Nonuniform Cellular Automata for Cryptography

Marco Tomassini
Mathieu Perrenoud
Computer Science Institute,
University of Lausanne,
1015 Lausanne, Switzerland

Cryptography is a basic requirement in today's distributed information storage and transmission systems. A single key cryptographic system based on cellular automata is described. The approach employs high-quality pseudorandom bit sequences produced by one- and two-dimensional nonuniform cellular automata. The robustness of the scheme against cryptanalytic attacks is discussed and it is shown that direct cryptanalysis requires an exponential amount of computational resources. A further advantage of the proposed scheme is that it is eminently suitable for hardware implementation.

1. Introduction

Cryptographic techniques are very important in these times dominated by the growth of digital information storage and transmission. In fact, increasingly available communication networks and databases make the need for privacy and authentication a basic requirement in many areas, especially in electronic commerce transactions and for classified material. There exist many different cryptographic techniques, an excellent review is given in [9]. Here we will describe the potential uses of some types of cellular automata (CA) in this domain. CA have previously been suggested as encrypting devices by Wolfram [14] and by Nandi *et al.* [8]. Independently, Guam [3] and Gutowitz [4] also used CA for cryptographic purposes. We will not discuss Guam's and Gutowitz's works here since the principles on which they are based are different from ours. In particular, Guam's work concerns public-key cryptography, while we will only discuss *symmetric* systems, where the encryption key and the decryption key are the same or can be calculated from each other. As in references [8, 14], our encryption scheme is based on the generation of pseudorandom bit sequences by CA. In section 2, we summarize work done on CA for random number generation by our group and by others. Section 3 presents the implementation of the proposed cryptographic system and compares it with previous approaches. Finally, we discuss the vulnerability of the scheme to possible cryptanalytic attacks, showing that the scheme is stronger that those previously proposed and of practical use for many common types of cryptographic applications.

2. Pseudorandom number generation by cellular automata

CA are dynamical systems in which space and time are discrete. A CA consists of an array of cells, each of which can be in one of a finite number of possible states, updated synchronously in discrete time steps, according to a local, identical interaction rule. Here we will only consider boolean automata for which the cellular state $s \in \{0, 1\}$. The state of a cell at the next time step is determined by the current states of a surrounding neighborhood of cells. The cellular array (grid) is d-dimensional, where $d = 1, 2, 3$ is used in practice. In this paper we shall concentrate on $d = 1$ and $d = 2$, that is, one- and two-dimensional grids. The identical rule contained in each cell is essentially a finite state machine, usually specified in the form of a rule table (also known as the transition function), with an entry for every possible neighborhood configuration of states. The cellular neighborhood of a cell consists of itself and the surrounding (adjacent) cells. For one-dimensional CA, a cell is connected to r local neighbors (cells) on either side where r is referred to as the radius (thus, each cell has $2r + 1$ neighbors). For two-dimensional CA, two types of cellular neighborhoods are usually considered: *5 cells*, consisting of the cell along with its four immediate nondiagonal neighbors (also known as the von Neumann neighborhood), and *9 cells*, consisting of the cell along with its eight surrounding neighbors (also known as the Moore neighborhood). When considering a finite-sized grid, spatially periodic boundary conditions are frequently applied, resulting in a circular grid for the one-dimensional case, and a toroidal grid for the two-dimensional case.

Nonuniform (also known as inhomogenous) CA function in the same way as uniform ones, the only difference being that the cellular rules need not be identical for all cells. Note that nonuniform CA share the basic "attractive" properties of uniform ones (e.g., simplicity, parallelism, locality).

A common method of examining the behavior of one-dimensional CA is to display a two-dimensional space-time diagram, where the horizontal axis depicts the configuration at a certain time t and the vertical axis depicts successive time steps (see Figure 1). The term "configuration" refers to an assignment of ones and zeros at a given time step (i.e., a horizontal line in the diagram).

