Design and Implementation of Cryptographic Algorithms using Chaotic Functions

A thesis submitted to the University of Hyderabad in partial fulfillment of the requirements for the award of

Doctor of Philosophy

in

Computer Science

by

Jhansi Rani Prathuri

07MCPC06

School of Computer & Information Sciences

University of Hyderabad

Hyderabad – 500 046, India

July 2013

CERTIFICATE

This is to certify that the thesis entitled **Design and Implementation of Cryptographic Algorithms using Chaotic Functions** submitted by **Jhansi Rani Prathuri** bearing Reg. No. **07MCPC06** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by her under my supervision and guidance.

The thesis has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Dr. S. Durga Bhavani
Supervisor
School of CIS
University of Hyderabad
Hyderabad – 500 046

Prof. A. K. Pujari
Dean
School of CIS
University of Hyderabad
Hyderabad – 500 046

DECLARATION

I, **Jhansi Rani Prathuri**, hereby declare that this thesis entitled **Design and Implementation of Cryptographic Algorithms using Chaotic Functions** submitted by me under the guidance and supervision of **Dr. S. Durga Bhavani** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma.

Date:

Name: Jhansi Rani Prathuri
Reg. No:07MCP06

Signature of the student

Acknowledgments

It is my pleasure to thank all the people who have helped me along the way for completion of Ph.D.

In the first place, I would like to express my thanks to my supervisor Dr. S. Durgra Bhavani for introducing me to the field of chaotic cryptography. I value the interesting discussions we had, which taught me the practical aspects of research and I feel that I am truly privileged to work under her supervision. I am grateful to her for being kind, patient and encouraging me through out the course.

I am extremely grateful to our Dean, School of Computer & Information Sciences, Prof. Arun K. Pujari for providing excellent computing facilities and a nice working atmosphere.

My sincere thanks to Y. V. Subba Rao and Rukma Rekha for their valuable suggestions. I also thank my doctoral committe members Dr. Alok Singh, Prof. K. Narayana Murthy and Prof. C. R. Rao for their guidance through out my course. I thank N. Wilson Naik for his support in many aspects. A special thanks to Dr. P. Anupama and Dr. T. Shoba Rani for their advices and support and giving valuable comments on my drafts.

I thank all my sisters P. Vijaya Rajeswari, P. Raja Kumari, P. Padma Prakashini and P. Parimala Suhasini for their never ending encouragement and prayers. A special thanks to my little loving sister P. Parimal Suhasini who is looking after my one year old son with love and care by sacrificing all her duties. I thank my second brother-in-law B. Suresh Babu for his help and support for progress of my work. I thank all the kids Kasu, Brighty, Grapy, Sam, Charan, Margee and Joy in our family for their love and prayers. I thank my husband Ch. Jaya Rao for his complete support and encouragement in my course.

My heartfelt thanks to my parents who have supported in every aspect of my education and career and for their struggle beyond their potential. I thank my father for his struggle to educate us. I thank my mother for her love, care and encouragement. My love and gratitude to them for everything, everything, everything.

P.Jhansi Rani

ABSTRACT

The thesis focuses on the design and implementation of hash functions and pseudo random bit generators using chaotic functions. Some of the main challenges faced by chaos based cryptography are that the chaotic maps involve real number computations that lead to slow algorithms and cryptanalysis of chaotic maps is theoretically difficult. Even though piece-wise linear maps are fast and facilitate easy analysis, the existing approaches use high dimensional piece-wise linear maps to counter dynamical degradation. We propose that a single one-dimensional non-linear chaotic map can produce ergodic orbits in a fast manner by skillfully manipulating the interplay between chaotic dynamics and shift dynamics. We claim that this approach gives rise to a generic and a transparent design amenable for cryptanalysis.

We design and implement hash functions using these ideas. We propose Baptista hash function that is inspired by Baptista's encryption scheme [15]. This scheme is analyzed and shown to be sensitive to input messages and resistant to collisions, further to possessing good confusion and diffusion capabilities but slow in computation. We propose a keyed hash function, following the elegant design that we envisaged, named Bernoulli keyed hash function. It proves to be an efficient scheme achieving speeds on par with the existing schemes in the literature. Extensive validation is carried out at byte, block and the whole message level for collision resistance and sensitivity to key analysis. Bernoulli hash function is shown to achieve higher confusion and diffusion capabilities than many recent schemes proposed in the literature. Further, since we keep the design simple and transparent, both of these schemes are shown to be amenable for cryptanalysis for preimage resistance and second preimage resistance.

