

IMPLEMENTING SYMMETRIC CRYPTOGRAPHY USING CHAOS FUNCTIONS

Dr. Ranjan Bose and Amitabha Banerjee

Electrical Engineering Department,

Indian Institute of Technology, Hauz Khas, New Delhi-110016

rbose@ee.iitd.ernet.in , amitabha_iitd@hotmail.com

ABSTRACT

Chaos functions have mainly used to develop mathematical models of non linear systems. They have attracted the attention of many mathematicians owing to their extremely sensitive nature to initial conditions and their immense applicability to problems of daily life. In this paper we investigate the utility of such functions in symmetric key cryptography for secure communications. An algorithm using one of the simplest chaotic functions $f(x)=4*x*(1-x)$ is proposed. The algorithm is compared with the DES algorithm which has by far been the most widely used standard for symmetric key cryptography. The algorithm is proved to be much more secure than the DES algorithm. However it has a higher computation cost. Practical applications of the algorithm are also suggested.

1. INTRODUCTION

Most of the research work at present on symmetric key cryptography concentrates on block based algorithms in which blocks of messages are subjected to numerous rounds of computation with the help of a single or multiple keys. Such algorithms are mainly based on the DES algorithm which bears a 56-bit key on 64-bit blocks with 16 rounds of key dependent computation[4]. But with ever increasing computation speeds , such algorithms have become more and more susceptible. With current computational resources available a hacker can try each of the 2^{56} keys possible in the DES algorithm in just a matter of 48 hours and thus crack the encrypted message. It is therefore no longer secure to use a single key to encrypt all data. The obvious solution is to use multiple keys , a different key for encrypting each block of data. But this has practical limitations , as all the multiple sets of keys have to maintained at both the sending and the receiving ends. Also the number of multiple keys unless extremely large in number will not pose any challenge to the hacker who we assume has unlimited resources at his disposal. Another solution is to generate one-time pads for encryption with the help of a single key and various chaining algorithms. But since encryption algorithms are publicly known , the above procedure critically depends on the security of the single key.

The idea of using mathematical functions for generating multiple keys or one time pads has been largely unexplored. Some such functions are suggested in [1]. However simple mathematical functions are not sufficient . It is always assumed that the algorithm for encryption is public which means that the function to generate the multiple keys is known to the hacker as well. This means that once the hacker is able to discover one key , he immediately has access to all the other keys. This is where chaotic functions can play a major role. In this paper an algorithm using the chaotic function $f(x)=4*x*(1-x)$ is explored to generate multiple keys for symmetric key cryptography. The encryption step proposed in the algorithm consists of just a simple XOR operation which should be sufficient unless there is a known ciphertext-plaintext attack. Therefore if felt necessary the multiple keys generated can always be applied in the DES algorithm, its various modified versions and other

block based algorithms which are available in the market currently. Also various complex chaos functions like Lorentz equations , Well Oscillator equations and Julia Sets or other fractal equations can be applied to generate multiple keys. A mathematical analysis of the above functions can be found in [2].

This paper is structure as follows. The properties of chaos functions and the function $f(x)=4*x*(1-x)$ in particular is discussed in section II. We propose the algorithm for generating the multiple keys and implementing it in symmetric key cryptography in section III. Section IV evaluates this algorithm. Section V presents the conclusions and the scope for futher work in this area.

2. CHAOS FUNCTIONS

Chaotic functions which were first studied in the 1960's show numerous interesting properties. The iterative values generated from such functions are completely random in nature although limited between bounds. The iterative values are never seen to converge after any value of iterations. However the most fascinating aspect of these functions is their extreme sensitiveness to initial conditions. For example even if the initial start value of iterations is subjected to a disturbance as small as 10^{-100} , iterative values generated after some number of iterations are completely different from each other. It is this extreme sensitivity to the initial conditions that make chaotic functions very important for applications in cryptography.