S. Wolfram in [15] first proposed one-dimensional CA as pseudorandom number generators (PRNG). In particular, he extensively studied the bit sequences generated by rule 30 in his numbering scheme for one-dimensional, $r = 1$ rules, where the rule number represents in decimal format the binary number encoding the rule table. For example, $f(111) = 1$, $f(110) = 0$, $f(101) = 1$, $f(100) = 1$, $f(011) = 1$, $f(010) = 0$, $f(001) = 0$, $f(000) = 0$, is denoted rule 184. In boolean form rule 30 can be written as:

$$s_i(t+1) = s_{i-1}(t) \text{ XOR } (s_i(t) \text{ OR } s_{i+1}(t)),$$

where $s_i(t)$ is the state of cell i at time t. The formula gives the state of cell i at time step $t+1$ as a boolean function of the states of the neighboring cells at time t. Pseudorandom bit sequences are obtained by sampling the values that a particular cell (usually the central one) attains as a function of time. Often, in order to further decorrelate bit sequences, *time spacing* and *site spacing* are used. Time spacing means that not all of the bits generated are considered as part of the random sequence. For instance, one might keep only one bit out of two, referred to as a time space value of 1, which means that sequences will be generated at one-half the maximal rate. In site spacing, one considers only certain sites in a row, where an integer number indicates how many sites are to be ignored between two successive cells. In practice, a site spacing of one or two is common, which means that one-half or two-thirds of the output bits are lost. For applications that need very good quality random numbers, this sacrifice is acceptable, if one takes into account that, in fact, many parallel streams of random bits are being generated simultaneously by a CA.

A nonuniform CA randomizer was presented in [5, 6], consisting of two rules, 90 and 150, arranged in a specific order in the grid. In boolean form rule 90 can be written as:

$$s_i(t+1) = s_{i-1}(t) \text{ XOR } s_{i+1}(t),$$

and rule 150 can be written as:

$$s_i(t+1) = s_{i-1}(t) \text{ XOR } s_i(t) \text{ XOR } s_{i+1}(t).$$

The performance of this hybrid (nonuniform) CA in terms of random number generation was found to be superior to that of rule 30 and to the usual linear feedback shift register approach.

Sipper and Tomassini [11] showed that good nonuniform CA randomizers can be evolved by a genetic algorithm, rather than being designed. This work was pursued by our group, finding better nonuniform CA for RNG by using a fine-grained parallel genetic algorithm called *cellular programming* (for details, see [13]). Figure 1 depicts one of the best one-dimensional nonuniform CA that was obtained. Actually, this CA was hand-constructed by randomly mixing the four rules that appeared most frequently within successfully evolved CA. The rule numbers are 90, 105, 150, and 165 and one of the results of [13] is that any

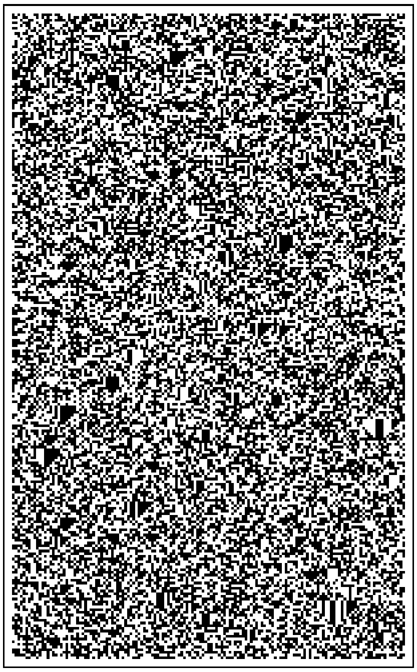

Figure 1. A good quality nonuniform one-dimensional random number generator consisting of a random mixture of rules 90, 105, 150, and 165.

random mixture of these rules produces a good RNG. This observation will be made use of later in the construction of cryptographic keys.

More recently, we have obtained by artificial evolution and design two-dimensional, nonuniform CA for random number generation with even better statistical quality [12], as demonstrated by the application of a complete battery of stringent tests for pseudorandomness.

