Lattice Gas Symmetric Cryptography

Laurent Signac

Lattice Gas Symmetric Cryptography

Laurent Signac

Laboratoire d'Informatique et d'Automatique pour les Systèmes
École Nationale Suprieurs d'Ingnieurs de Poitiers
Université de Poitiers, France

Abstract

Lattice gas cellular automata (LGCA) are particular cellular automata that imitate the behavior of particles moving on a lattice. We used a particular set of LGCA rules, called HPP, to mix bits in data blocks and obtain a symmetric cryptographic algorithm. The encryption and decryption keys are the positions of perturbation sites on the lattice (walls). Basically, this paper presents an original way to perform cryptographic operations, based on cellular automata. In this paper, we show several characteristics about our algorithm: typical block size (2^{2n-1}), key-length (2^n), number of rounds (2^{n+1}). We also evaluate avalanche and strict avalanche properties with respect to key and plain text. Finally, we highlight the underbellies of our method and give clues to solve them.

Keywords: cryptography, cellular automata, lattice gas, block cipher

Introduction

We propose to use a particular kind of cellular automata, called lattice gas cellular automata (LGCA), as a symmetric cryptographic algorithm: Encryption and decryption are achieved by the same algorithm and same key. One of the advantages of this algorithm is the number of parameters that can be modified, such as the size of data blocks, the size of keys, number of rounds... In the first section, we recall what is symmetric block cryptography and how the issue of cellular automata cryptography has been addressed yet. In section 2, we introduce lattice gas cellular automata which may be new to the cryptography community. In the third section, we detail how a particular set of rules, called HPP [1] can be used to encrypt any data. In section 4, we show how the different algorithm parameters (number of rounds, key size...) influence the ciphered block using gray levels images. In the last section, we check the avalanche and strict avalanche properties and point out the weaknesses of our method. Then, we conclude.

1. Cellular automata and Cryptography

Cryptography techniques may be divided into two categories: symmetric-key and public-key algorithms. Using symmetric key cryptography means that either the same key is used for encryption and decryption or it is easy to obtain one key from another.

On the contrary, if the sender and the receiver do not use the same key, and it is hard to compute the receiver's key from the sender's one, we use asymmetric cryptography.

This paper deals with the first category algorithms. More precisely, symmetric cryptography falls into stream or block ciphers. A stream cipher encrypts one byte at a time whereas a block cipher breaks up the message into fixed length blocks and encrypts one block at a time. This work is about a cellular automaton based symmetric block cipher.

Connecting cryptography and cellular automata (CA) is not new. But every cryptography topic do not have the same relationship with cellular automata. First of all, during the 80's, Wolfram [2] showed that CA were good pseudo random number generators. As random number generators have a very important role in stream cipher algorithms, CA based stream cipher algorithms have been widely studied [3, 4, 5, 6, 7]. The common principle of these works is to use the initial CA state as a seed and to run the CA in order to generate a random sequence. This sequence is then used in the stream cipher algorithm (Vernam cipher for instance).

Email address: laurent.signac@univ-poitiers.fr (Laurent Signac)

On the opposite, CA based public key cryptography literature is not abundant. See for instance [8, 9, 10].

Between the two opposites, several papers tackle the issue of symmetric block encryption. Most of them use a key built on the CA rules [9, 11].

From this point of view, our approach is quite different. Indeed, in our algorithm, keys are encoded in the CA state rather than in the CA rules. This is due to the particular kind of CA we used: Lattice Gas Cellular Automata.

Nevertheless, we recently found a very similar approach in [12] (the present work was initiated a (very) long time ago...). In paper [12], the authors also propose a block cipher symmetric cryptography based on LGCA. Contrary to us, they do not focus on a particular LGCA but tackle the issue in a more general way. The secret key in [12] is XORed each step with the lattice state (see section 2.1) whereas we added a particular step called *reflection* in our algorithm to process the key (to a certain extend, the two approaches are similar, although our is more *visual*, but less general). Besides, some remarks are similar : number of rounds, propagation of error. Finally, differential cryptography issue is tackled in [12] whereas we focused on avalanche and strict avalanche properties.

2. Lattice Gas Cellular Automata

The world of CA [13, 14] gave birth to particular simulation methods called Lattice Gas Cellular Automata (LGCA) [15]. LGCA are designed to mimic the moves of many particles on a grid. The first LGCA has been proposed in 1973 by Hardy, Pazzis and Pomeau [1]. It is this particular LGCA called HPP that we use in this paper.

2.1. HPP

HPP is a two-dimensional cellular automaton living on a square lattice. The Von Neumann neighborhood is used: each cell has four neighbors. Each cell may be occupied by up to four particles: the east, the north, the west and the south particle.

Figure 1 shows a lattice made of 6 cells. The upper-left corner cell is occupied by the north and the east particles. Its right neighbor is empty and its bottom neighbor is occupied by the four particles.

It is convenient to represent the state of a cell by a four bits number: the most significant bit will be the east particle, followed by the south, west and north particles.

The upper line of Figure 1 can be represented as: 1001 0000 1010 ($9\,0\,A$ in hexa).

Evolution of particles in HPP is a two steps algorithm. First, particles collide in a cell, then particles propagate.

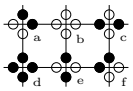


Figure 1: A sample lattice with 6 cells

Among the 16 possible cell configurations, particles do collide in only two particular configurations: 1010 and 0101. For instance, on Figure 1, particles in cell c and e will collide. No collision will occur in other cells.

A west-east collision will give a north-south configuration, and a north-south collision will give a west-east configuration.

After the collision step, the lattice of Figure 1 will turn into the lattice of Figure 2.

