THE COMPOUND TENT MAP AND THE CONNECTION BETWEEN GRAY CODES AND THE INITIAL CONDITION RECOVERY

by Alexandru Dinu[1] and Adriana Vlad[2]

We reconsider some aspects regarding the usage of tent map in chaos based cryptography. The first part of the paper discusses the precision and how difficult or not is to recover the initial condition as a function of the way the binary sequence used in the recovery process is obtained. The answer opens the way for an introspection in the problem of the compound tent map. We will use the Gray codes to find the compound tent map and we will see how these codes are already subtle hidden in the binary sequence that we have.

Keywords: tent map, initial condition recovery, compound tent map, Gray codes

1. Introduction

In this paper, we thoroughly analyse the recovery of the initial condition issue for tent map, recovery based on the binary representation of sequences of values generated by tent map. We shall show how precisely we can find the initial condition when the discretisation threshold is equal to the tent map control parameter, the imprecision in the contrary case (threshold different from the control parameter) and the obstacle represented by tent map sampling. The basic tool that will be used in the mathematical argumentation of these results will be the compound tent map. There are some interesting studies in the literature referring to finding the initial condition and to the precision of this (see [1, 2, 3]), but our demonstration facilitates this mathematical approach, leaving room for eventual new ways to use the compound tent map.

Tent map is a 1-dimensional discrete time chaotic system defined by the following equation:

$$x_{k+1} = \begin{cases} \frac{x_k}{p} & \text{if } 0 \leq x_k \leq p, \\ \frac{1-x_k}{1-p} & \text{if } p < x_k \leq 1, \end{cases} \tag{1}$$

[1] Technische Universität München, Germany, e-mail: alexandru.dinu@tum.de, alexandrudinu89@yahoo.com

[2] Faculty of Electronics, Telecommunications and Information Technology, Politehnica University of Bucharest, Romania; The Research Institute for Artificial Intelligence, Romanian Academy, Bucharest, Romania, e-mail: avlad@racai.ro, adriana_vlad@yahoo.com

where $p \in (0,1)$, p is the tent map parameter. Due to the ergodicity and to the sensitivity to the initial condition and to the p control parameter (which has to be different from 0.5, otherwise by applying (1) with $p = 0.5$ we obtain a sequence that is not chaotic any more), completed by the uniform probability law of the x_k values, the tent map can be successfully used in chaos-based cryptographic applications. In most of the studies that have tent map as a subject, this chaotic signal is used as a generator of enciphering sequences, e.g., [3, 4, 5]. The enciphering sequence ("key") is obtained through binarization with a certain threshold c of a trajectory (the successive iterations of the tent map) defined by (1).
If $x_k \leq c$, we assign a binary value $b_k = 0$
If $x_k > c$, we assign a binary value $b_k = 1$
2.5mm

One way of creating the cryptogram is to sum modulo 2 (symbol by symbol) the clear message and the "key". So, the question that shows up is: **"Can someone, having a part (a string of bits) of the binary sequence, recover the initial condition x_0 that generated the respective trajectory of the tent map?"**. If this is the case, one can reconstruct the entire "key", no matter how long it is, and in this case the initial condition cannot be used as an element in the secret key.
2.5mm

Section 2 is an experimental study on the accuracy and the difficulty encountered when trying to recover the initial condition. The experimental study is theoretically supported by Section 3.
2.5mm

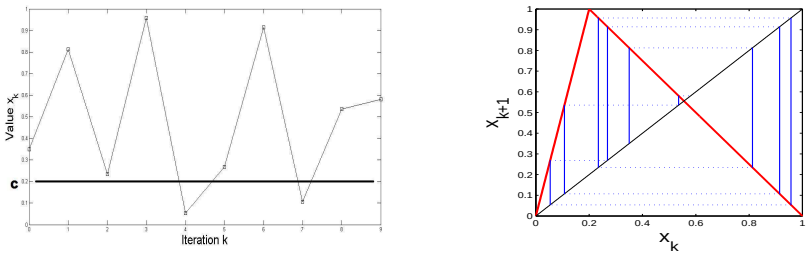
Section 3 deals with the compound tent map and the connection between Gray codes and initial condition recovery. The mathematical expression for the compound tent map is given and the proof for initial condition recovery in Section 2 is presented.
2.5mm

The last section summarizes the main issues and conclusions of the paper and states some new directions of research regarding the compound tent map.