Pseudorandom number sequences are needed in many important applications, such as Monte Carlo techniques, Brownian dynamics, and stochastic optimization methods. Moreover, CA offer a number of advantages over other methods, especially where hardware implementation is concerned. Among the beneficial features of CA for VLSI implementation one can cite simplicity, regularity, and locality of interconnections, which make them suitable for on-board applications such as built-in self-test circuits, associative memories, and special purpose massively parallel fine-grained computers. See [2] for a good description of the theory and VLSI applications of CA. In section 3 we will focus on a cryptographic application of the pseudorandom bit sequences generated by the CA described.

3. A cellular automata-based key stream generator

Let P be a plaintext message and E an enciphering algorithm. The fundamental transformation to obtain the ciphertext C is thus:

$$C = E_k(P),$$

where k is the *key* of the transformation which distinguishes a particular encryption in a family of transformations using the same enciphering algorithm. To recover the original message, a deciphering function D_k, using the same key, is defined as the inverse of E:

$$P = D_k(C) = D_k(E_k(P)).$$

Enciphering algorithms that operate on the plaintext a single bit at a time are called *stream algorithms* or *stream ciphers*. A stream cipher breaks the message P into a stream of successive bits or bytes $p_1, p_2, \dots, p_q$ and enciphers each p_i with a stream of bits (or bytes) $k_1, k_2, \dots, k_q$ generated by a *key stream* generator such that:

$$E_k(P) = E_{k_1}(p_1)E_{k_2}(p_2)\dots\,.$$

A common enciphering operation, and the one used here, is the exclusive-or operation XOR:

$$c_i = k_i \text{ XOR } p_i,$$

where c_i is the ith bit of the ciphertext. Applying the same operation on the ciphertext allows the recovery of the original text:

$$p_i = c_i \text{ XOR } k_i = (k_i \text{ XOR } p_i) \text{ XOR } k_i.$$

If the key stream is truly unpredictable, then we have the so-called "one-time pad" system which is perfectly safe (assuming that the keys are not stolen or eavesdropped). The one-time pad was invented by J. Mauborgne, an American Army officer, and is based on a variation of the Vernan cipher in which the key never repeats itself, (e.g., [9]). However, the one-time pad is impractical because the sender and the receiver must be in possession, and protect, the random key. Moreover, the total amount of data that can be encrypted is limited by the length of the key available.

Thus, the security of a stream cipher scheme rests on the predictability of the bits in the key stream. Good statistical pseudorandomness of the key stream is not sufficient in cryptographic applications: a perfectly good RNG may be completely unsuitable if the next random bit can be predicted from the previous sequence. From that point of view, CA are more suitable than classical RNGs such as linear congruential generators, which are very easy to crack, given the algorithm and a small portion of the sequence (e.g., [9]).

Given a CA of size N, a *configuration* of the grid at time t is defined as:

$$C(t) = (s_0(t), s_1(t), \ldots, s_{N-1}(t)),$$

where $s_i(t) \in \{0,1\}$ is the state of cell i at time t. For a CA-based key stream generator, the initial configuration of states $C(0)$ is the key or a part of the key. Thus, cryptanalysis of the system amounts to the reconstruction of $C(0)$. In [14], the bits $k_1, k_2, \ldots$ to be used for enciphering the message stream $p_1, p_2, \ldots$ are extracted from the sequence $(s_i(t), s_i(t+1), s_i(t+2), \ldots)$ attained through time at a site i (say, the central cell) of a rule 30 CA. It would appear that reconstructing the key from a fragment of the sequence is a difficult problem. However, in [7] it is shown that there is a known plaintext attack against this generator which is successful in reasonable time and with limited computing power for key sizes N up to 500. The attack is based on reconstructing a right (or left) sequence adjacent to the sequence at site i. This can be done by guessing adjacent site states with high probability of success, due to the particular nature of rule 30. Once the adjacent sequence is known, it is an easy matter to successively compute the missing site values backwards to the complete $C(0)$ key. The generator is thus insecure.