One of the simplest chaos functions that has been studied in recent times is the function $f(x)=p*x*(1-x)$ which is bounded for the limits $0 < p < 4$. This function can be written in iterative form as $x_{n+1}=p*x_n*(1-x_n)$ with x_0 as the starting value. A thorough treatment and analysis of this function can be found in [3]. In this paper some of the important properties of this function that are of relevance are described.

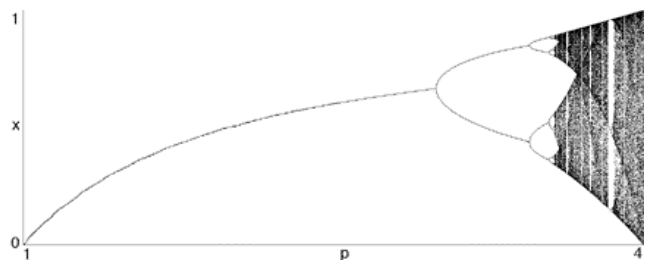


Figure 1: The "bifurcation diagram for the function $f(x)=p*x*(1-x)$

Figure 1 shows the "bifurcation diagram"[3] of this function. This is a plot of the parameter 'p' with the values that are obtained after some number of iterations. For $0 < p < 3$, the function is seen to converge to a particular value after some number of iterations . As p is increased to just greater than 3 the curve splits into 2 branches. The values generated by this function now oscillate between two different values. As the parameter 'p' is further increased , the curves bifurcate again

and now the oscillations are seen in between 4 values. As 'p' is further increase the bifurcations become faster and faster , 8, 16 then 32. Beyond a certain value of 'p' known as the "point of accumulation" periodicity gives way to complete chaos. This is found for $p > 3.57$. The chaos values generated at this point are seen to be restricted to two different bounds. As the value of 'p' is further increased the two bounds give way to a single bound. Also the range between which chaos values are yielded increases constantly as the value of 'p' is increased. Finally for $p=4$, we observe that chaos values are generated in the complete range of 0 to 1. It is this point that we are interested in. Therefore the chaos function that we investigate for applications in generation multiple keys and one time pads for symmetric key encryption in this paper is $4*x*(1-x)$.

As mentioned earlier ,a slight difference in the initial starting value i.e x_0 leads to substantial difference in the obtained iterative values.This is tabulated in Table 1. For an error as small as the order of 10^{-30} we achieve differences of greater than 0.0625 after about 100 iterations. The divergence of obtained chaos values for just a small disturbance in the initial start can be seen much more clearly in Figure 2. The initial start is given a perturbation of the order of 10^{-30} and a difference is observed in iterative values after 100 iterations.

Error in the starting value	Iteration number after which difference in values > 0.0625
1 E -02	5
1 E -05	14
1 E -10	26
1 E -15	42
1 E -19	59
1 E -30	99

Table 1: Difference in achieved iterative values for error in start values

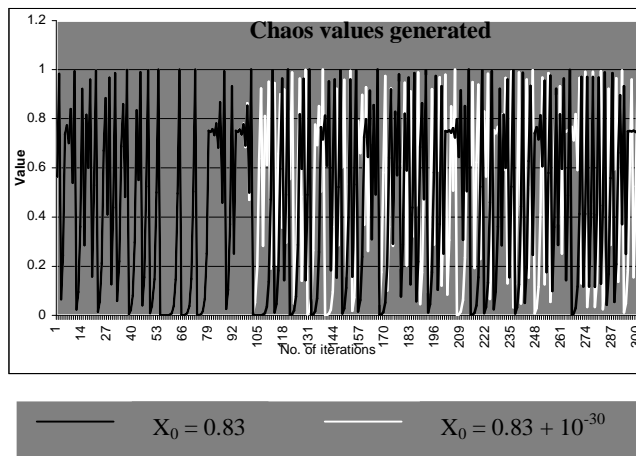


Figure 2: Iterative values obtained for two different starts

Also this function is seen to be so sensitive that two calculating machines which round off fractions after different places of decimal will yield different chaos values after a certain number of iterations. This is shown in Figure 3 where a calculating machine rounding off after 35 places of decimal yields different values than a calculating machine rounding off after 50 places of decimal after about 125 iterations. The complete randomness of chaos functions is shown in Figure 4. The iterative values obtained from iteration number 100 to iteration number 200 with $x_0 = 0.83$ are plotted. As is seen

values lie in the region of 0 to 1 quite uniformly and no two values are same.

