. Both systems, PA and PoW, have additional requirements such as time and memory hardening. Both are needed to raise the cost (in terms of computational time and hardware requirements) of the puzzle for the potential attacker. Hash functions are inflexible and cannot be used to regulate the cost, so additional machinery is required on top of hash algorithms.

2. Hashless Password Authentication / Proof of Work Puzzles

Our proposal is an alternative provable by security argumentation because both of the problems (the n-body and CA predictability) are related to unsolved mathematical problems. The relevant observations are the following.

1. *Butterfly effect*. They both exhibit the butterfly effect; that is, the system has an extremely high sensitivity to its initial state or initial conditions. This property might be interpreted as a lack of correlation between input and output. That is, similar inputs do not result in similar outputs. Even a minuscule change in the initial state (input) will produce a different outcome (output). Such a property could be considered equivalent to the avalanche effect required and evident in hash function designs [8]. Figure 1 illustrates the butterfly effect using cellular automata (CAs) [9].

2. *Computational irreducibility*. The principle of computational irreducibility says that the only way to determine the answer to a computationally irreducible question is to perform, or simulate, the computation [10].

   The n-body problem is a good example. When $n = 2$, the system has a known closed-form solution, meaning a set of equations is known that defines body motions and positions for any point in time. That is, positions can be computed efficiently. However, for $n > 2$, we do not have a closed-form solution. Instead, there is a set of equations that can only approximate body motions and positions for a given time. Today, the preferred method for solving an n-body system is step-by-step numerical simulation.

3. *Partial knowledge effect on reversibility*. The laws of physics are time reversible (including n-body systems), meaning that the known state of a physical system determines all past and future state configurations. Similar reversibility exists in the CA world. For example, the left-hand side of rule 30 evolution shows a pattern (Figure 4). Furthermore, if we know two columns of evolution (16 and 17), then the whole left-hand side (columns 1 to 15) is determined. By reducing knowledge to one column and showing only every second cell, the evolution is undetermined [11, p. 605]. A partially known state of a system in some circumstances can provide one-way functionality required by proposed schemata.

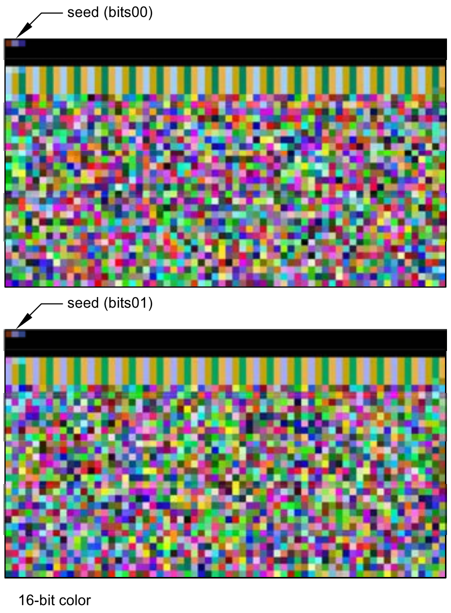

Figure 1. CAs and chaos. Two evolutions of the same CA are shown. The initial state consists of an array of 128 cells, each cell 32 bits wide. Every pixel represents two bytes. The seed, the first six pixels (six bytes, top-left of each image), is two strings "bits00" and "bits01", respectively. The rest of the state is set to zero, hence the black region after the seed. The first evolution iteration is almost the same for both (regions with stripes). Consequent iterations show chaos in action.

2.1 *n*-Body Puzzle

In Newton's time, the details of the motions of more than two orbiting bodies were considered as intractable. The problem became very important in the second half of the nineteenth century. Oscar II, king of Sweden, founded a prize for a 3-body solution. Henri Poincaré, a French mathematician, won that prize, even though the solution did not address the entire problem. However, that was an important moment in history because the origin of chaos theory could be traced to Poincaré's solution. Currently, *n*-body has a general solution [12, 13]. That solution is presented by a convergent power series, which is very slow and impractical to use. Numerical methods (approximations) or system simulations are used instead to solve practical problems [14, 15]. Figure 2 shows an example of *n*-body simulation.

n-body puzzle preliminaries. The puzzle proposal relies on a simulation environment. The idea is that only step-by-step simulation can evaluate an *n*-body system efficiently. In other words, if the future details of the system are needed, the quickest way of knowing them is to simulate it.