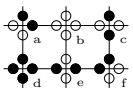


Figure 2: The sample lattice (fig. 1) after the collision step

The propagation step follows. During this step, every particle will move. The north particle of a given cell will move up (and remain a north particle). The west particle will move left (and remain a west particle) *etc.*

After the propagation step, the lattice of Figure 2 will turn into the lattice of Figure 3.

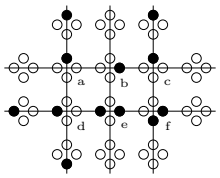


Figure 3: The sample lattice (fig. 1) after the collision (fig. 2 and the propagation step

We have seen how particles move on a 2-dimensional infinite lattice. The HPP rules may also be used on a finite lattice by connecting the upper and lower bounds as well as the right and left bounds of the lattice (torus topology).

Figure 4 shows the first evolution rounds of a 4×4 toric lattice. As mentioned above, the set of cells may be represented as set of 16×4 bits *i.e.* 8 bytes. The top-left corner cell will be the most significant 4 bits of the first byte. The cell on its right will be the less significant 4 bits of the first byte and so on. In the first step, the top left corner site contains west and north particles ($1001 \rightarrow 9$). Its right neighbor contains no particle (0). The third site of the top line contains east and west particle ($1001 \rightarrow A$). Le last site only contains the west particle ($0010 \rightarrow 2$).

Here are the 8 bytes representing the initial state, and the 2 states obtained after the propagation step (hexadecimal numbers):

- `90 A2 F5 15 5D 10 00 00`
- `18 30 A9 F2 48 82 14 10`
- `17 90 02 A8 50 F8 50 10`

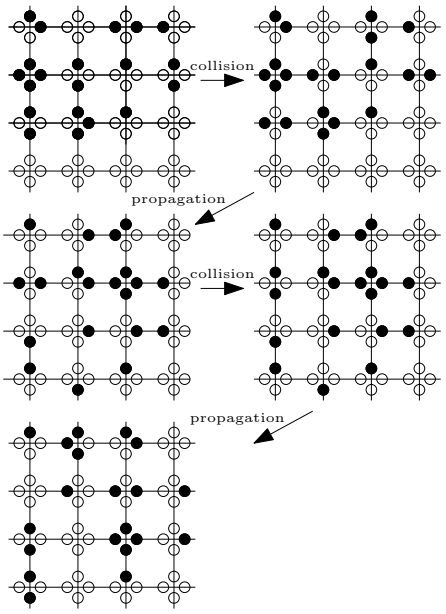


Figure 4: Particles evolving on a toric lattice

Now, we know how to make a set of bytes evolve. Let us add perturbation sites.

2.2. Obstacles Reflections

We now introduce special cells known as 'walls' or 'obstacles', and a step called 'reflection'. After the collision and propagation steps, direction of particles that live on a wall cell are inverted, so that the north and south, as well as east and west are swapped.

Figure 5 shows an example of a 3×2 lattice with two 'walls' on cells b and d.

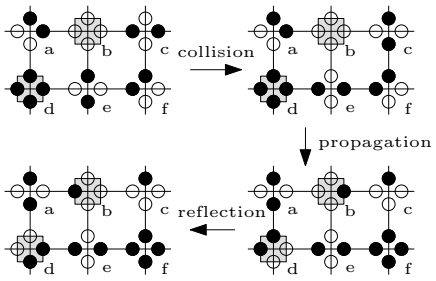


Figure 5: Collision, propagation and reflection step on two walls

3. HPP Symmetric Cryptography

In this section, we show how HPP rules could be used as a symmetric cryptographic algorithm. We first recall that HPP simulations are reversible. Then we detail the steps of the cryptographic algorithm. Finally, we propose to use obstacles positions as a secret key.

3.1. Time Reversibility

The most important property of HPP rules we used to design the cryptographic algorithm is time reversibility. In other words, we used the fact that, during particles evolutions, no information is lost, and the initial configuration can be computed from any subsequent configuration.

Even though we focused on HPP rules, any LGCA set of rules could make the job. The condition is that the set of rules is time reversible. This is not the case for FHP rules [16], on an hexagonal lattice: the collision step, for physical realism reasons (chirality), is probabilistic. As we are not interested in physical realism, FHP rules could be modified to be time reversible.

3.2. Cryptographic Algorithm

The way we used HPP as a cryptographic algorithm may now be clear:

1. Choose a lattice size. It may be convenient to choose a power of two. For instance 256×256.
2. Segment raw data into blocks to populate the lattice. For a 256×256 grid, we need 32768 bytes data blocks, as every cell is populated with 4 bits.
3. Choose a secret key, and use this secret key to compute the walls positions. The way how a secret key is chosen and how it is used will be tackled in section 3.3.
4. Choose n, the number of rounds that will be computed.
5. For each clear data block:
   - Apply n times: collision, propagation, reflection.
   - Reverse time (to get an involutive algorithm).
   - Get the encrypted data block.

Decryption is achieved by applying exactly the same algorithm to encrypted data blocks, provided each parameter is the same:

- block size
- secret key
- number of rounds

3.3. Secret Key

The secret key is the position of the walls on the grid. If the lattice size is a power of two, say $2^n \times 2^n$, the coordinates of a cell may be represented with $2 \times n$ bits. As a consequence, for a $2^n \times 2^n$ lattice, and K walls, the size of the key must be $2nK$ bits. However, the number of *real* keys is not 2^{2nK}, as exchanging certain walls positions might give the same key, and several walls may be in the same cell. The number of walls configurations is the number of multisets of cardinality K of elements taken from a set of cardinality 2^{2n}:

$$\left(\!\!\binom{2^{2n}}{K}\!\!\right) = C_K^{2^{2n}+K-1} = \frac{(2^{2n}+K-1)!}{K!(2^{2n}-1)!}$$

The number of potential secret keys is quite important. If we use a 256×256 lattice ($n = 8$), and 256 walls ($K = 256$), the number of different walls configurations is $C_{256}^{256^2+256-1} \approx 2 \times 10^{726}$. For a small 64×64 lattice ($n = 6$, 2048 bytes data blocks) and 32 walls (K=32), the number of available keys is about 1.6×10^{80}.