2. How difficult is to find the initial condition?

We shall next clarify the initial condition recovery issue for two important cases in applications, i.e., $c = p$ and $c = 0.5$ (see [4, 5]), where c is the threshold used to obtain the enciphering binary sequence and p was introduced in (1).

2.1. Initial condition recovery when $c = p$. Let us have a trajectory for $p = 0.2$ and $x_0 = 0.35$. The first 10 successive real values $x_0, x_1, ..., x_9$ and the corresponding binary symbols $b_0, b_1, ..., b_9$ are shown in Table 1 and Figure 1.



(a) (b)

FIG. 1. The first 10 iterations for: (a) $x_0 = 0.35$ and $p = 0.2$ and (b) the corresponding Cobweb diagram.

Let us suppose that we know $b_0, b_1 ... b_m$ (the binary values of the "key" from *0* to m) and the value of p. To recover x_0 we have the following algorithm (let us call it "*"):

(1) Assign a real value to x_m corresponding to its binary symbol:

- If b_m is 0, assign to x_m a real value smaller than p (let us say $\frac{0+p}{2}$)
- Otherwise, assign to x_m a value larger than p, but smaller than 1 (for example, $\frac{1+p}{2}$)

(2) To find the previous term, x_{m-1}, take into account its binary value, b_{m-1}:

- If b_{m-1} is 0, go back using the 1st branch from (1): $x_{m-1} = p \cdot x_m$
- If b_{m-1} is 1, go back using the 2nd branch from (1): $x_{m-1} = 1 - (1-p) \cdot x_m$

(3) Iterate backwards in this way to x_0 (an estimate, for the moment, as we do not know the accuracy yet).

We will apply this algorithm to the previous example with $x_0 = 0.35$ and $p = 0.2$. The second row in Table 1 shows the binary sequence corresponding to the m-term trajectory (based on equation (1)). We will try to go back to x_0 from a certain real value for x_m. The third row in Table 1 shows the obtained results for the m-term trajectory of (1) when x_m is chosen according to the mentioned algorithm. In addition, we will see if the real value from which we iterate backwards has any effect on the precision in finding x_0 (Table 1, fourth row).

Table 1

The initial condition recovery based on the first 10 successive bits. *Recovered a)* trajectory starts iterating backwards from the value in * algorithm; *Recovered b)* trajectory starts from an arbitrary value (0.95, here)

Continuous	0.3500	0.81	0.23	0.95	0.05	0.26	0.91	0.10	0.53	0.58
Binary	1	1	1	1	0	1	1	0	1	1
Recovered a)	0.3498	0.81	0.23	0.95	0.05	0.26	0.91	0.10	0.52	0.60
Recovered b)	0.3469	0.8164	0.22	0.96	0.04	0.23	0.96	0.05	0.24	0.95

Table 2

Results obtained for initial condition recovery trial in decimal and hexadecimal for 3 different x_0 values, given the first successive 100 bits.

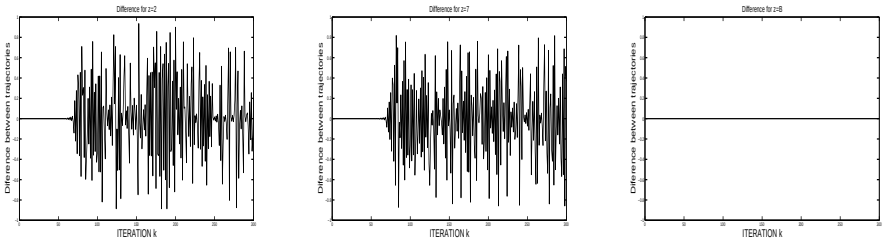
Trajectory number	Original x_0	Recovered x_0
	Decimal	
1	0.679824758432644	0.679824758432644
2	0.345762146343287	0.345762146343287
3	0.123465637517625	0.123465637517625
	Hexadecimal	
1	3FE5C11FDA0F567**6**	3FE5C11FDA0F567**7**
2	3FD620D78DAF4ED**1**	3FD620D78DAF4ED**0**
3	3FBF9B71AB5167C**B**	3FBF9B71AB5167C**C**

First of all, let us notice that even with 10 iterations backwards, we can go very close to the original x_0 (this fact will be more obvious from Table 2 where, after 100 iterations back, the estimated x_0 will be identical with the original one in decimal representation). Secondly, maybe even surprisingly, the value assigned to x_m is not very important (see Table 1, fourth row); the estimated initial condition value is not so much different compared to the true x_0 (this aspect will be treated separately in the last part of the article, as one of the most relevant results of the paper).