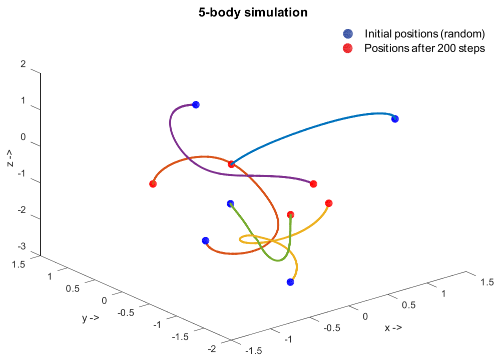

Figure 2. 10 bodies' paths traced out during three-dimensional simulation.

4-body puzzle. Imagine four bodies (planets). From initial state s_0, all four bodies (n_1, n_2, n_3, n_4) evolve to a state s_t by two-dimensional n-body simulation [16], after some time t.

This scenario could be translated to a password hashing scheme. The hash function is a 4-body simulation simulated for time duration t. The initial state of bodies (n_1, n_2) is encoded as a salt and (n_3, n_4) as a password. Then, simulation positions of bodies (n_3', n_4') represent the hash

$$(n_3', n_4') = \text{4bodysim}(n_1, n_2, n_3, n_4, t). \tag{4}$$

It is easy to see that if initial state (n_1, n_2, n_3, n_4) is known, future state (n_3', n_4') can be derived by a two-dimensional simulator. If details of n_3, n_4 (password) are unknown, public knowledge of n_1, n_2, n_3', n_4' details (salt and hash) is not sufficient to reverse two-dimensional simulation. Figure 3 illustrates this proposal.

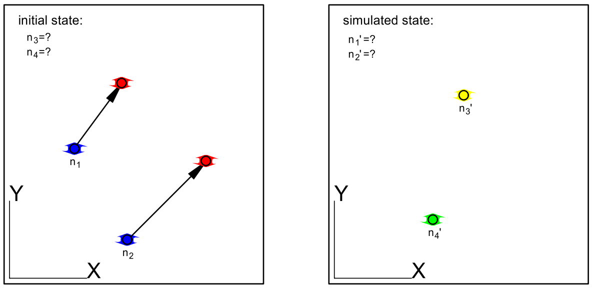

Figure 3. Initial state: n_1, n_2 details (a salt) are known and n_3, n_4 are unknown (a password). Simulated state: after time t; only n_3', n_4' details are known (a hash).

3. Cellular Automaton Puzzle

Stanislaw Ulam [17] and John von Neumann [18] discovered the concept of the cellular automaton (CA) in the 1940s. CAs are used as a modeling tool in various scientific fields, from computer and complexity science, mathematics and physics to biology. Stephen Wolfram was the first to propose the use of CAs (rule 30) for cryptography [19].

Figure 4 shows Wolfram's one-dimensional rule 30 CA. Rules on the top show how a cell (black or white) is transformed, depending on the neighboring cells. Row 1 is the initial state of the CA. Consecutive rows 1, 2, 3, … are evolved next generations. A next-generation cell is derived from neighbors of the previous one. For example, the cell (row 4, column 13) is derived by case 7. When a cell does not have a left or right neighbor in the row above it (e.g., row 16, column 1), it uses the cell from the opposite end of the preceding row (row 15, column 31). Therefore, the cell (row 16, column 1) is derived by case 7 (white, white, black).

The prize for solving rule 30 CA problems was announced in October 2019 (the center column is the 16th outlined column Figure 4):

- Problem 1: Does the center column always remain nonperiodic?
- Problem 2: Does each color of cell occur on average equally often in the center column?
- Problem 3: Does computing the n^{th} cell of the center column require at least O(n) computational effort [20]?

Problems 1 and 2 ask if some pattern in the center column exists because it behaves randomly (rule 30 is used as a random number generator by Wolfram's Mathematica software).