4. Experimental Results : LGCA point of view

In this section, we show how to choose a key and the number of rounds. We also point out the problem of invariants which may be a problem in cryptographic applications whereas it is a desirable behavior in LGCA.

In this section, we used pictures as the properties we want to show can be well understood by thinking of our algorithm as a lattice-gas cellular automaton. Nevertheless, one must keep in mind that our block cipher is not particularly designed to encrypt pictures, but any data block.

Each cell of the automaton may be represented by a 4 bits number. We use this number as a gray level. Thus, a $2^n \times 2^n$ lattice populated with particles is turned into a 16 gray levels image of size $2^n \times 2^n$.

In the following, we used two 64×64 images (Figure 6).

(a) Eiffel tower (b) A squirrel

Figure 6: Images used in our tests

4.1. Number of Rounds

First, we encrypt our image, with only one wall (look at the (very) small cross) in the center (Figure 7). Then, we remove the wall, and try to decrypt the picture. We first make this experiment with a 16 rounds algorithm (Figure 8), and then with a 128 rounds algorithm (Figure 9).

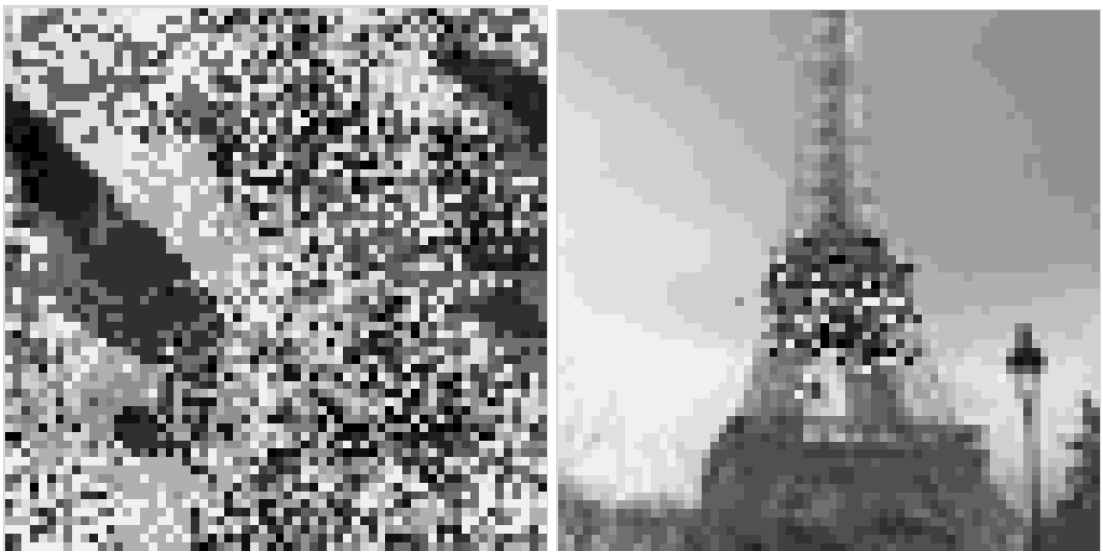
We see that if the number of rounds is not large enough, a part of the image can be decrypted without the key. This is an evidence that a wall can not influence cells that are too far. More precisely, by computing N rounds, cells that are outside the discus of radius N (we consider the Hamming distance) and centered on the wall will not be affected by the wall. Cells that are inside the discus may or may not be affected: it depends on the particles positions, that is not known *a priori*.

The consequence is that we must choose a large enough value for N so that every cell is affected by every walls, whatever the position of the walls.

Let us choose a number of rounds such that *every* cell can be affected by every wall. For a $2^n \times 2^n$ lattice, the maximal Hamming distance is 2^n (keep in mind the lattice is on a torus). By choosing $N = 2^n$, we avoid the problem of key-unaffected bits. Unfortunately, computing 2^n rounds may be time consuming.

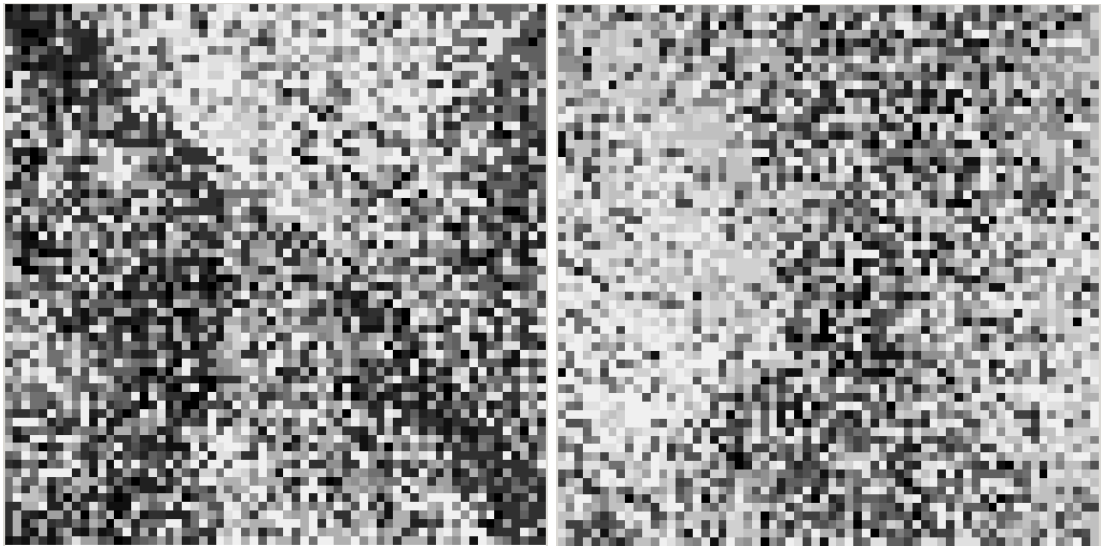
One may think that, because of the large number of walls, it is possible to reduce the number of rounds. It would not be a good idea.