We want to see the precision of the initial condition recovery when we are given the first $m = 100$ terms of a trajectory of (1) and the p parameter (for $m > 100$ and $p = 0.2$, the improvement in precision is not significant). Table 2 shows the results obtained for different tent map trajectories (corresponding to 3 different x_0 initial conditions).

The differences are not obvious in decimal representation; in hexadecimal representation, although the first bits are identical, we have some inaccuracies in the last group of 4 bits (the last nibble). So, with this algorithm, we can find an estimated x_0 and then, by plugging in all the 16 possibilities, we have 16 different possible tent map trajectories. Now, supposing that we have the "key" in binary for a long number of iterations, we can compare this "key" with each of the 16 possible trajectories corresponding to the 16 modified x_0, and see which one matches what we have.

We exemplify for $p = 0.2$ and $x_0 = 0.123465637517625$ (last row from Table 2). We find an estimated x_0 and even though we do not know if this is or not the true initial condition, we calculate and compare the 16 possible trajectories with the initial one (the trajectory of the original x_0). The comparison is made in real values. The discrepancies shown below reveal that, from a certain iteration step in time, all the trajectories corresponding to the wrong x_0 will deviate so much from the true trajectory of the "key", that in binary values the differences will lead to the same conclusion: only one x_0 is the right one.



(a) (b) (c)

FIG. 2. The difference between the original ($z = 1011 = B(hexa)$) and 3 recovered trajectories: (a) $z = 0010 = 2(hexa)$; (b) $z = 0111 = 7(hexa)$; (c) $z = 1011 = B(hexa)$.

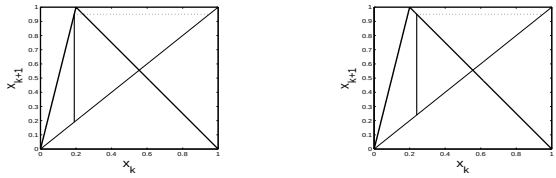
Be z the last nibble (a hexadecimal symbol). The original x_0 has $z = B = 1011$. We present the differences between the true trajectory obtained with this x_0 and 3 possible trajectories that can be obtained by replacing the last nibble z of the recovered x_0 in hexadecimal by 3 out of the 16 possible values from *0,...,F*.

To sum up, it is clear that there is a modality to come back to the original x_0 knowing the binary sequence of the "key" and the p value. In order to do this, we need to test only 16 possible initial conditions. Furthermore, we need only $m = 100$ binary symbols to come back. This is however dependent on p. For p closer to 0 or 1, we need $m > 100$. We will explain this in Section 3.

2.2. **Obstacles to the initial condition recovery.**

2.2.1. *The binarization threshold value.* **A first obstacle** in the way of finding x_0 is the case **when the threshold used for binarization is** $c = 0.5$. We first discuss the problem of x_0 recovery by considering all successive iterations, although this is not the usual case of interest in the literature (see [4, 5]). If $c = 0.5$ and we do not sample the tent map (1), the successive binary symbols are statistically dependent. But if we sample the binary sequence with a certain d (number of iterations) distance, we can reach statistical independence and flawlessly simulate the behaviour of the fair coin (see [4, 5, 6]).

Let us summarize the steps and see where the problems appear:
(1) Create a binary sequence obtained by discretizing a trajectory of the tent map (1) for $p = 0.2$. The discretization will be done with $c = 0.5$, and considering all successive iterations.
(2) It is possible to reach the following situation: $\ldots b_k = 0,\ b_{k+1} = 1 \ldots$
(3) Let us suppose that we apply the previous algorithm *. Note that $c \neq p$, so one may think that p is unknown. It is shown in [2], [4] that we can very precisely estimate p based on the binary sequence, therefore we apply algorithm * for p=0.2. The result of the algorithm could be $x_{k+1} = 0.95$. In this case, x_k can have two real values, both leading to x_{k+1}. We have the binary value for x_k, but this is useless now (considering the way we binarized



(a) (b)

FIG. 3. The two values (a) $x_k = 0.19$ and (b) $x_k = 0.24$ that could generate $x_{k+1} = 0.95$.