In another study [8], using a nonuniform CA generator based upon rules 90 and 150 is proposed for producing a key stream. It has been shown that such a generator produces good-quality pseudorandom streams (see section 2 and [5]), better than either rule 30 or linear feedback shift registers. In [8] both block and stream cipher algorithms are presented, here we will only briefly discuss the stream cipher scheme. They proposed two different systems: a programmable CA with ROM (PCA) and a two-stage programmable CA. Both schemes are easy to implement in hardware. In the PCA, an N-cell CA register is loaded with one of a number of $(90, 150)$ rule configurations stored in ROM and run for a clock cycle. A group of four consecutive output bits is chosen out of the N output bits and xored with a portion of plaintext of the same length to give the corresponding ciphertext. The CA is then loaded with another rule configuration from the ROM and the cycle continues until all the text has been encrypted. The two-stage system works according to the same principles but instead of configuring the CA from a fixed set of configurations stored in ROM, a second CA register, which is itself constituted by a fixed $(90, 150)$ rule vector, randomly produces the desired $(90, 150)$ current rule configuration for the first CA register at each clock cycle.

In view of the static and periodically repeating rule configurations stored in ROM, the PCA appears to be vulnerable to attacks. In fact, successive rule configurations are always used in the same sequential order. Moreover, the suggested values for the CA register length and for the number of rule configurations stored in ROM are small, thus facilitating a cryptoanalytic attack. The two-stage system should be more secure since, ideally, there is no correlation between successive rule configurations. Both systems use groups of four consecutive bits, a fact that makes rule induction by divide-and-conquer and completion easier.

In our scheme, we use nonuniform CA in which each cell rule is randomly sampled with uniform probability from rules 90, 105, 150, and 165. We have shown in [12] (see also section 2) that these automata have excellent PRNG properties. The key K is formed from the current vector of CA rules R_i and the initial random state of the CA $C(0)$: $K = \langle R_i,\ C(0)\rangle$. For an N-cell CA each of the N rules can be chosen independently among the set of four and each bit in the initial configuration is also independent; thus, the apparent length of the key is: $4^N \times 2^N = 2^{3N}$. To give an order of magnitude, N can be well above 100 if the key stream generator is implemented in software and in the order of 100 for VLSI implementation, although this is only a cost/benefit tradeoff and suitable values of N can be used as a function of the degree of security one wants to reach. Obviously the hardware implementation will be several orders of magnitude faster. The rule vector and the initial state can be established once per session (session key) or they can be changed at any time. Frequent change of K is advisable not only for security reasons but also because there is no guarantee that the nonuniform CA generated in this way will be maximum-length CA. However, our previous work [12] has shown that the average cycle lengths will be large enough, in the order of 2^{n-4}. As in [14], bits are sampled from the central column of a cyclically connected CA. But in our case divide-and-conquer attacks of the kind proposed in [7] are much more difficult and the following argument shows that guessing the key is indeed a hard task.

Suppose that we have a plaintext and the corresponding ciphertext of a given arbitrary but finite length. The goal of the cryptanalysis is to find the key $K = \langle R_i,\ C(0)\rangle$ starting from the bit stream generated with K during a number n of time steps. The bit stream used for encoding can obviously be deduced unambiguously from the plaintext/ciphertext pair. The search space is huge as there exist 2^{3N} possible key configurations. However, among those only a small fraction generate the given bit stream as the following counting argument will show. We will start by trying to reconstruct the automaton backwards in a triangle shape starting from a given bit in the central sequence until we have reconstructed a full line. The triangle shape is due to the fact that each cell state depends on the state of its two neighbors at the previous time step. For each time step we must make hypotheses about the rules and the states of the cells and we will count the number of guesses made in order to complete the line. Consider Figure 2(a) which represents the states of the relevant cells and the rules at time steps t and $t-1$ respectively.

rules	A
t-1	m x n
t	y

(a)

rules	B A C
t-2	o p z q r
t-1	m x n
t	y

(b)

Figure 2. Schema of the first two stages in the backward completion of the automaton by guessing rule and cell values.