An application of hash functions is carried out by constructing an integrity check value(ICV), which is used as an authentication field in the authentication header of IPSec. ICV is the truncated version of a hash value. Baptista ICV and Bernoulli ICV show strong collision resistance, sensitivity to control parameters and capability for confusion and diffusion.

We propose pseudo random bit generators(PRBG) based on the ideas underlying our hash functions. Randomness of the four proposed PRBG's is evaluated using NIST statistical test suite. All the key streams generated using the proposed PRBG's achieve a p-value ≥ 0.01 which indicates that these sequences can be considered to be random. As an application, we construct stream ciphers using the key streams generated by these PRBG's. We show that one of the stream ciphers proves to be a strong cipher by exhibiting near uniform distribution of ciphertext characters. Further, all the four stream ciphers are validated to possess good avalanche effect by extensive sensitivity analysis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Discovery of the phenomenon of chaos created a huge paradigm shift in sciences and its story is well-documented by James Gleick[27]. Since 1990s, many researchers have found that there exist strong connections between the properties of chaos and the requirements of cryptography. Shannon envisaged a role for chaotic maps due to their mixing (transitivity) property years ago [80]. The intermixing of order and chaos within the attractor of a chaotic dynamical system makes the important characteristic like sensitivity to initial conditions to produce randomness of dynamical behaviour. The property of sensitivity to initial conditions induces diffusion and the mixing property creates confusion, thus making the chaotic maps prime candidates for many cryptographic applications. The literature on chaos based cryptography is abound with proposals for Pseudo Random Number Generator (PRNG), the one-way hash functions, symmetric encryption, public encryption etc.

Recent studies reveal that several well-known methods such as MD5, SHA1 and RIPEMD of classical cryptography too are not immune to collisions [87, 89]. Since chaotic dynamics provides plenty of deterministic nonlinear systems that exhibit unpredictable and random-like dynamical behaviour, chaos based cryptography has been gaining ground as another potential avenue to look for secure hash functions, PRNG's etc. [83, 84, 98, 101, 103, 108]. L.Kocarev presents an excellent overview on chaos based cryptography, and more recently has edited a compilation of work in this field along with S. Lian [41, 44].

1.1 Motivation

Two important aspects in which chaos based cryptography differs from classical cryptography is that a) real number computations are involved instead of integer arithmetic which makes the chaotic algorithms much slower than their counter parts in conventional cryptography; b)cryptanlaysis of chaotic maps is theoretically difficult. In order to meet the challenges of speed and analysis, the chaos community has been making use of piece-wise linear chaotic maps rather than the nonlinear chaotic maps. However due to the finite precision of hardware, the computations of linear maps dynamically degrade much faster than the nonlinear maps. Hence most of the recent approaches use a network of chaotic maps or additional mechanisms for perturbation of the parameters to sustain ergodicity which lead to complex designs difficult for cryptanalysis. The main aim of this work is to evaluate if a simple non-linear map like logistic map can be used to design secure hash functions and pseudo random number generators without compromising on efficiency and security. Since cryptanalysis for nonlinear maps is difficult the design has to be kept simple and transparent, amenable for analysis. We identify three important aspects to focus on during the design process.

1. Since the chaotic functions generate real numbers, the bit generation schemes have to be looked at more carefully instead of following the conventional approaches.
2. To ensure that the underlying design is transparent since cryptanalysis of chaos-based schemes is difficult [5, 35].
3. Of course, the speeds of the algorithms have to be kept on par or even better than the existing schemes.

To the best of our knowledge, this kind of design and analysis has not been carried out before.

1.2 Thesis Contributions and Chapter Organization

Our work mainly focuses on studying the design and implementation of hash functions in the domain of chaos based cryptography. The secure hash functions that we design lead to proposals for pseudo random number generators which can be in turn utilized to produce stream ciphers. We propose that by using one dimensional(1-D) nonlinear chaotic maps along with the Bernoulli shift map in an appropriate fashion, one can achieve faster algorithms while retaining as good a confusion and diffusion as the schemes proposed in the literature.