Table 3

Results obtained for initial condition recovery trial, in decimal and hexadecimal, for 3 different x_0 values, given the first 100 bits.

Trajectory number	Original x_0	Recovered x_0
Decimal		
1	0.123465637517625	0.172428258114886
2	0.814723686393179	0.844981651714274
3	0.485375648722841	0.172413797547786
Hexadecimal		
1	3FBF9B71AB5167E1	3FC6122110C13D48
2	3FEA1237688ABA7B	3FEB0A16F5fAA6DC
3	3FDF106506628552	3FC611A7C2EC676C

x_k). The graphical representation of the uncertainty case is shown in Figure 3. Applying the algorithm * from Section 2.1, the following results were obtained: 3 trajectories for 3 different x_0 are considered, but none of these x_0 values is recovered as exactly as previously (see Table 3).

The differences appear now even in decimal values. In hexadecimal representation (which really counts), the inaccuracies are not limited to 4 bits as before. So, in this case, we could not recover x_0.

2.2.2. *The sampling obstacle.* **Sampling the sequence** of the key can be **a more complicated obstacle**. In this case the "key" is created by taking only values separated by n iterations in time. Even for $c = p$, the situation can become really complicated because we do not have a formula for jumping from time index to time index (the mathematical effect of sampling). Section 3 finds a way to mathematically describe these jumps and further elaborates on this topic.

Remarks: In [6], by using the statistical test procedure described in [7], the minimum sampling distance (number of iterations) that enables to generate the i.i.d. data from tent map was determined. For example, for $p = 0.2$ one needs to sample the tent map with a distance $d = 35$ iterations in order to have independent data, while for $p \in [0.4, 0.6]$ the sampling distance is about

15 iterations. Sampling the tent map opens the way to the introspection in the compound tent map.

3. Gray codes, precision in initial condition recovery and the compound tent map

3.1. **The compound tent map.** We want to find a way to mathematically define the compound tent map $f_n(x)$ for any n order. We will see that for all n, even though $f_n(x)$ looks quite complicated, its representation consists only of lines,slopes and intervals. Our task is to find these intervals and slopes and to define the function.

Let us proceed with the second order compound function $f_2(x)$:

$$f_2(x) = f \circ f(x) = \begin{cases} \frac{f(x)}{p} & \text{if } 0 \le f(x) \le p \Longrightarrow \triangle 0 \\ \frac{1-f(x)}{1-p} & \text{if } p < f(x) \le 1 \Longrightarrow \triangle 1 \end{cases} \tag{2}$$

Here $\triangle$ stands for the binary code of the previous n-1 iterations (for n=2, $\triangle$ means the code assigned to $f_1(x)$).

We will code the 4 intervals intervals for $f_2(x)$ with two binary symbols. One can notice that each time we use the first branch from equation (1) we add a zero. For example, if we use the first branch twice and obtain the first entry for $f_2(x)$, we will code this interval with *00*. Similarly, we get the other three combinations, equation (3). What is interesting is that there is a certain pattern in these interval-codes.

$$f_2(x) = \begin{cases} \frac{x}{p^2} & \text{if } 0 \le x \le p^2 \Longrightarrow 00 \\ \frac{p-x}{p(1-p)} & \text{if } p^2 < x \le p \Longrightarrow 01 \\ \frac{x-p}{(1-p)^2} & \text{if } p < x \le 1-p(1-p) \Longrightarrow 11 \\ \frac{1-x}{p(1-p)} & \text{if } 1-p(1-p) < x \le 1 \Longrightarrow 10 \end{cases} \tag{3}$$

A visual representation of the iteration process with $f_2(x)$ is shown in Figure 4. For the moment, the importance of interval coding might not be clear. What we can definitely notice is the way the interval limits are obtained. These interval limits are [0,p,1] for $f_1(x)$, equation (1). When we compute $f_2(x)$ we have five such interval limits. The values of these limits are shown in Table 4.

Table 4

The limits of the intervals for $f_1(x)$ and $f_2(x)$.