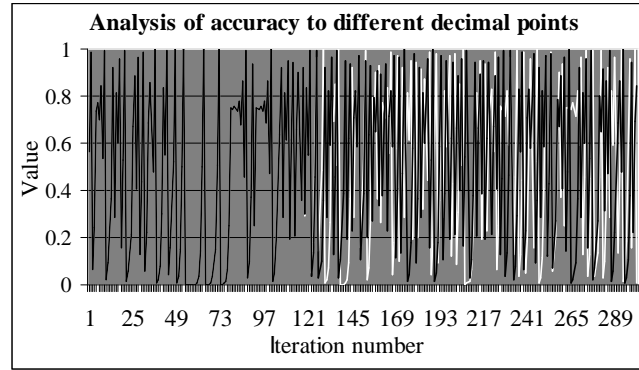


Figure 3: Iterative values generated by calculating machines rounding off after different places of decimal

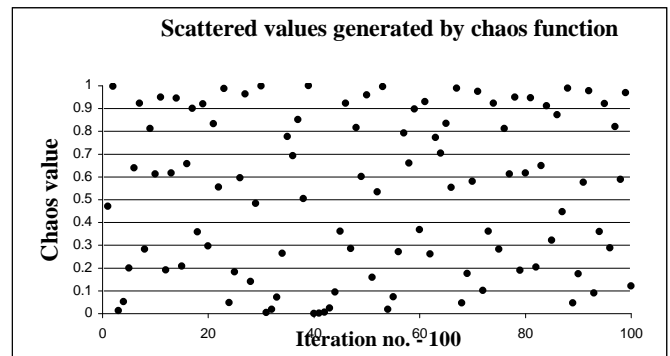


Figure 4: Chaos functions generated by the chaos function $4*x*(1-x)$ between iterations nos. 100 and 200 ($x_0 = 0.83$)

3. GENERATION OF MULTIPLE KEYS AND ONE TIME PADS

For application of the above function for generating multiple keys for symmetric cryptography, it is proposed that the values yielded by the chaos function be converted to binary fractions whose first 64 bits are taken to generate PN sequences. These 64 bit PN sequences can be then used as keys. For this, the following three factors have to be first decided::

- The starting value for the iterations (x_0),
- The number for decimal places of the mantissa that are to be supported by the calculating machine and
- The number of iterations after which the first value can be picked for generating keys.
- The number of iterations to be maintained between two picked values thereafter.

We discuss the following parameters one by one.

3.1 Starting value

The start value (x_0) should be so chosen that the PN sequences finally yielded are remarkably distinct from each other. To find the starting value , an idea of the the degree of similarity between PN sequences that will be finally generated by the

yielded iterative values is needed to be found out. For this a similarity function measure is proposed. Let the two PN sequences (non-return to zero) be represented as $p_1[n]$ and $p_2[n]$ in discrete time domain. The similarity function is defined as:-

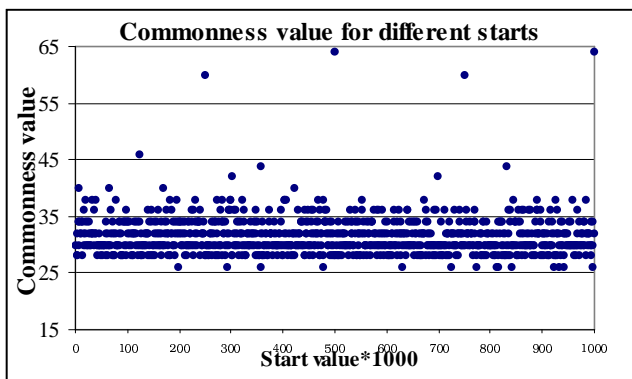
$$S = \text{Max}_{\substack{p=64 \\ p=0}} \sum_{\substack{m=64 \\ m=0}} p_1[m] * p_2[(m+p) \bmod 64]$$