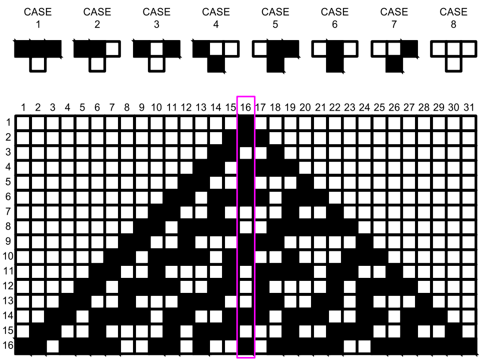

Figure 4. CA rule 30. Transformation rules and evolution history. Outlined column 16 can be considered as a random sequence.

Problem 3 asks if there is an approach shorter than actually running the CA (computational irreducibility). The run consists of each cell update contained in the diamond (Figure 5) and the runtime cost is $O(n^2)$ ($n^2 / 2$ updates). The question is, Can we determine the n^{th} cell without doing intermediate (diamond) cells updates [21]?

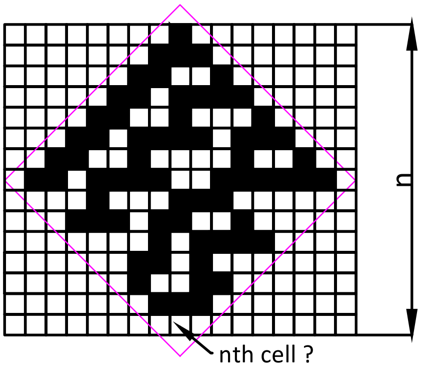

Figure 5. Rule 30. To determine the n^{th} cell of the center column, the cells in the diamond need to be determined first.

CA puzzle. In principle, every CA showing computational irreducibility can be used for puzzle proposal. On this occasion, we will use a cellular automaton generator (CAG) [9]. The rule for a cell transformation is shown in Figure 6. Details of the *cag* function are given in the Appendix, (function *stir*).

Figure 7 shows a CAG evolution history of eight cycles. The initial state is an array with 128 cells, 32 bits wide. It can contain a password (input x) and a salt (data d). The black region is set to zero (in grayscale encoding, zero is black), and that is the initial state. The third generation already shows some randomness. The part of the eighth cycle (output y) serves as a hash of the password

$$y = cag(x, d). \tag{5}$$

Note that *cag* is not a one-way function. Meaning, if the current state is known, the future and past states are computable. However, the same problem remains, because only the partial final state is known. The complete final state must be known before reversal. Therefore a provisional part has to be guessed, and that provisional part with the output (y) has to meet the initial state constrained with public data. Finding a correct provisional part is as costly as a search for correct input (password)

$$\text{initial state} = \text{inverse } cag(y, \text{provisional state}). \tag{6}$$

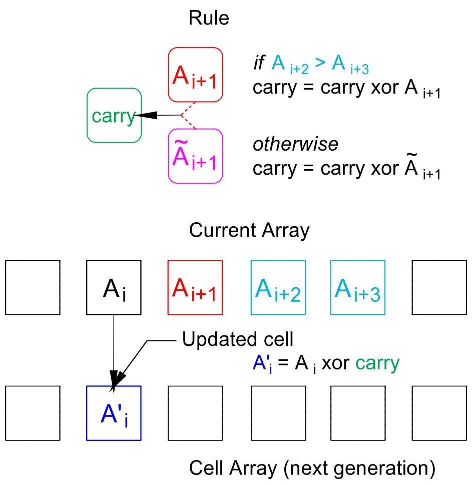

Figure 6. CAG transformation rule. “Current Array” and “Cell Array” are the previous and next rows in one-dimensional CA evolution. Cells are multi-bit integers (e.g., 32 bit). The next cell (blue) is the result of extended or (xor) between the previous cell (black) and carry (green). The carry is the result of xor of the previous carry value and the right-hand neighbor (red). The right-hand neighbor comes in two values: original state (red) or flipped state (magenta). Which neighbor state (original or flipped) is used depends on the cyan neighbor’s relation.