Limit	v_1	$\Longrightarrow$	Limit	v_2
L1	0		L1	0
L2	p		L2	p^2
L3	1		L3	p
			L4	$1-p(1-p)$
			L5	1

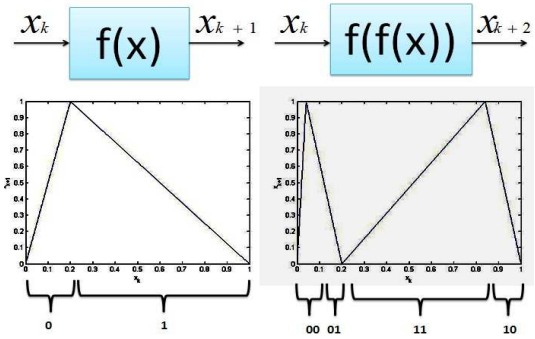


FIG. 4. $f_1(x)$ and $f_2(x)$ and the interval-codes.

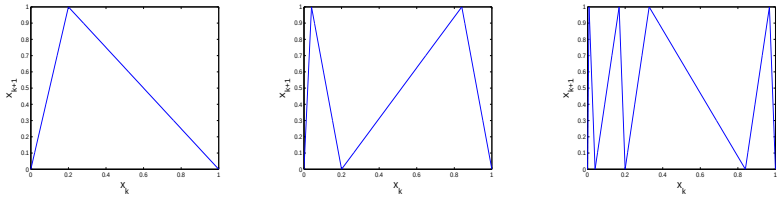
What can be noticed is that the number of these limits is $2^n + 1$, where n is the order of $f_n(x)$. Secondly, p is always in the middle of these limits (its limit-counter is in the middle of $1...2^{n+1}$). Furthermore, we can extend these results to a more general way of creating the limits for $f_n(x)$ using the set of limits for $f_{n-1}(x)$. We explain the algorithm for $f_1(x)$ and $f_2(x)$ and we will see that the algorithm also holds for $f_3(x)$. (1) Let us denote the set of limits corresponding to $f_1(x)$ by $v_1 = [0, p, 1]$.
(2) Compute $v_{11} = p \cdot v_1 \longrightarrow v_{11} = [0, p^2, p]$, as a first intermediary set.
(3) $v_{12} = 1 - (1-p) \cdot v_1 \longrightarrow v_{12} = [1, 1-p+p^2, p]$ is our second set, computed on the basis of v_1.
(4) Next, compute the union of the two intermediary sets: $v_2 = v_{11} \cup v_{12} \longrightarrow v_2 = [0, p^2, p, 1-p+p^2, 1]$. After this procedure we obtain a v_2 set of limits identical to the last column in Table 4.

More generally, we have v_{i-1} corresponding to $f_{i-1}(x)$ and build the intermediary vectors v_{i1} and v_{i2} in the same way as above. By performing the union between these two, we get v_i corresponding to $f_i(x)$. Recursively, we can obtain the set of limits for every $f_n(x)$.

We shall next present the equations for $f_3(x)$. Our assumptions apply here too. This can be extended to any $f_n(x)$.

$$f_3(x) = \begin{cases} \frac{x}{p^3} & \text{if } 0 \le x \le p^3 \Longrightarrow 000 \\ \frac{p^2-x}{p^2(1-p)} & \text{if } p^3 < x \le p^2 \Longrightarrow 001 \\ \frac{x-p^2}{p(1-p)^2} & \text{if } p^2 < x \le p(1-p(1-p)) \Longrightarrow 011 \\ \frac{p-x}{p^2(1-p)} & \text{if } p(1-p(1-p)) < x \le p \Longrightarrow 010 \\ \frac{x-p}{p(1-p)^2} & \text{if } p < x \le p+p(1-p)^2 \Longrightarrow 110 \\ \frac{1-p(1-p)-x}{(1-p)^3} & \text{if } p+p(1-p)^2 < x \le 1-p(1-p) \Longrightarrow 111 \\ \frac{p-p^2-1+x}{p(1-p)^2} & \text{if } 1-p(1-p) < x \le 1-p^2(1-p) \Longrightarrow 101 \\ \frac{1-x}{p^2(1-p)} & \text{if } 1-p^2(1-p) < x \le 1 \Longrightarrow 100 \end{cases} \tag{4}$$

So, for each step we get a big formula for $f_n(x)$, but very simple in a graphical representation: only lines (between 0 and 1 on OY) and bounded by two interval limits on OX, see Table 4 and Figures 4 and 5. We know the limits



(a) (b) (c)

FIG. 5. Graphical representations: (a) $f_1(x)$; (b) $f_2(x)$; (c) $f_3(x)$.

on OX. The next step is to find a way to compute the slopes of those lines. With two points and a slope we have more than enough to graphically describe $f_n(x)$.