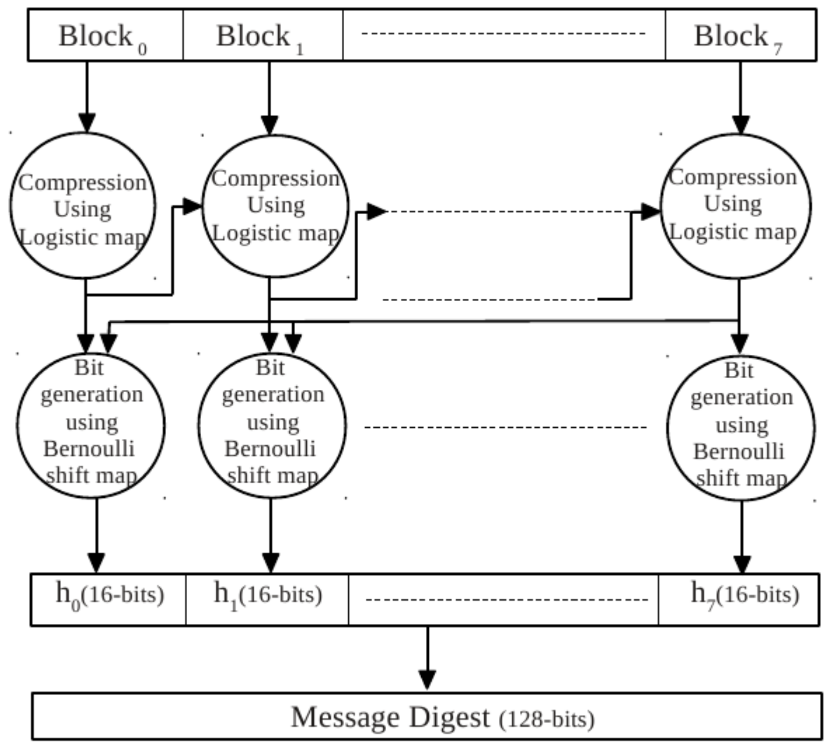
Chapter 2 starts by giving the mathematical definition of a chaotic map and discusses the details for a few of the classical one-dimensional chaotic maps like the logistic map, the tent map, Bernoulli saw-tooth transformation etc. These maps are discussed in the context of the important challenges faced by the chaotic maps in cryptography namely, dynamical degradation and bit representation. Further, the chapter presents a survey of the hashing schemes and PRNG's that have been proposed in the literature. We present a few insights regarding the two mechanisms of a) intermittent perturbation to avoid dynamical degradation and b) the Bernoulli shift map for bit generation that have been useful in designing our algorithms.

Chapter 3 begins with initial work of the thesis by presenting design and analysis of a hash function using Baptista's scheme[15]. We utilize Baptista's algorithm skillfully to develop a novel one-way chaotic hash function which outputs a 128 bit message digest and call it *Baptista hash function.* A pseudo-random number generator using chaotic tent map is incorporated within the hash algorithm to strengthen the hash function. Most of the approaches use an additional machinery like chaotic neural network or a multitude of chaotic maps [92, 103, 108] and we show that using two chaotic maps judiciously achieves a secure hash function. The performance of the hash function is on par with recent chaotic hash functions proposed in the literature without requiring a network of chaotic maps as in [103]. In the space of 1-bit neighbourhood of a given message, the hash function is shown to exhibit diffusion effect with average hamming distance between message digests obtained as 63.56-bits which is close to the ideal of 50%. The collision performance compares favourably with that of chaotic hash functions proposed in the recent literature. Further, Baptista hash function is analyzed theoretically to determine the security properties of preimage and second preimage resistance. Initial version of this proposed hash function has been published in *Proceedings of CNSA-2011.*

A limitation of Baptista hash function is with regard to the bit generation scheme since it uses integers for producing hash value. Moreover, due to the many safe-guards that have been put in place in order to avert preimage resistance, like the chaotic PRNG, the scheme becomes time consuming and hence is not satisfactory. Hence we propose, what we feel is our signature hashing scheme, in **Chapter 4** which reflects all the wisdom gained in the process.

Chapter 4 of the thesis provides a more efficient scheme for a chaotic keyed hash function. The proposed scheme basically compresses each block of data into a real number using a chaotic function to which then, Bernoulli shift map is applied to generate bits which form the message digest and hence we name it *Bernoulli hash function.* We claim that one could plug in a non-linear chaotic function of author's choice in the compression block and generate a new hash function. The design of this function is given in Figure 4.1. The intermediate computations are made faster along with tightening of the bit generation step to ensure security properties of hash function. Through an extensive experimentation and cryptanalysis, the algorithm is shown to achieve excellent performance in collision resistance and preimage resistance which is better than the schemes proposed in the literature.