Figure 7: Clear picture, with a central wall (small white cross)



(a) Encrypted picture (b) Decrypted picture, without the central wall

Figure 8: Encrypted picture, and the decrypted picture obtained by omitting the central wall (16 rounds)



(a) Encrypted picture (b) Decrypted picture, without the central wall

Figure 9: Encrypted picture, and the decrypted picture obtained by omitting the central wall (128 rounds)

Here is another example: the squirrel of Figure 10(a) is encrypted with a 13 walls key. Unfortunately, the keys are not well spread and there is no wall next to the squirrel's head. Using a number of rounds too low (say 24), we obtain the encrypted picture of Figure 10(b). It is then possible to decrypt a part of the image (squirrel's head) by applying the algorithm with a key that does not contain a wall next to the head. We then obtain the picture of Figure 10(c).

This would be impossible with a higher number of rounds.

The only way to avoid (to a certain extend) the decryption of parts of an image by only knowing parts of the key, is to choose a number of rounds such that each wall influences every cell.

By using a key of about 2^{n-1} walls ($2^n \times 2^n$ is the size of the image), and a number of rounds of 2^n, the algorithm would be very sensible to key variations. Figure 11(a) shows the 64×64 squirrel encrypted with a 32 walls key and 64 rounds.

This encrypted image is decrypted by a key that differs only on one wall, that is shifted right one cell. The 'decrypted' picture is shown on Figure 11(b)

In section 5, we will see that, for a 64×64 lattice (2048 bytes), a good number of rounds is in fact 128.

4.2. Algorithms Invariants

Hpp, as well as most Lgca rules, have been designed to simulate physical systems. One characteristic is that Lgca obey physical laws such as mass conservation, momentum conservation...

For our particular application, this is a great disadvantage. For Hpp model, mass and momentum are the same, because each particle has the same velocity. More informations about real or spurious invariants in Lgca can be found in [15].

For our application, it means that the number of bits equal to 1 remains constant in a data block. This is very annoying, especially if the number of 1's (or 0's) is very low.

On our gray levels pictures, it means that a black picture will remain black, whatever the number of walls, their positions, and the number of rounds...

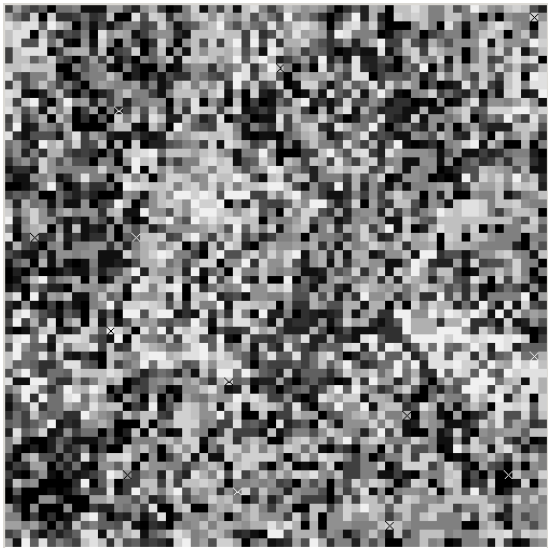
We conclude that the algorithm must not be used on data blocks that contain redundant informations.

Let us consider the issue in this way: the number of bits equal to 1 in the clear block can be deduced from the number of bits equal to 1 in the encrypted block. How to make this information useless? We could for instance only consider data blocks where the number of 1's is *always* about half the number of bits. If the algorithm is only applied on such blocks, the knowledge of the number of 1's becomes useless.

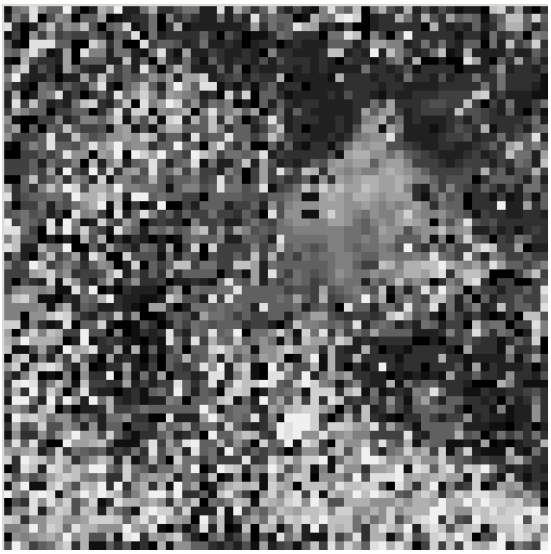
Consequently, we need a preprocessing step that turns any data blocks into data blocks containing as many 1's as 0's. This job may be done with almost any compression algorithm, that theoretically produce random-like data (Gnu/gzip utility (Lempel-Ziv algorithm) for instance).

Using compression mainly solves the mass invariance of Lgca. Furthermore it speeds up encryption.