This is the moment when the codes come on the scene. We can see that the codes obtained in the $f_n(x)$ formulas of this section are nothing else than the Gray codes associated to the numbers from 0 to 2^n-1 (converted to binary and from binary to Gray codes [8, 9, 10]). Even more important is that the slope on each small interval between 0 and 1 is related to these codes as follows:

(1) The slopes look like: $\dfrac{\pm 1}{p^a \cdot (1-p)^b}$, where:

(i) a = the number of zeros from the code of the interval.

(ii) b = the number of ones from the code of the interval.

(2) The sign of the slopes alternates (positive/negative).

This is a fact that applies to any $f_n(x)$. Now we are in the position when we know the following:

At each step from $f_{n-1}(x)$ to $f_n(x)$ we double the number of intervals.

We know these newly created intervals; *0,p* and *1* will always be among the interval limits.

We also know the slopes for each subinterval $\in [0,1]$.

We can draw our functions $f_n(x)$, for all $n > 1$ (Figure 5).

Here comes an **important remark**. Be the real values denoted by $x_0, x_1, ..., x_m$ obtained by iterating (1) and the corresponding binary values $b_0, b_1, ..., b_m$ for $c = p$ as binarization threshold. Note that $x_m = f_m(x_0)$, and x_0 lies in the interval of $f_m(x)$ that has assigned the Gray code $b_0, b_1, ..., b_{m-1}$.

Let us verify this by an example for $x_0 = 0.123465637517625$ and $p = 0.2$. For $f_1(x)$ we have $x_0 \in [0, p]$, so having a 0 assigned in binary representation. For $f_2(x)$ $x_0 \in (p^2, p]$, so the interval corresponding to $f_2(x)$ is 01.

If we continue in this way, for a certain $f_m(x)$ we can find the interval where x_0 lies. **The code of this interval is the m-binary sequence obtained by successively iterating tent map (1) from x_0 and discretizing with $c = p$. This is the reason why we get x_0 so precisely (as shown in Section 2). In the binary sequence we have all the information we need: the Gray codes**.

3.2. **The mathematical explanation of finding the initial condition from 16 attempts.** We have seen that $f_n(x)$ is nothing else but a combination of 2^n lines between 0 and 1; these lines are bounded on OX between two successive limits of the intervals corresponding to $f_n(x)$ ($v_n(j)$ and $v_n(j+1)$) and have a certain known slope ($slope_n(j)$, where $j \in \{0, \ldots, 2^n - 1\}$).

We can write this as:

$$f_n(x) = \begin{cases} slope_n(j) \cdot (x - v_n(j)) & \text{if } v_n(j) < x \leq v_n(j+1) \text{ and } j \text{ is even} \\ slope_n(j) \cdot (x - v_n(j+1)) & \text{if } v_n(j) < x \leq v_n(j+1) \text{ and } j \text{ is odd} \end{cases} \tag{5}$$

In Section 2.1 that dealt with x_0 recovery, we took the binary sequence of the iterations from *0* to m (binarized with $c = p$; the p control parameter was considered known) and we showed that we can find x_0 very close to the original one from 16 attempts. The x_m value can be obtained by applying the compound tent map to x_0, i.e., $x_m = f_m(x_0)$. When we come back from iteration m to 0 we invert $f_m(x)$ and get x_0 from x_m ("inversion" is not exactly correct; one branch from $f_m(x)$ is inverted).

In this way we can express x_0 recovered as: $x_0 = \dfrac{x_m}{slope_m(j)} + v_m(j)$. Let us have a look at the terms involved in this formula:

(1) $v_m(j)$ is one of the interval limits. So, x_0 is in a vicinity of $v_m(j)$ specified by the Gray code of the interval, i.e., the binary representation of the first m successive values of (1).

(2) x_m is the value that we do not know and we assign in algorithm * when we begin iterating backwards.