The above similarity function gives us the worst case similarity between two generated PN sequences. It first evaluates the extent to which two given discrete time sequences are similar by multiplying them. Let the result be called ' R_0 '. In case the two sequences are completely similar, ' R_0 ' shall be 64, while if they are orthogonal to each other ' R_0 ' shall be -64. One of the sequences is now rotated around by one bit. (if the sequence was earlier "1010...001" it shall now be "11010...00"). R_1 is now calculated by rotating this sequence with the other sequence. R_2, R_3, \dots, R_{63} are calculated by repeating the above steps 64 times after which the sequence has been rotated around once. The maximum of the values R_0, R_1, \dots, R_{63} is returned as the similarity function as the similarity measure between the two sequences. The above worst case analysis is very important as it is important that the PN sequences generated should be such that even the subparts of the sequences should be as distinct from each other as possible.

The maximum of the similarity values for any two PN-sequence pairs out of a given set of PN sequences is defined as the commonness value ' C '. The measure of ' C ' gives us an idea of how close the generated PN sequences are to each other in worst case analysis.

To find the best possible starting value the following course is adopted. The iterative values generated between the 100th and 200th iterations for a start value are converted into PN sequences and the commonness value is calculated. This is done for start values spaced at 0.001 from each other. The above process should give a rough idea on the degree of randomness of the generated PN sequences. A plot is shown in Figure 5.

Figure 5 ::Commonness values for different starting value for the spaced at differences of 0.001



It can be observed that the commonness value peaks in the case of some starting points (0.123 , 0.250 , 0.358 , 0.5 , 0.75,1.0) These should be avoided as the commonness value peaks at these points. The value of ' C ' for other points are nearly the same, the least value of 26 is obtained for the following points: 0.197 , 0.293 , 0.342 etc.

Any of these points can be used as a possible start value.

3.2 No. of decimal places

This purely depends on the degree of security required for the system. It can be seen from Table 1 that a difference of the order of 10^{-30} in the start value leads to difference in iterative values after 99 iterations. An order of sensitivity of 10^{-30} means that we can have 10^{30} start values between 0 and 1. Each start value can be used as a key to derive the multiple PN sequences from the chaos function. Also a sensitivity of 10^{-30} in the system means that the calculating machine must be supporting upto 30 points of decimal. Therefore the number of decimal places to be supported depends purely on the security that we desire in the system.

3.3 No of iterations

For security in the system we must ensure that two successive values are as different as possible. For this a tolerance margin of 0.0625 is proposed as sequences that are different from each other by at least this amount will have a difference after the third MSB. Therefore before picking up chaos values for generating multiple keys, one must wait for the number of iterations which will ensure that two values yielded are at least separated by 0.0625. For example, from Table 1 we get that for an order of sensitivity of 10^{-30} , one must wait for 99 iterations before starting to pick values.

3.4 No. of iterations of separation

In determining this, we would like to ensure that even if a potential hacker can get an idea of the PN sequence that was employed as a key, he should not be able to calculate the other keys with the knowledge of the chaos function and the number of decimal places supported by the calculating machine.

In the above mentioned algorithm the PN sequence is an approximation of the actual value that was generated by the chaos function to an order of 64 bits. This implies that if a potential hacker is able to get hold of one such key that was used for encryption of the data, he will be able to get the chaos value to an approximation of the order of 10^{-19} ($2^{-64} = 10^{-19}$). Therefore sufficient number of iterations should be allowed to elapse so that a difference of 10^{-19} causes sufficient deviation in the chaos values. From Table 1, we get the value to be 59. Therefore one must wait for 59 iterations before the next chaos value is picked up to be converted into a PN sequence.