(a) Original image. No wall next to the head

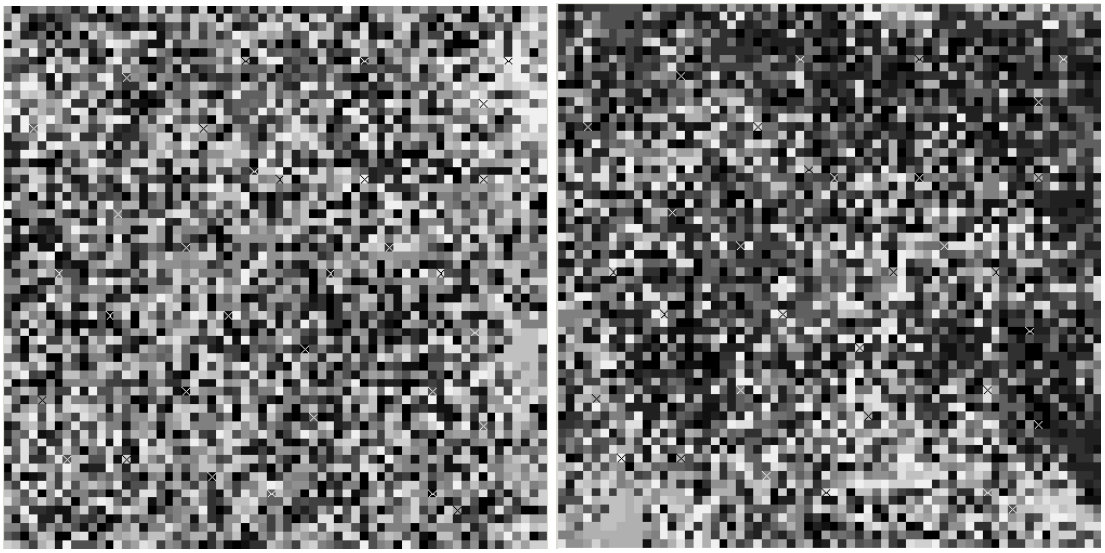


(b) Encrypted image with 24 rounds



(c) Decrypted image with a wrong key

Figure 10: A part of the image can be decrypted if the number of rounds is too low: the squirrel's head is visible



(a) Encrypted image with 32 walls and 64 rounds (b) Decrypted image with very slightly different key

Figure 11: With a large key and a large number of rounds, the algorithm is very sensible to key variations

5. Experimental Results : cryptography point of view

5.1. Avalanche property

Any encryption algorithm should satisfy the following property: a small change in the key or in the plain text should result in a great modification of the ciphered text. More precisely, if we invert one bit in the plain text or in the key, we expect that nearly half ciphered text bits are inverted. This property, called *avalanche property* has been introduced in [17].

To show the avalanche property, we encrypt 2048 bytes blocks (64×64 lattice). The key is 56 bytes long.

We used the following algorithm to produce the Figure 12 that shows the percentage of bits that have been inverted as a function of the number of rounds[1]:

$t \leftarrow$ 2048 random bytes
$k \leftarrow$ 48 random bytes
for $r = 10 : 10 : 200$ **do**
  $p \leftarrow 0$
  $c_{ref} \leftarrow$ encrypt(t, k) with r rounds
  for $i = 0$ **to** 383 **do**
    $k_2 \leftarrow k$ with bit i inverted
    $c_2 \leftarrow$ encrypt(t, k_2) with r rounds
    $p \leftarrow p$+fraction of inverted bits between c_{ref} and c_2
  end for
  $p \leftarrow p/384$
  plot the point (r, p)
end for

Figure 12 shows that avalanche property is satisfied with respect to the key.

In order to check the avalanche property with respect to the plain text bits, we used a similar algorithm, executed 20 times:

$t \leftarrow$ 128 random bytes
$k \leftarrow$ 8 random bytes
for $r = 10 : 2 : 200$ **do**
  $p \leftarrow 0$
  $c_{ref} \leftarrow$ encrypt(t, k) with r rounds
  for $i = 0$ **to** 1023 **do**

[1] This algorithm has been executed 5 times, Figure 12 represents the average results

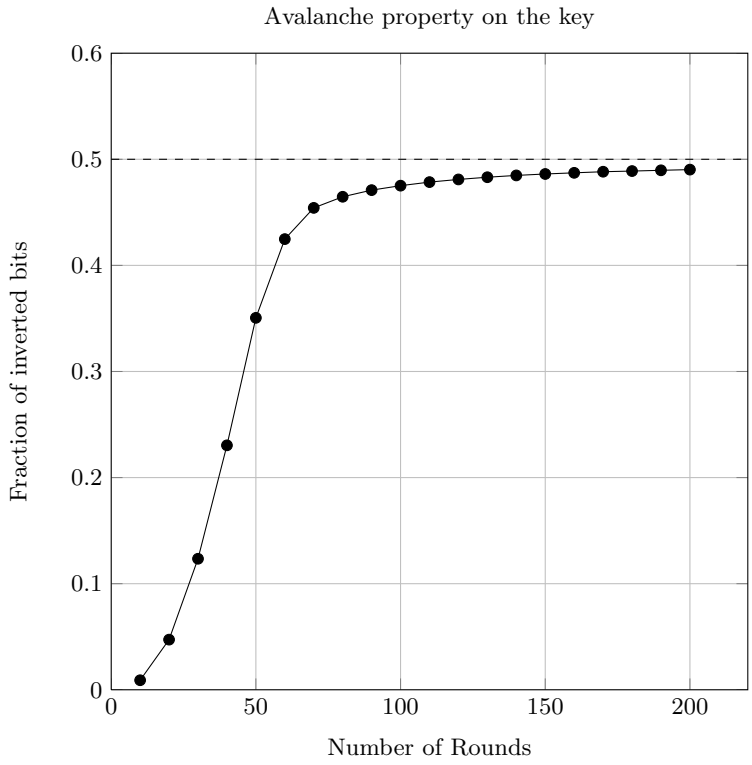


Figure 12: Avalanche property if we change one bit of the key

$t_2 \leftarrow t$ with bit i inverted
$c_2 \leftarrow \text{encrypt}(t_2, k)$ with r rounds
$p \leftarrow p$+fraction of inverted bits between c_{ref} and c_2
end for
$p \leftarrow p/1024$
plot the point (r, p)
end for

Figure 13 shows the surprising results. Only 25 percent of output bits have been changed. This annoying phenomenon is explained in the following subsection.