(3) $\dfrac{1}{slope_m(j)} = p^a \cdot (1-p)^b$, where a and b are the number of zeros and ones respectively from the m-binary sequence. *The binary sequence is identical to the code associated with the interval from $f_m(x)$ where x_0 lies.*

(4) For $p = 0.2$ and $m = 100$, a value is around $0.2 \cdot m$ and b value is around $0.8 \cdot m$. This is not entirely true (we may have 0.75 or 0.84, instead of 0.8, but not great variations around the mean value). With these values, $p^a \cdot (1-p)^b = 1.85 \cdot 10^{-22}$. The 10^{-22} value is around 2^{-70}, which is no longer noticeable by the computer. Thus, we will find x_0 with a small deviation around one of the interval limits. These limits are very close one to each other for $m = 100$. This is the reason why we can recover so accurately the initial condition when $c = p$.

However, we should take into account that for a very small or very large p value (close to 0 or 1), the intervals will not be so well weighted and we may need more than 100 steps to go back to x_0.

These short notes also support the idea that the sampling of the tent map might be a serious obbstacle when one tries to recover x_0. When we first apply sampling and then binarization, the information expressed by the Gray codes (localizing the initial condition) is no longer available.

4. Conclusions

We have shown that in certain cases the initial condition should not be included in the secret key in cryptography, because it can be found very easily. One solution to make it more difficult is discretizing with $c = 0.5$ and sampling the chaotic map to get i.i.d. data. An additional security in this respect could be obtained by applying a "running key approach" for tent map as it was advanced for logistic map in [11]. This approach was not addressed in this paper.

The last part of the paper concentrates on the compound tent map issue. We found an easy general formula to describe $f_n(x)$. In addition, we showed why one can find x_0 from 16 attempts, a result stated in the literature, but not so convincingly proved. This result is valid for $c = p$, when considering the successive iterations, without sampling the tent map. Furthermore, in this case, we have noticed a subtle connection with the Gray codes. The compound tent map description brings additional support to the idea that sampling the tent map could be a serious obstacle in recovering the initial condition.

There are some difficulties that may appear when someone wants to make use of our formula for the compound map. These are mainly connected to the order of writing the operations that define the compound tent map. However, this new computational signal - the compound tent map - has the same properties as (1) (the same uniform probability law; the minimum statistical independence sampling distance diminishes by the m order of the compound tent map, e.g., for p=0.4 and $f_{15}(x)$ the successive values are practically i.i.d.) and can be a source of new research in this field.

REFERENCES

[1] *D. Arroyo, G. Alvarez, J.M. Amigo and S. Li*, Cryptanalysis of a family of self-synchronizing chaotic stream ciphers, Communications in Nonlinear Science and Numerical Simulations, **16**(2011), pp.805-813.

[2] *G. Alvarez, D. Arroyo and J. Nunez*, Application of Gray code to the cryptanalysis of chaotic cryptosystems, 3rd International IEEE Scientific Conference on Physics and Control, 2007

[3] *D. Arroyo*, Framework for the analysis and design of encryption strategies based on discrete-time chaotic dynamical systems, PhD thesis, 2009.

[4] *A. Ilyas, A. Luca and A. Vlad*, A study on binary sequences generated by tent map having cryptographic view, In Proc.9thInternational conference on Communications (COMM), Bucharest, June 21-23, 2012, pp. 23-26.

[5] *A. Luca, A. Ilyas and A. Vlad*, Generating random binary sequences using tent map, In Proc.10th.International Symposium on signals, Circuits and Systems (ISSCS), Iasi, Romania, June 30-July1, 2011, pp. 81-84

[6] *A. Luca, A. Vlad , B. Badea and M. Frunzete*, A study on statistical independence in the tent map, Proc. IEEE Int. Symposium on Signals, Circuits and Systems (ISSCS), Iasi, Romania, July 9-10, 2009, pp. 481-484

[7] *B. Badea and A. Vlad*, Revealing statistical independence of two experimental data sets: An improvement on Spearmans algorithm, LNCS 3980, 1166-1176.

[8] , Q21: What are Gray codes, and why are they used?, "`www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/part6/faq-doc-1.html`", Accessed online 15.04.2013

[9] *G. Alvarez, M. Romera, G. Pastor and F. Montoya*, Gray codes and 1D quadratic maps , Electronics Letters, **34**(1998), pp. 1304-1306

[10] *X. Wu, H. Hu and B. Zhang*, Parameter estimation only from the symbolic sequences generated by chaos system, Chaos, Solitons and Fractals, **22**(2004), pp. 359-366.

[11] *A. Vlad, A. Ilyas and A. Luca*, Unifying running-key approach and logistic map to generate enciphering sequences, Annals of Telecommunications, **68**(2013), pp. 179-186.