Cell states x and y are known since they belong to the central sequence. We must guess the central cell's rule, let us call it A, and the value (0 or 1) of one of the neighbors of the cell whose value is x. Assume that we guess m. This is enough to infer the $t-1$ slice of the triangle since we only work with additive rules (XOR and XNOR logic only). At this stage, there are eight different configurations of the triplet (A, m, n) producing the sequence (x, y). If we now go one time step further backwards to time $t-2$, we have the situation depicted in part (b) of the figure. The central cell value z is known and we need to infer the state values o, p, q, r. With A, m, x, y, n now fixed we must guess rules B and C and one of o, p, q, r as the other values can then be inferred. There are 32 possible configurations of (B, C, o, p, q, r) that produce the bits (z, m, x, y, n). Now it is easy to see that for each further time step backwards there are 32 possible configurations of rules/values to consider in order to obtain the situation at the previous time step and the process can be iterated $(n-1)/2$ times until we get a full line. Taking the product gives us the number of combinations of rules/values that produce the given sequence of $n/2$ bits. This number is $u = 2^{(5n-9)/2}$. But only one of those combinations will fit a complete sequence. In fact, if we now try to extend the bit sequence further down by one bit, only one-half of the solutions will produce the right sequence. The next bit will reduce that to a quarter and so on. Thus, to find the right combination of rules and initial state values, one has to test all u cases over $\log(u)$ subsequent bits belonging to the central sequence.

The system can be made more secure by time spacing the bit stream, that is, by discarding one bit out of two or three. This will have the effect of increasing the number of rules/values that have to be guessed to reconstruct the time/space diagram backwards.

Although the previous argument shows that direct cryptanalysis of the system is a task requiring an exponentially increasing amount of computational resources, one cannot rule out the possibility that, by exploiting some kind of information-theoretic or statistical regularities, the system can be transformed into an equivalent one that is easier to analyze. With the information given here, our generator can be easily programmed and any length of plaintext/ciphertext pair can be generated. As such, the proposed encryption technique is open to scrutiny.

Here we have mainly described the use of one-dimensional nonuniform CA. In [12] we showed that two-dimensional nonuniform CA are even better PRNGs. They can be used in a manner analogous to the one-dimensional case for generating key streams. We found that seven additive rules tend to emerge consistently *via* artificial evolution and give rise to excellent PRNGs. Randomly mixing these rules again produces very good generators. Since in this case the number of rules is higher with respect to the one-dimensional case and the neighborhood comprises five cells, reconstructing the key is even more difficult.

4. Conclusions

CA are an attractive approach for cryptographic applications. They are simple, modular logic systems that can generate good quality pseudorandom bit streams as required in robust cryptographic systems. A further advantage is that they can be easily and efficiently implemented in VLSI, and could thus find applications in many areas. The system we have described is based on nonuniform one-dimensional or two-dimensional CA. Comparing it with the only other two proposals known to the authors shows that it appears to be more secure, since reconstructing the key demonstrably requires an amount of computational resources which is an exponential function of the key length. Of course, we cannot exclude that the system could be decrypted in reasonable time and with reasonable computing resources. The stream generator can be easily programmed and analyzed from the description given here and it is thus open to scrutiny.

In this respect, it is also worth noting that nearly all known systems are insecure in absolute terms, the important consideration being the tradeoff between the time and the resources available to the cryptanalyst and the lifetime of the encryption key. With our system, the key can be changed at will and on the fly, although this in turn raises the question of the key exchange between parties. But this problem can be solved by using standard public-key cryptography for exchanging one-time session keys for use with the CA-based stream scheme presented here as frequently as needed. Of course, theoretically, even RSA public-key cryptography is not absolutely safe since the difficulty of factoring, barring new and unlikely efficient algorithms, depends on the length of the key and on the amount of computing power that a cryptanalyst is able to deploy. This power is on the rise, which implies that longer and longer keys have to be used. In fact, public-key schemes could easily be cracked today if efficient quantum computing algorithms devised for the task could be implemented in practice [10].