Out[5]=

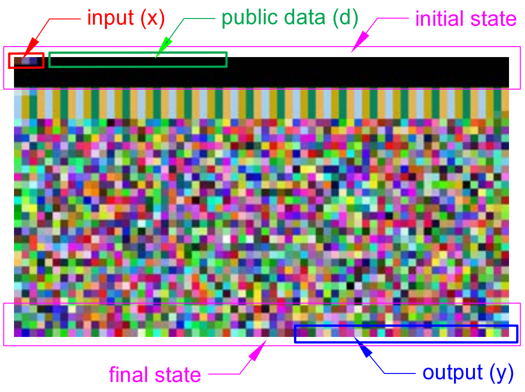

Figure 7. CA puzzle. The CA initial state is partitioned as input and some public data (red and green). Output (blue) is a partial state of the last CA evolution cycle. When green, blue and CA transformation rules are known, it is hard to find red.

4. Proposed Puzzles and Provable Security

Cryptanalysis dealing with CA rule 30 appeared in [22]. Hash and one-way functions based on CAs are proposed and discussed in [23, 24]. Those works are in the heuristic security category, meaning that security is based on resistance against newly discovered and known attacks. That argumentation is open-ended because new attacks and defenses techniques can be developed in the future.

From the heuristics point of view, puzzle proposals should accommodate the following hash requirements:

- Arbitrary input and fixed output size.
- Pre-image resistance: when given a hash, it is difficult to find the message that will result in the given hash.
- Second pre-image resistance: when given a particular message, it is difficult to find another message with the same hash.
- Collision resistance: it is challenging to find a pair of messages with the same hash.

Depending on the hash design, security is argued by addressing known design weaknesses. For example, the hash function based on block ciphers has an argumentation advantage because block ciphers are extensively studied, and their weaknesses are well known.

Hash functions can be based on mathematical problems as well. This approach tends to be impractical. Some hash functions based on problems usually found in public-key cryptography are:

- Discrete algorithm problem, muHASH [25].
- Problem of finding modular square roots, VSH [26].

Because proposed puzzles are connected with well-known hard problems (n-body and rule 30), a provable security method appears to be a reasonable approach.

Discussion. First we assume that the computational irreducibility principle holds. That means a desired state of some system is only reachable by running/simulating that system. From a mathematical point of view, we can consider it as a function with inputs and outputs, where the state transformation is the function and various states in transition are inputs and outputs (Table 1). For example:

$$s' = \text{rule30}(s,\ i) \tag{7}$$

where rule30 is CA rule 30, s is initial state (e.g., the first row of Figure 4) and s' is the state after i CA iterations (e.g., the last row after 15 iterations, Figure 4).

Now, we can speculate on the properties of the state transition function and its input/output (I/O) behaviors. First, we can sort the

initial state	evolved state
input	output
s_1	s'_1
s_2	s'_2
…	…

Table 1. Tabular representation of a CA state transition.

input column and observe what happens to the output column after several experiments. We find that the output column does not behave in an orderly fashion, and there is no visible pattern in the output data. This phenomenon can be seen in chaos theory as a butterfly effect, where very similar inputs produce drastically different output. A CA example of it is shown in Figure 1, where 4096-bit initial states differ by just one bit. That behavior might be expected because we presume that the only access to I/O mapping is running the CA rule or simulating an n-body system. On those premises, we can create a puzzle by restricting knowledge to a partial value of input and output. Since running a system is the only way to find output, an exhaustive search is needed to match a partially known input with a partially known output. In this case, the exhaustive search cost is $O(n)$—a linear traversal of the elements in a column.

However, this hypothesis might not hold. In that case, the searching time cost for I/O matches will be $<O(n)$. In theory, that is possible. A complete I/O table sorted by output will enable a search of $O(\log n)$ cost. In practice, that table is impossible to create because of memory constraints. For example a 256-bit hash (output) needs a table of $\approx 2^{256}$ entries.

Some other approaches might exist that can determine some order behind the table mappings. One promising candidate is machine learning and artificial neural networks algorithms. They are the latest techniques for solving complex problems. For example, we have a headline from *MIT Technology Review* such as "A Neural Net Solves the Three-Body Problem 100 Million Times Faster" [27]. Comments on that article are [28]:

> The revolution of machine learning has been greatly exaggerated.… The trouble is, the authors have given no compelling reason to think that they could actually do this.