The results of this proposed keyed hash function have been communicated to a journal.



Block 0
Block 1
Block 7
Compression Using Logistic map
Bit generation using Bernoulli shift map
h0(16-bits)
h1(16-bits)
h7(16-bits)
Message Digest (128-bits)

Figure 1.1: Generic block diagram of Bernoulli keyed hash function

Chapter 5 of the thesis is an application of hash functions that have been proposed in Chapters 3 and 4. Integrity check value is used as an authentication data field in both IPSec authentication header and encapsulating security payload, which is usually generated by MAC algorithms [64], [65]. Hash functions proposed in the literature can be theoretically used for generating integrity check value. Chaotic hash function algorithms have not been explicitly evaluated for this purpose in the literature. We apply our hash functions proposed to construct integrity check values and compare their performance with MD5 and SHA1 by considering the first 96 bits of the hash value which is a standard procedure. Since truncating the length of the hash value may lead to collisions, we conduct a systematic analysis to prove their strength. We assess how the truncation of hash value of length 128 to 96 bits influences its performance without lowering security and report our results. This work is published in *Proceedings of ACC-2011*.

Pseudo random bit generators have been found to be crucial in defining initial vectors for hash functions, secret keys for symmetric encryption schemes and as one time pads in cryptography. **Chapter 6** of the thesis presents designing of pseudo random bit generators. The difficulty of practical realization of a few existing schemes in the literature that need to satisfy many pre-conditions is presented using experimentation with different chaotic maps. We propose four pseudo random bit generators based on our main theme that of using the ergodic orbit of logistic map and sampling with Bernoulli shift map. We test the randomness and strength using NIST statistical test suite and show that these are suitable for cryptographic applications.

Chapter 7 presents an application of the proposed pseudo random number generators for constructing stream ciphers. The four PRBG's are analyzed for their capability for producing good stream ciphers in two ways: by performing fitness test on the distributions of the ciphertext derived using these keystreams with the uniform distribution. One of the stream ciphers emerges to be a strong cipher by exhibiting near uniform distribution of ciphertext characters. Avalanche effect is checked in the cipher text at various points of generation of the cipher as the keys are varied infinitesimally. It is observed that at any point, 50% of change is found for all the four stream ciphers which indicates good avalanche effect.

We conclude the thesis in **Chapter 8** and give future scope of the work. To summarize, the main approach followed in this work has been to generate a "truly" ergodic orbit of logistic map which is sampled using Bernoulli shift map in order to produce secure hash functions and PRBG's which are then applied to generate integrity check values and stream ciphers.

CHAPTER 2

Background and Related Literature

During the last three decades chaotic dynamics has been playing a major role in the field of nonlinear sciences. In 1961, Edward Lorenz while using his basic computer for weather prediction, discovered that small changes in the initial conditions produce large changes in the long-term behaviour which is famously referred to as the 'butterfly effect' [20, 40]. Many of the properties of chaotic systems have their counterparts in conventional cryptosystems. Kocarev in his seminal paper reviewing the field of chaos based cryptography[41] tabulates the differences and similarities as in Table 2.1 between conventional and chaotic cryptography[2, 41].

Cryptographic Algorithms	**Chaotic Systems**
Phase space: Finite set of integers	Phase space: (sub)set of real numbers
Based on mod function	Based on chaotic maps
Rounds	Iterations
Key	Parameters
Diffusion	Sensitivity to initial condition(s)
Confusion	Ergodicity
Integer arithmetic	Floating point arithmetic
Computationally fast	Computationally slow
Based on prime numbers	Does not require prime numbers

Table 2.1: Conventional vs Chaotic cryptography.

2.1 Preliminaries of Chaotic Maps

We consider discrete chaotic dynamical systems throughout the thesis. Let X be a metric space. A map $f : X \to X$ is said to be chaotic according to Devaney if f satisfies the the following three properties[23, 38]:

- sensitivity to initial conditions
- topological mixing property
- dense periodic orbits

We only consider one dimensional chaotic maps in this thesis, for which $X = [0, 1]$. For $x \in X$, $(x, f(x), f^2(x) \ldots)$ is called the orbit of f under x, where $f^i(x) = f(f^{i-1}(x)), i \geq 2$. A few important one dimensional(1D) chaotic maps that are considered in this thesis are logistic map, tent map and Bernoulli saw-tooth transformation whose details are given below.