One can think that the distribution of walls is of great importance. We checked the behaviour of the avalanche property for key walls restricted to a small area of the grid (a 8×8 subgrid, the whole grid being a 64×64 grid). Figure 14 gives the results compared to what we obtained on Figure 12. We can see that increasing the round numbers counterbalances the non-random repartition of walls.

5.2. Strict avalanche properties

Another important property is that, for every input value (key and plain text), a single bit change must affect the whole ciphered text: every output bit must depend upon all input bits. This property, known as completeness has been combined with the avalanche property: each output bit should change with a probability of one half whenever a single input bit is complemented. This property is known as *strict avalanche property* [18].

Let us look closer at this property. We have to check it for all input bits. First we are going to check it for the key bits, and then for the plain text bits.

The following experiments were made with small data blocks (128 bytes→ 1024 bits), and small keys (8 bytes). The number of rounds was 64, and the following algorithm has been executed for $N = 1\,000$.

$v \leftarrow 0, 0, 0..., 0, 0, 0$
for $n = 1$ **to** N **do**
$p \leftarrow$ 128 bytes random plain text
$k \leftarrow$ 8 bytes random key
$c \leftarrow \text{encrypt}(p, k)$
for $i = 0$ **to** 63 **do**

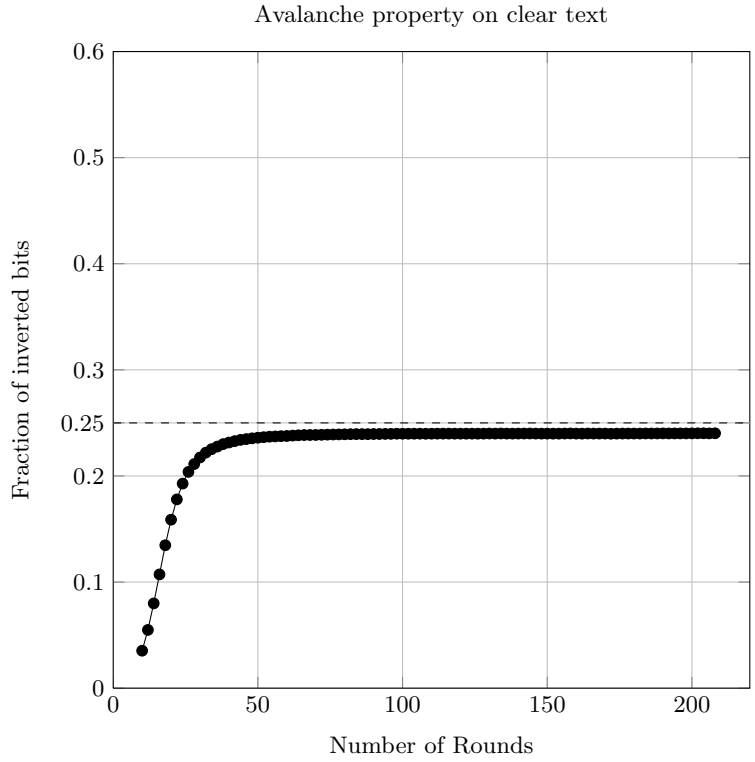


Figure 13: Avalanche property if we change one bit of the plain text

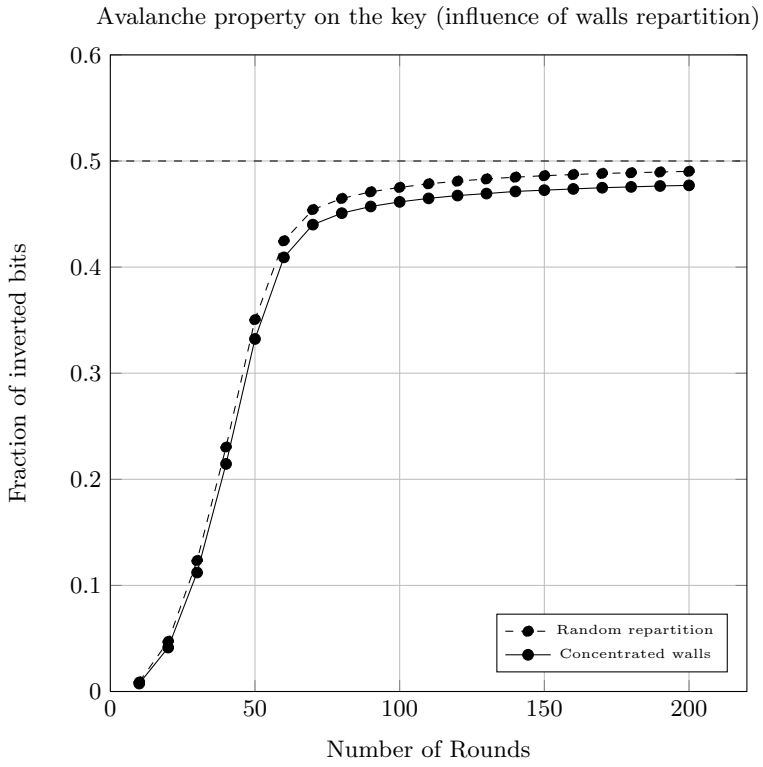


Figure 14: Avalanche property: comparison between a random repartition of walls and a configuration with walls concentrated in a small surface

$k_2 \leftarrow k$ with bit i inverted
$c_2 \leftarrow \text{encrypt}(p, k_2)$
$v \leftarrow v + c$ xor c_2 $\{v, c, c_2$ are arrays$\}$
end for
end for
$v \leftarrow v/(N \times 64)$

The results, given on Figure 15 show that if we change one bit of the key, the probability for *every* ciphered bit to be inverted is about 0.47.