Even if we do not know how the proposal can be broken, we can estimate some of the attack properties. Its I/O match searching cost has to be less than exhaustive search ($O(n)$ time cost estimate).

For example, the attack on CA rule 30 [22] shows how some patterns in evolution history can be exploited. For example, the pattern of rule 30 is clearly visible (Figure 4, left-hand side).

It is also shown that for rule 30, the initial state of $N = 200$ cells and success probability of $\delta = 0.5$, the number of trials needed to find the mapping in question is only $\mu = 23\,000$. However, the number of cells N and number of trials μ remain in an exponential relation. For a number of trials $\mu = 2^{50}$, the estimated number of cells is between $N = 750$ and $N = 900$, indicating $O(n)$ searching cost. A similar situation occurs with the factoring problem. While the latest factoring record [29] is to factor a 829-bit number, the factoring problem remains unsolved, and cryptography based on the factoring problem remains sound as well (excluding quantum computing).

If proposed puzzles are defeated, then Table 1 will emerge with some mathematical structure and should be comparable with other data structures where the searching cost is less than $O(n)$ (such as various types of tree data structures). The eventual mathematical structure will be a significant finding for the n-body problem, potentially inferring an efficient closed solution. For CA rule 30 problems, randomness questions will be faced with apparent mathematical structure. The need to perform all intermediate steps between initial state s and state s' to find the n^{th} cell will disappear.

Appendix

A. Implementation Details

With advances in computer hardware, the password exhaustive search method becomes a very viable option. A couple of schemes were developed in the 1990s to address this issue. MD5 crypt [30] and bcrypt [31] are examples still used in some Unix flavors today. The idea behind schemes is to make a hash function computation slow (time hardening). For example, let us assume that one hashing of a password eight characters long can be done in 0.001 seconds. Brute force with the newest hardware finds the hash/password pair in 10 minutes. If we iterate the hash function 500 times by feeding the resulting hash as a new hashing input, the computing cost is 0.5 seconds. Half a second waiting for a user is acceptable, but brute force cost is 83.33 hours now. Please see Table A.1.

hash iterations	user wait	attacker search cost
1	0.001 second	10 minutes
500	0.500 second	83.33 hours

Table A.1. PA time hardening.

A similar approach is adopted for hashing memory requirement. By increasing the amount of fast storage required for hashing, the brute force hardware cost should go up as well. One example of a memory hardening scheme is scrypt [32]. However, this strategy does not prevent custom-made hardware, especially in PoW schemes:

1. Hardware specialization resistance is futile.
2. The "success" of altcoins is not reliant on resistance.
3. PoW algorithms need to be easy to manufacture [33].

The PA puzzle implementation uses the CAG cellular automaton in Listing 1. The description details of the CA used are given in [9]. Because it is ideal for mapping onto field-programmable gate arrays (FPGAs), it is efficiently implemented in hardware [34].

The original listing appeared in the PHC competition. A modified version with the exclusion of memory hardening is presented. The motivation for that comes from some conclusion points made about hardware resistance (memory hardening), especially on PoW's requirement to be easily implemented in hardware (for which the proposed hash function showed great potential).

The time hardening is achieved by specifying the length of evolution. Line 30; rounds = 4; specifies that four evolution cycles are applied to the initial state. For the desired time increase of evolution, this variable should be adjusted accordingly.

```
//http://lists.openwall.net/phc - discussions/2014/09/18/3
#include < stdio.h >
#include < stdint.h >
#include < string.h >
#include < stdlib.h >

void stir (uint64_t * state, uint64_t statelen)
{
        uint64_t mixer = 6148914691236517205; // 010101 ...
        uint64_t carry = 1234567890123456789;
        uint64_t j;

        for (j = 0; j < statelen; j++)
        {
                if (state[(j + 2) % statelen] > state[(j + 3) % statelen])
                        carry ^= state[(j + 1) % statelen];
                else
                        carry ^= ~state[(j + 1) % statelen];
                state[j] ^= carry;
                carry += mixer;
        }
}
```

```
int PHS (void *out, size_t outlen,
  const void *in, size_t inlen,
  const void *salt, size_t saltlen)
{
        uint64_t state[256] = {0};
        uint64_t statelen = 256;
        uint64_t rounds = 4;
        size_t i;
        memmove (& state[0], in, inlen);
        memmove (& state[(inlen / 8) + 1], salt, saltlen);
        state[statelen - 3] = outlen;
        state[statelen - 2] = inlen;
        state[statelen - 1] = saltlen;

        stir (state, rounds * statelen);

        for (i = 0; i < outlen; i++)
                memcpy (& out + i, & state[i * 8], sizeof (char));

        return 0;
}
```