2.1.1 Logistic map

This is an important family of uni-modal maps $f(x) = \lambda x(1-x),\ x \in [0, 1],\ \lambda > 0$ is a family of logistic maps which are proved to be chaotic for $\lambda \geq 4$.

2.1.2 Tent map

$$T(x) = \begin{cases} t * x & \text{if } 0 \leq x \leq \frac{1}{2} \\ t * (1 - x) & \text{if } \frac{1}{2} \leq x \leq 1 \end{cases}$$

for $t > 0$, $T(x)$ denotes a family of tent maps which attains full chaos for $t \geq 2$.

The tent map $T(x)$ and logistic map $f(x)$ are closely related, mathematically termed as topologically conjugate to one another by a nonlinear transformation $h(x) = sin^2(\frac{\pi x}{2})$, such that $f(h(x) = h(T(x))$, $x \in [0, 1]$

2.1.3 Bernoulli map

Bernoulli map is defined as $S(x) = 2x\ mod\ 1,\ x \in [0, 1), S(1) = 1$. The mapping is also referred to as saw-tooth transformation because of its visual appearance as seen in Figure2.1.

The mapping is also called a shift map because if the computations are in binary form, say $x_0 = 0.1010110$ then multiplying by 2 simply would mean shifting all the bits over to the left such that we get $\sigma(x_0) = 1.010110\ldots$ and then dropping the integer part.

The tent map and the Bernoulli map are closely related by $T^2(x) = T(S(x))$.

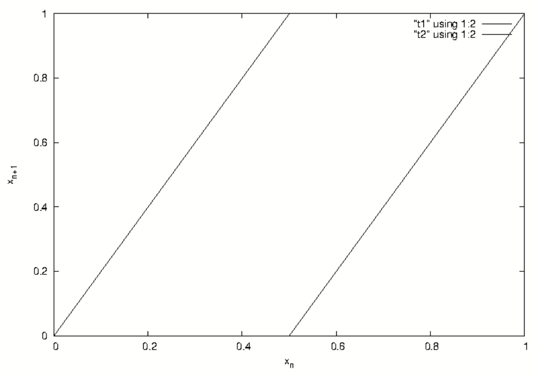


Figure 2.1: Bernoulli map

2.1.4 The shift dynamical system

$\Sigma = \{a = (a_0, a_1, \ldots a_i \ldots) : a_i = 0 \textit{ or } 1, i \geq 0\}$ denoting all the possible infinite bit sequences is called the code space. The shift map is defined as $\sigma : \Sigma \rightarrow \Sigma$ with

$$\sigma(a_0, a_1, \ldots a_i \ldots) = (a_1, a_2 \ldots a_i \ldots)$$

σ basically shifts the bit sequence to the left by one bit. (Σ, σ), the shift dynamical system is an important example of chaotic dynamical system. Theoretically it is very easy to prove that σ is a chaotic map[74]. This map is shown to display all the properties of chaos. For example $(a_1, a_2 \ldots a_i, a_1, a_2 \ldots a_i \ldots)$ are periodic points under the shift map. It can be seen that σ is sensitive at $a \in \Sigma$: choose a to be very close to b, with their first k significant bits identical. Suppose $a_i \neq b_i$ for all $i > k$, then under dynamical iteration of the shift map, the points $\sigma^{k+1}(a)$ and $\sigma^{k+1}(b)$ differ in all their bits. Hence two points which are close in the code space Σ, due to the shifting of bits can be made to diverge arbitrarily by choosing the least significant bits as complements of one another. The shift map is one of the most beautiful prototypes of chaos and is shown to be topologically equivalent to the any Devaney chaotic map[23].

2.1.5 Sensitivity to initial conditions

f is sensitive at a point x if there exist points y arbitrarily close to x, such that if we track the evolution of the orbits of x and y under f, termed as $f^n(x)$ and $f^n(y)$ then $d(f^n(x), f^n(y)) > \delta(> 0)$. If f is sensitive to initial conditions, the trajectories of the dynamical system defined by f change drastically for points y that are initially very close to x. Thus, an arbitrarily small perturbation of the current trajectory may lead to significantly different future behaviour. Sensitivity to initial conditions is popularly known as the "butterfly effect": the flapping of wings represents a small change in the initial condition of the system, which may cause a chain of events leading to large-scale phenomena.