Finally, it has to be said that cryptographic systems that are both provably secure and physically realizable have been found. These are based on the cryptographic protocol invented in 1984 by C. Bennet and G. Brassard [1] which makes use of quantum information concepts. This approach is extremely promising but still difficult to use in practice due to the technological constraints of today's communication and storage devices. The practical advantage of the CA described here is that they are very easy and cheap to implement both in hardware and software and could meet the needs of many applications in which a high degree of security is needed but provably absolute inviolability is not called for. Using VLSI implementations, very high speed encrypting and decrypting circuits can be built and incorporated in communication devices such as hand-held computers and mobile phones. In this respect, an exciting prospect would be to build them using modern reconfigurable circuits such as field-programmable gate arrays (FPGA). These circuits are becoming competitive with custom VLSI but, instead of being wired once and for all, their function can be easily and quickly reprogrammed on the fly. The advantage is that in this way programmability of the generator would be retained together with the implied security aspects, as well as speed of operation.

References

[1] C. Bennet, G. Brassard, and A. Ekert, "Quantum Cryptography," *Scientific American*, **267** (4) (October, 1992) 50–57.

[2] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata: Theory and Applications, Volume 1* (IEEE Computer Society, Los Alamitos, CA, 1997).

[3] P. Guam, "Cellular Automaton Public Key Cryptosystem," *Complex Systems*, **1** (1987) 51–56.

[4] H. Gutowitz, "Cryptography with Dynamical Systems," in *Cellular Automata and Cooperative Phenomena*, edited by E. Goles and N. Boccara (Kluwer Academic Publishers, Boston, 1993).

[5] P. D. Hortensius, R. D. McLeod, and H. C. Card, "Parallel Random Number Generation for VLSI Systems Using Cellular Automata," *IEEE Transactions on Computers*, **38** (10) (October, 1989) 1466–1473.

[6] P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller, and H. C. Card, "Cellular Automata-based Pseudorandom Number Generators for Built-in Self-test," *IEEE Transactions on Computer-Aided Design*, **8** (8) (August, 1989) 842–859.

[7] W. Meier and O. Staffelbach, "Analysis of Pseudorandom Sequences Generated by Cellular Automata," in *Advances in Cryptology: Eurocrypt*

'91, Volume 547 of *Lecture Notes in Computer Science*, pages 186–199 (Springer-Verlag, Heidelberg, 1991).

[8] S. Nandi, B. K. Kar, and P. P. Chaudhuri, "Theory and Applications of Cellular Automata in Cryptography," *IEEE Transactions on Computers*, **43** (12) (December, 1994) 1346–1357.

[9] B. Schneier, *Applied Cryptography* (John Wiley & Sons, New York, 1996).

[10] P. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," in *Proceedings of the 35th Annual Symposium of Foundations of Computer Science*, pages 124–134, 1994.

[11] M. Sipper and M. Tomassini, "Generating Parallel Random Number Generators by Cellular Programming," *International Journal of Modern Physics C*, **7** (2) (1996) 181–190.

[12] M. Tomassini, M. Sipper, and M. Perrenoud, "On the Generation of High-quality Random Numbers by Two-dimensional Cellular Automata," to appear in *IEEE Transactions on Computers*.

[13] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud, "Generating High-quality Random Numbers in Parallel by Cellular Automata," *Future Generation Computer Systems*, **16** (1999) 291–305.

[14] S. Wolfram, "Cryptography with Cellular Automata," in *Advances in Cryptology: Crypto '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 429–432 (Springer-Verlag, Heidelberg, 1986).

[15] S. Wolfram, "Random Sequence Generation by Cellular Automata," *Advances in Applied Mathematics*, **7** (June, 1986) 123–169.