Listing 1. Password hash function (shhvrch01.h).

References

[1] J. Bourgain. "Searching for Randomness." Institute for Advanced Study. (Jul 22, 2021) www.ias.edu/ideas/searching-randomness.

[2] P. Sarnak, "Randomness in Number Theory," *Asia Pacific Mathematics Newsletter*, **2**(3), 2012 pp. 15–19. www.asiapacific-mathnews.com/02/0203/0015_0019.pdf.

[3] M. Jakobsson and A. Juels, "Proofs of Work and Bread Pudding Protocols (Extended Abstract)," in *Secure Information Networks: Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99)*, Leuven, Belgium (B. Preneel, ed.), Boston: Kluwer Academic Publishers, 1999 pp. 258–272. doi:10.1007/978-0-387-35568-9_18.

[4] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," in *Advances in Cryptology—CRYPTO '92*, Santa Barbara, CA (E. F. Brickell, ed.), Berlin, Heidelberg: Springer, 1992 pp. 139–147. doi:10.1007/3-540-48071-4_10.

[5] A. Back. "Hashcash—A Denial of Service Counter-Measure." (Aug 2, 2021) www.hashcash.org/papers/hashcash.pdf.

[6] digiconomist. "Energy Consumption Index." (Jul 22, 2021) digiconomist.net/bitcoin-energy-consumption.

[7] "Hash Functions." NIST: Information Technology Laboratory. (Jul 22, 2021) csrc.nist.gov/projects/hash-functions/sha-3-project.

[8] H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, **228**(5), 1973 pp. 15–23. www.jstor.org/stable/24923044.

[9] R. Vuckovac, "Secure and Computationally Efficient Cryptographic Primitive Based on Cellular Automaton," *Complex Systems*, **28**(4), 2019 pp. 457–474. doi:10.25088/ComplexSystems.28.4.457.

[10] N. Israeli and N. Goldenfeld, "Computational Irreducibility and the Predictability of Complex Physical Systems," *Physical Review Letters*, **92**(7), 2004 074105. doi:10.1103/PhysRevLett.92.074105.

[11] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[12] K. F. Sundman et al., "Mémoire sur le problème des trois corps," *Acta Mathematica*, **36**, 1913 pp. 105–179. doi:10.1007/BF02422379.

[13] W. Qiu-Dong, "The Global Solution of the *N*-body Problem," *Celestial Mechanics and Dynamical Astronomy*, **50**(1), 1990 pp. 73–88. doi:10.1007/BF00048987.

[14] K. T. Alligood, T. D. Sauer and J. A. Yorke, *Chaos: An Introduction to Dynamical Systems*, New York: Springer-Verlag, 1996.

[15] M. Trenti and P. Hut, "N-Body Simulations (Gravitational)," *Scholarpedia*, **3**(5), 2008 3930. doi:10.4249/scholarpedia.3930.

[16] E. Zeleny, "N-Body Problem in 2D" from the Wolfram Demonstrations Project–A Wolfram Web Resource. demonstrations.wolfram.com/NBodyProblemIn2D.

[17] S. Ulam, "Random Processes and Transformations," in *Proceedings of the International Congress of Mathematicians,* Cambridge, MA, 1950, vol. 2., Providence, RI: American Mathematical Society, 1952 pp. 264–275.

[18] J. von Neumann, *Theory of Self-Reproducing Automata* (A. W. Burks, ed.), Urbana, IL: University of Illinois Press, 1966.