The following Figure 2.2 shows sensitivity to initial condition for logistic map with the initial condition $x_0 = 0.345671$ being perturbed by a very small value. After about 40 iterations, it can be seen that the orbits start diverging. Figure

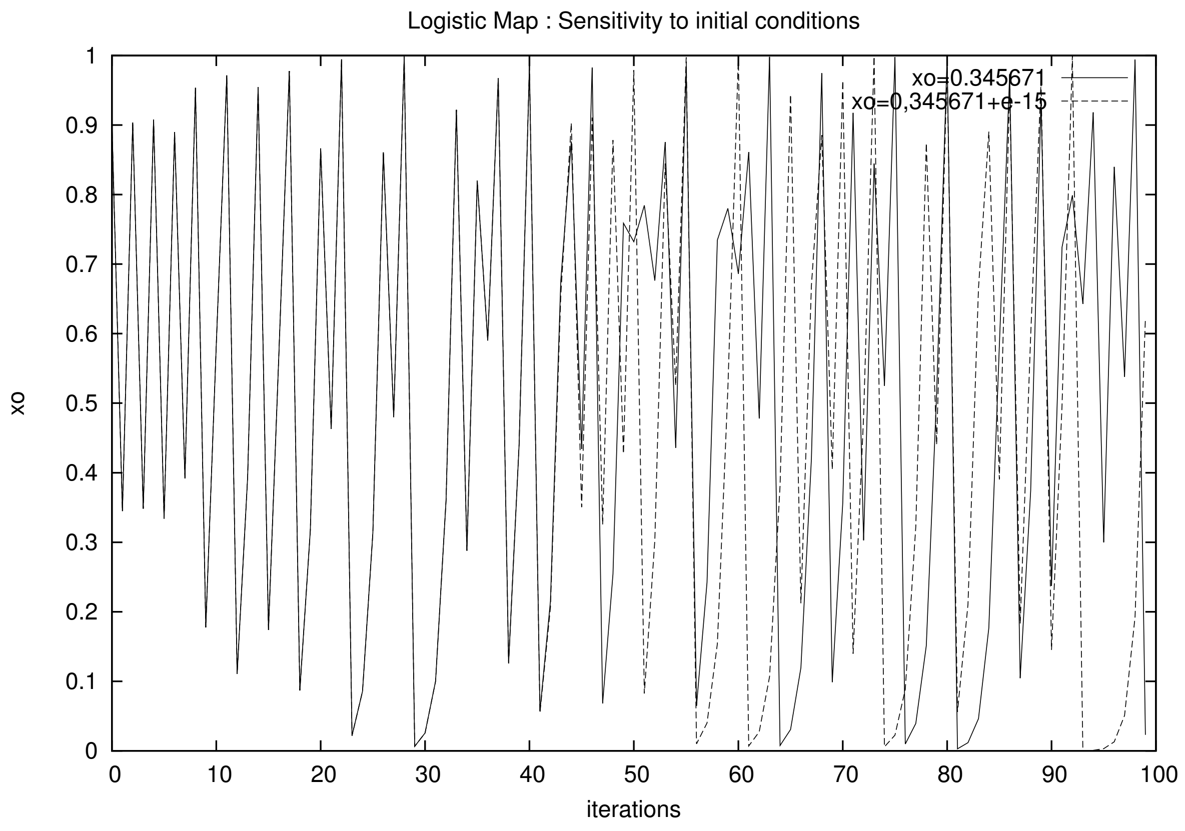


Figure 2.2: Sensitivity of logistic function $f(x) = \lambda * x * (1 - x)$

2.3 shows sensitivity to initial condition for tent map with the initial condition $x_0 = 0.23$ being perturbed by a very small value. Figure 2.4 shows how orbits of points which are very close initially diverge significantly demonstrating the property of sensitivity to initial conditions of Bernoulli map at $x_0 = 0.34$.

2.1.6 Topological mixing property / Ergodicity

A chaotic map is said to have topological mixing property if any two intervals U and V in the domain under dynamics of the map intersect at some iteration. If there exists a point whose orbit comes arbitrarily close to all the points of the attractor, such an orbit is called an ergodic orbit. A chaotic map with ergodic orbit is trivially topologically mixing. It is not easy to discover an ergodic orbit for a nonlinear map. An orbit of the logistic map can be termed as ergodic if