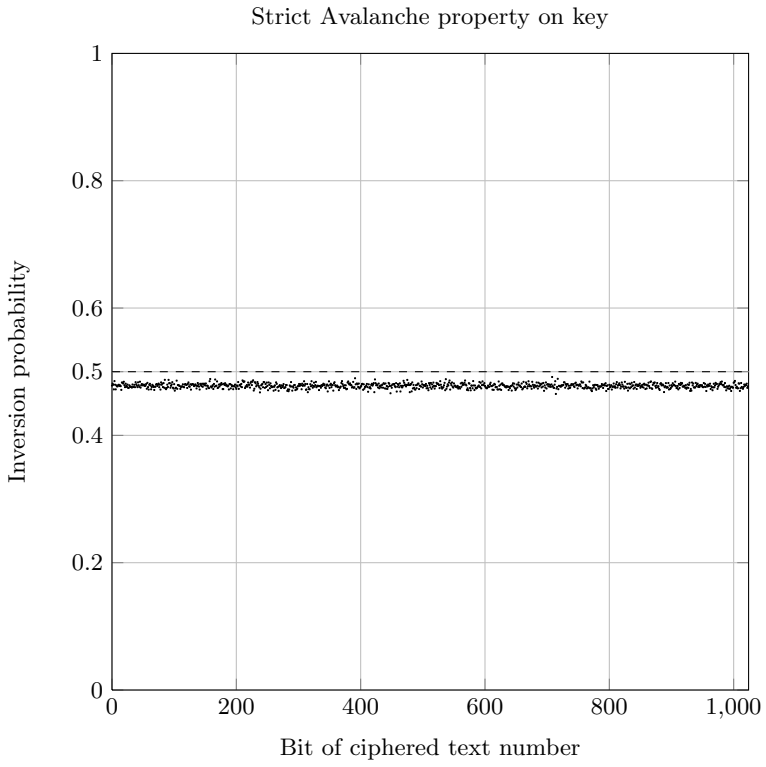


Figure 15: Influence of changing a bit in the key : each output bit has a probability of about 0.47 to be inverted

Now, let us take a look at the influence of plain text bits. The following algorithm has been used ($N = 1\,000$):

$v \leftarrow 0, 0, 0..., 0, 0, 0$
for $n = 1$ **to** N **do**
$p \leftarrow$ 128 bytes random plain text
$k \leftarrow$ 8 bytes random key
$c \leftarrow \text{encrypt}(p, k)$
for $i = 0$ **to** 1023 **do**
$p_2 \leftarrow p$ with bit i inverted
$c_2 \leftarrow \text{encrypt}(p_2, k)$
$v \leftarrow v + c$ xor c_2 $\{v, c, c_2$ are arrays$\}$
end for
end for
$v \leftarrow v/(N \times 1024)$

Figure 16 shows the result. Again, for a bit inversion in the clear text, the probability for a bit in the ciphered text to be inverted is 25% instead of 50%.

The can be explained if we take a closer look at the Hpp crypto algorithm structure. Think of the lattice as a chessboard. A single plain text bit may influence either the black or the white cells. More precisely, if complementing bit K influences black cells then, complementing bit $K + 4$ will influence white cells. Using $r + 1$ rounds instead of r will invert black and white. The truth is that a bit inversion in the clear text does not produce a bit inversion of every ciphered bit with a probability 0.25. On the

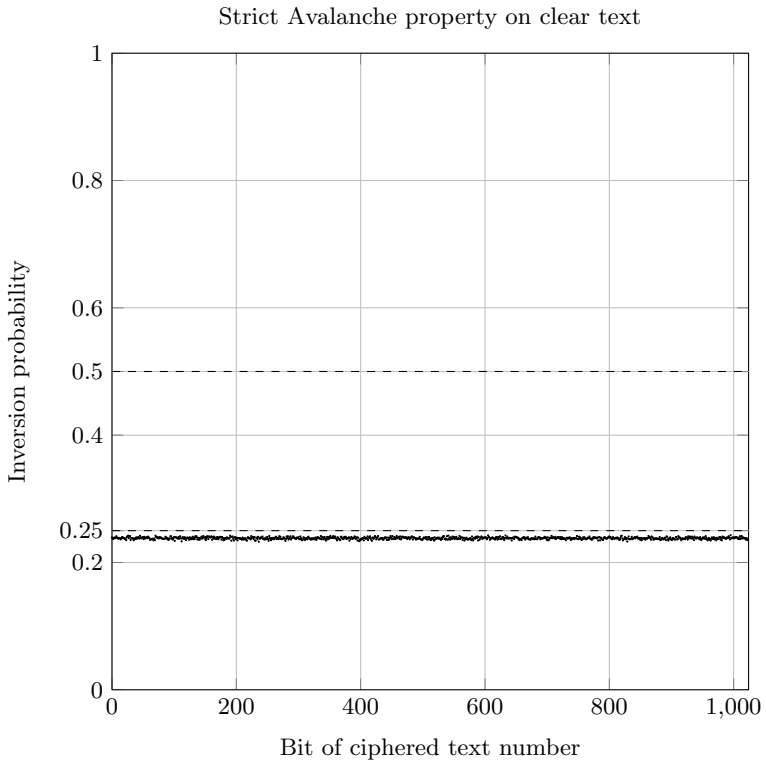


Figure 16: Influence of a bit inversion on plain text

contrary, a bit inversion in the clear text produce a bit inversion in the ciphered text with a probability 0.5, for *half the bits*.

Figure 16 shows that strict avalanche property *is not* true for this HPP crypto algorithm. Instead it is true for half the cells.

This can be emphasized if we reproduce the strict avalanche experiment but only invert bit 0 (for instance) of the plain text. The result is given on Figure 17.

6. Conclusion

In this paper, we presented an original algorithm for symmetric encryption based on the HPP lattice gas cellular automaton. The principle is to make data blocks evolve on a perturbed lattice. The perturbations locations play the role of the encryption key.