As can be concluded from the above, the number of iterations one has to wait for completely depends on the length of the PN sequence that one employs. In case 32 bit coding is chosen instead of 64-bit coding, the number of iterations one has to wait for reduces to 26 which leads to 52 iterations for a 64 bit block. However a 32-bit coding also implies a much lesser range of keys possible for encoding. Therefore the choice of 32 or 64 bit coding will depend on the nature of security required to the amount of computational overhead one is able to bear.

3.5 Algorithm

For an average security requirement the following parameters are suggested for the function $f(x)=4*x*(1-x)$ to develop multiple keys

- (i)Sensitivity of the system: 10^{-30}
- (ii)Start value(key):: 0.197
- (iii)No. of digits to be supported by the calculating machine :: 35 (5 as margin)
- (iv)First iteration to be considered::100th

- (v) No. of iterations of separation::59
- (vi) Bit length of Key::64 bits

Encryption will involve an XOR operation of a generated key with a 64-bit block of data to produce a 64 bit encrypted message. Decryption will involve an XOR operation of the encrypted block with the 64-bit key. Therefore the above mentioned parameters need to be known at both the source and the destination ends.

For supporting upto 35 places of decimal the following data type is proposed according to the IEEE floating point number representation.

Sign bit(1)	Mantissa (105 bits)	Exponent(7)
-------------	---------------------	-------------

The above parameters are just proposals. If one is willing to bear the computational overhead, he can choose the sensitivity of the system to be of the order of 10^{-300} which means that there are 10^{300} different start keys that are possible. But to pick the first iterative value it is needed to wait for about 1000 iterations which will increase the computational burden and the computational time of the system. Thus the sensitivity of the system can be adjusted with the required degree of security at the cost of the computational overhead.

4. EVALUATION

The chief merit of the above algorithm is that the key (starting value) is analog in nature while in other block base encryption algorithms like the DES, the key is a fixed length PN sequence. The multiple keys generated by the algorithm are expected to be completely random and non deterministic in nature. Also as outlined earlier, even if a potential hacker is able to get hold of one key, he doesn't have access to any other key.

To explain the security of the algorithm, a hacker's approach to crack a message encrypted by this algorithm is presented here. Since the 64-bit sequences are completely different from each other, it is computationally infeasible for the hacker to guess each and every key used. The only feasible method is to try out all the different start values. For the algorithm that has been presented, there are 10^{30} start values. Also the number of decimal places supported by the calculating machine is between 30 and 100 and the first value of the start iteration can be anything between 100 and 1000. The number of iterations between any two successive picks also ranges from 60 to typically 500. All these parameters will cause an effective

burden of roughly of the order of 10^7 to the hacker. Therefore the number of tries one must make to crack the algorithm is roughly of the order of 10^{37} . The above is much higher than the corresponding value for the DES algorithm which is 10^{19} .

However the chief demerit of the algorithm is the amount of computational time that one has to spend to encrypt a message. For example in the above algorithm to encrypt every 64 bits, one has to generate 60 iterative values and then convert the value picked into a 64 bit PN sequence. Also a 113 bit data structure to generate the chaos values demands a lot of computational resources. This might make the algorithm unsuitable for any real time applications. However for encryption of very sensitive information where delay might be a secondary factor, the algorithm might prove to be useful. Suggested applications of the above algorithm is in banking systems, financial transactions through the internet and military applications.

5. CONCLUSION

An algorithm using a simple chaos function to generate multiple keys for use in symmetric key cryptography is presented in the paper. The algorithm is shown to provide a high degree of security but requiring lot of computational resources and computational time. Use of other chaos functions like Lorentz Equations, Well Oscillator equations and Julia Sets can be experimented with for symmetric cryptography on the above lines.

6. REFERENCES

- [1] Levi-dit-Vehael and Naceatic *A class of one way functions necessary and sufficient for secure key agreement*, Proceedings, Eurocrypt 96
- [2] Crilly, A.J *Fractals and chaos*, Prentice International Limited, NJ, 1994
- [3] Gleick James *Chaos, Making a new science*, Minerva Limited, 1988
- [4] Ford, Warwick, *Computer communication security*, Prentice Hall, 1994