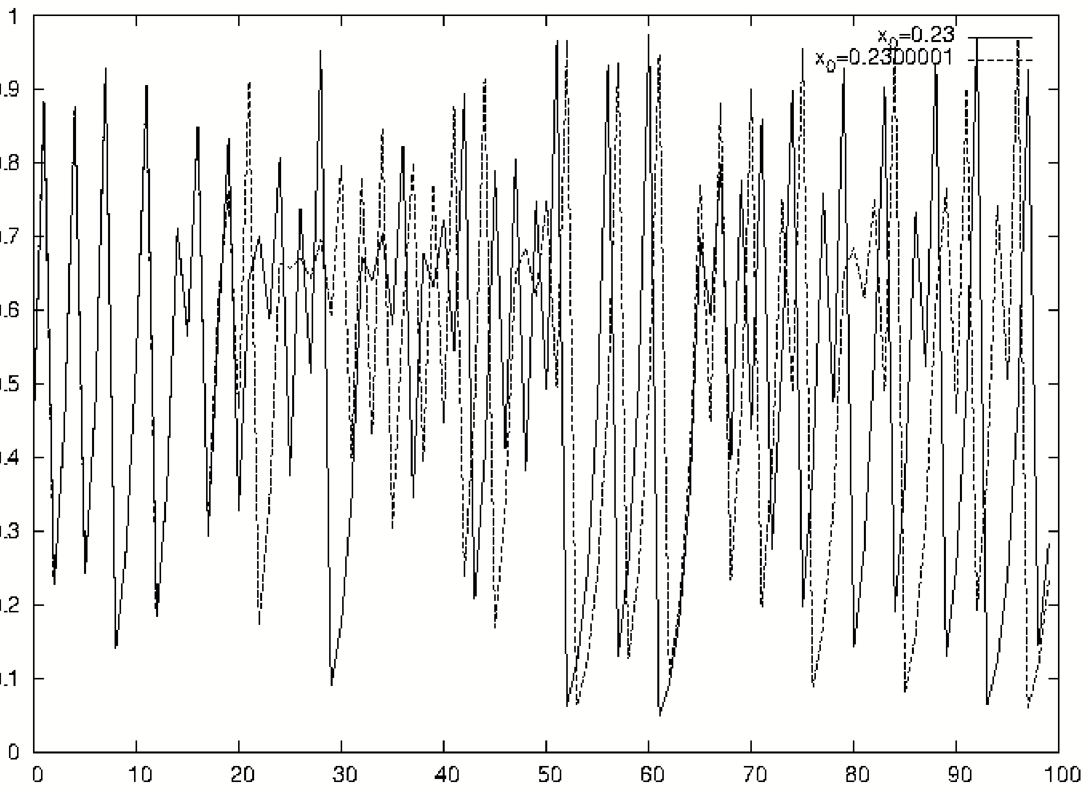


Figure 2.3: Sensitivity of tent map

it approximates the underlying invariant density function $\frac{1}{\pi\sqrt{x(1-x)}}$ of the logistic map.

2.1.7 Dense periodic orbits

The set of periodic orbits is dense in the attractor for a chaotic map. i.e., every point in the space is approached arbitrarily closely by periodic orbits of a chaotic map [94]. Figure 2.5 shows the popular bifurcation diagram for the family of logistic maps, in which, as the parameter value λ is increased from 2 till 3, it can be seen that the orbits settle in one fixed point and when λ is slightly greater than 3 the orbit bifurcates into 2-periodic orbits and then on very quickly to a periodic orbit of period 4 and so on which is the popular period doubling route to chaos, showing complete chaos at $\lambda = 4$.

The important characteristics of the chaotic maps like ergodicity of dynamical behaviour, sensitivity to initial conditions and possession of positive Lyapunov exponents make these prime candidates for many cryptographic applications.