It is proposed to use a large number of rounds, and to compress data, before the encryption step. In any case, this algorithm should not be used as such.

The fact that LGCA are used to simulate complex dynamic systems [19, 12] lead us to think that there is no obvious shortcut to compute the final configuration of the cryptographic algorithm. Let us point out the pros and cons of HPP cryptography:

- Very large key space
- Avalanche and strict avalanche properties are satisfied for key bits
- Avalanche and strict avalanche properties are not satisfied for plain text bits. Nevertheless, the properties are satisfied for half the encrypted block.
- HPP cryptography is slow. But parallel computation of cellular automata is conceivable.
- Size of encrypted blocks can be modified. Using large blocks increases size of the key space.

We plan to work on a modified (time reversible) FHP version of this algorithm as using an hexagonal lattice is likely to solve the "checker problem" and give better strict avalanche property results.

[1] J. Hardy, Y. Pomeau, O. de Pazzis, Time evolution of a two-dimensional model system, J. Math. Phys 12 (1973) 1746–1759.

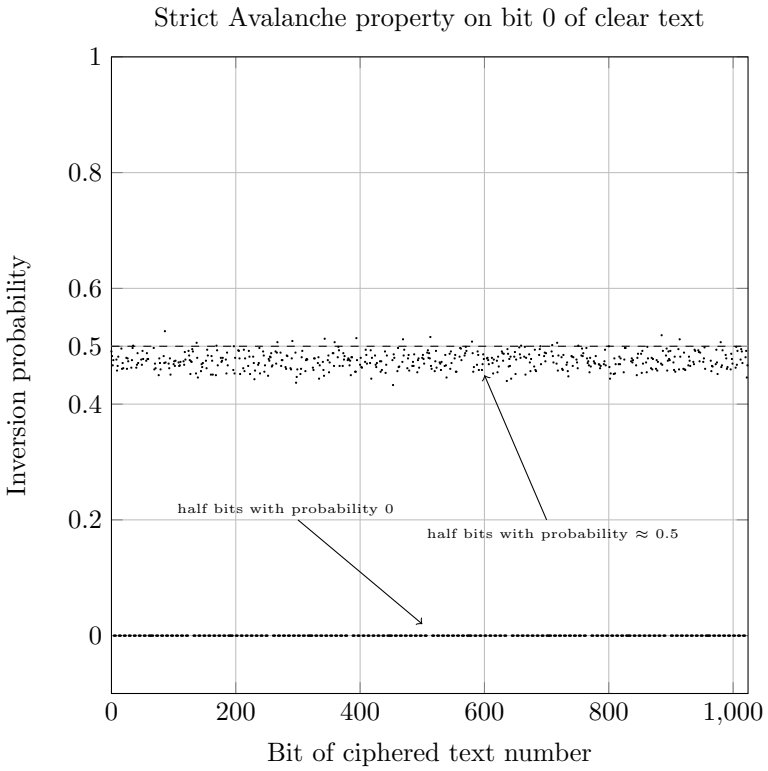


Figure 17: Influence of changing a single bit in plain text : only half of the bits is influenced

[2] S. Wolfram, Random sequence generation by cellular automata, Adv. Appl. Math 7 (1986) 123–169.
[3] F. Seredynski, P. Bouvry, A. Y. Zomaya, Cellular automata and symmetric key cryptography systems, in: Springer (Ed.), Genetic and Evolutionary Computation - Gecco, 2003, pp. 1369–1381.
[4] M. Tomassini, M. Parrenoud, Stream ciphers with one and two dimensionnal cellular automata, in: Springer (Ed.), Parallel Problem Solving from Nature - PPSN, 2000, pp. 722–731.
[5] M. Tomassini, M. Parrenoud, Cryptography with cellular automata, Applied Soft Computing (2001) 151–160.
[6] S. Wolfram, Cryptographyy with cellular automata, in: Springer (Ed.), Advances in Cryptology : Crypto'85, 1985, pp. 428–432.
[7] F. Seredynski, P. Bouvry, A. Y. Zomaya, Cellular automata computations and secret key cryptography, Parallel Computing 30 (5-6) (2004) 753–766.
[8] P. Guan, Cellular automaton public-key cryptosystem, Complex Systems 1 (1987) 51–56.
[9] J. Kari, Cryptosystems based on reversible cellular automata, Tech. rep. (1992).
[10] H. Gutowitz, Cryptography with dynamical systems.
[11] M. Seredynski, P. Bouvry, Block Encryption Using Reversible Cellular Automata, Vol. 3305, Springer, 2004, pp. 785–792.
[12] S. Marconi, B. Chopard, Discrete physics, cellular automata and cryptography, in: 7th International Conference on Cellular Automata for Research and Industry, ACRI, Vol. 4173 of Lecture Notes in Computer Science, 2006, pp. 617–626.
[13] S. M. Ulam, Random processes and transformations, in: Proceedings of the International Congress of Mathematicians, 1952.
[14] J. V. Neumann, Theory of Self-Reproducing Automata, University of Illinois Press, 1966.
[15] D. A. Wolf-Gladrow, Lattice-Gas Cellular Automata and Lattice Boltzmann Models, Springer, 2000.
[16] U. Frisch, B. Hasslacher, Y. Pomeau, Lattice-gasz automata for the navier-stockes equation, Phys. Rev. Lett. 14 (1986) 1505–1508.
[17] H. Feistel, Cryptogrpahy and computer privacy, scientific american 228 (5) (1973) 15–23.
[18] A. Webster, S. Tavares, On the design of s-boxes, in: Advances in cryptology : Crypto'85, Springer, 1985, pp. 429–432.
[19] B. Chopard, M. Droz, Cellular automata Modeling of Physical Systems, Cambridge University Press, 1998.