[19] S. Wolfram, "Cryptography with Cellular Automata," in *Advances in Cryptology—CRYPTO '85 Proceedings,* Santa Barbara, CA (H. C. Williams, ed.), Berlin, Heidelberg: Springer, 1985 pp. 429–432. doi:10.1007/3-540-39799-X_32.

[20] "The Wolfram Rule 30 Prizes." The Wolfram Foundation. www.rule30prize.org.

[21] S. Wolfram. "Announcing the Rule 30 Prizes." writings.stephenwolfram.com/2019/10/announcing-the-rule-30-prizes.

[22] W. Meier and O. Staffelbach, "Analysis of Pseudo Random Sequences Generated by Cellular Automata," in *Advances in Cryptology—EUROCRYPT '91*, Brighton, UK (D. W. Davies, ed.), Berlin, Heidelberg: Springer, 1991 pp. 186–199. doi:10.1007/3-540-46416-6_17.

[23] M. Mihaljevic, Y. Zheng and H. Imai, "A Cellular Automaton Based Fast One-Way Hash Function Suitable for Hardware Implementation," in *Public Key Cryptography—PKC 1998*, Pacifico Yokohama, Japan (H. Imai and Y. Zheng, eds.), Berlin, Heidelberg: Springer, 1998 pp. 217–233. doi:10.1007/BFb0054027.

[24] M. Mihaljevic, Y. Zheng and H. Imai, "A Family of Fast Dedicated One-Way Hash Functions Based on Linear Cellular Automata over GF(q)," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **82**(1), 1999 pp. 40–47.

[25] M. Bellare and D. Micciancio, "A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost," in *Advances in Cryptology—EUROCRYPT '97*, Konstanz, Germany (W. Fumy, ed.), Berlin, Heidelberg: Springer, 1997 pp. 163–192. doi:10.1007/3-540-69053-0_13.

[26] S. Contini, A. K. Lenstra and R. Steinfeld, "VSH, an Efficient and Provable Collision-Resistant Hash Function," in *Advances in Cryptology—EUROCRYPT 2006*, St. Petersburg, Russia (S. Vaudenay, ed.), Springer, 2006 pp. 165–182. doi:10.1007/11761679_11.

[27] Emerging Technology from the arXiv. "A Neural Net Solves the Three-Body Problem 100 Million Times Faster." *MIT Technology Review*. (Jul 22, 2021) www.technologyreview.com/2019/10/26/132171/a-neural-net-solves-the-three-body-problem-100-million-times-faster.

[28] G. Marcus and E. Davis, "Are Neural Networks About to Reinvent Physics?," *Nautilus*. (Jul 22, 2021) nautil.us/issue/78/atmospheres/are-neural-networks-about-to-reinvent-physics.

[29] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P. Zimmermann, "Comparing the Difficulty of Factorization and Discrete Logarithm: A 240-digit Experiment." arxiv.org/abs/2006.06197.

[30] N. Provos and D. Mazières, "Md5 crypt," in *USENIX*, 1999. (Aug 3, 2021) www.usenix.org/legacy/event/usenix99/provos/provos_html/node10.html.

[31] N. Provos and D. Mazières, "A Future-Adaptable Password Scheme," in *1999 USENIX Annual Technical Conference*, Monterey, CA, 1999 pp. 81–91. www.usenix.org/legacy/events/usenix99/provos.html.

[32] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function," *IETF Draft*. (Jul 22, 2021) datatracker.ietf.org/doc/html/draft-josefsson-scrypt-kdf-00.

[33] StopAndDecrypt. "Asic Resistance Is Nothing but a Blockchain Buzzword." Hacker Noon. (Jul 22, 2021) hackernoon.com/asic-resistance-is-nothing-but-a-blockchain-buzzword-b91d3d770366.

[34] A. Palchaudhuri and A. S. Dhar, "Primitive Instantiation for Speed-Area Efficient Architecture Design of Cellular Automata Based Mageto Logic on FPGA with Built-In Testability," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Fayetteville, AR, Piscataway, NJ: IEEE, 2020 pp. 207–207. doi:10.1109/FCCM48280.2020.00038.