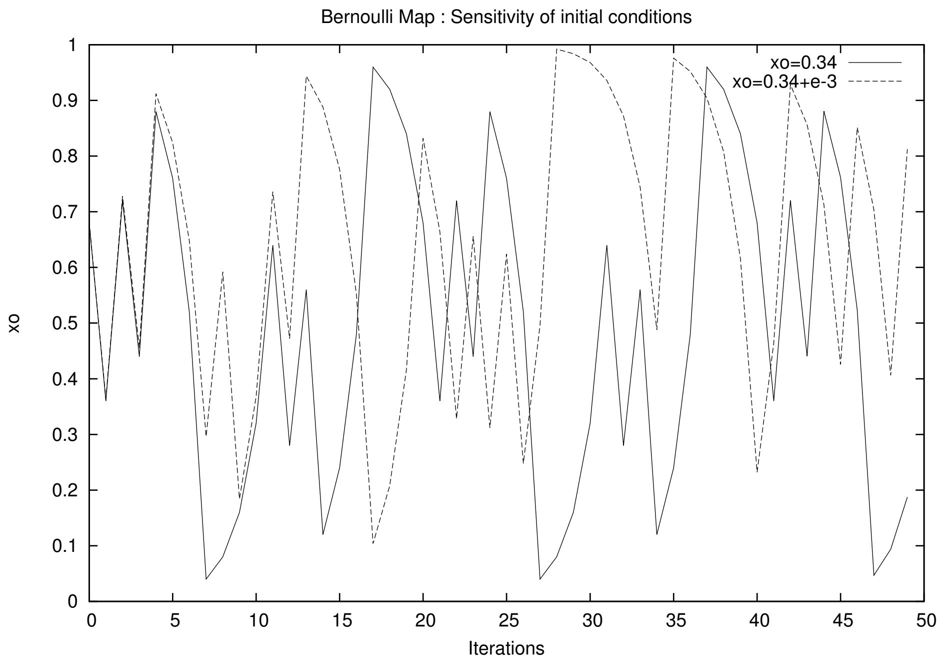


Figure 2.4: Sensitivity of Bernoulli map $\sigma(x) = 2x\ mod 1$

2.2 Chaos Based Cryptography

Robert A. J. Matthews in 1989 proposed a stream cipher using logistic map[66]. This attracted much attention from researchers [3, 25, 29, 43, 55] into this field. M.S. Baptista's symmetric encryption scheme in 1998 spurred new interest in chaotic cryptography among researchers. K.W.Wong and his group proposed improved encryption schemes based on Baptista's algorithm such that the cipher text exhibits uniform distribution [97, 99]. Even though Alvarez et al.[5] showed that Baptista's algorithm is vulnerable to attacks, the idea behind the algorithm led to many chaos based cryptographic schemes.

2.2.1 Hashing schemes

Wong proposed a hashing scheme by adopting dynamic look up table along with Baptista's algorithm in 2003 [98]. Many of the schemes so far used logistic map to generate the chaotic orbit. Many researchers have been making a strong case for the chaotic piece-wise linear maps to be utilized for efficiency purposes in the place of non-linear maps [39, 55, 57].

X.Yi [106] in 2005 proposed a hash function based on chaotic tent maps which is claimed to be better than Wong's scheme in its computational complexity. Jacques and Guyeux[14] present the link between the notions of Devaney's topological

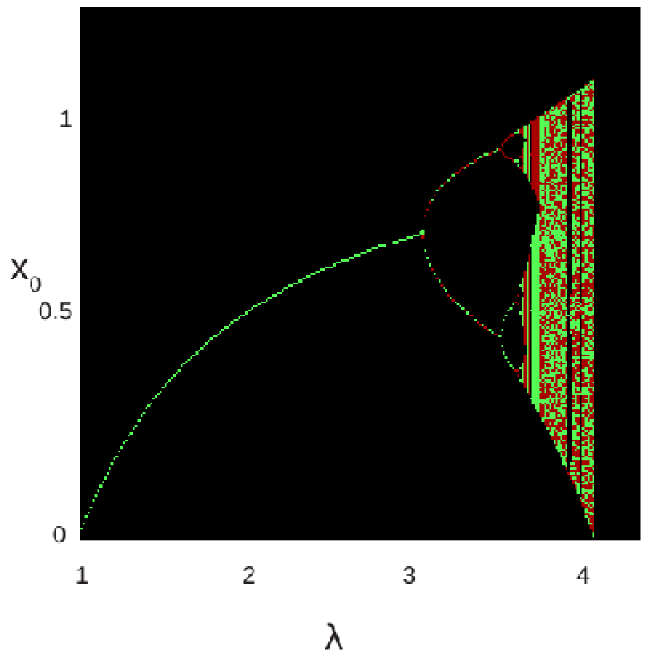


Figure 2.5: Bifurcation diagram of logistic function $f(x) = \lambda * x * (1 - x)$

chaos and discrete chaotic iterations and use this concept in developing a hash function. In 2005 Di Xiao [102] proposed one-way hash function using cipher block chaining method with changeable parameter of piece-wise linear map. It was clear that one piece-wise linear map will not suffice and one has to adopt some higher level architecture in order to induce diffusion and confusion while generating the output.

In 2006, Lian et al.[62] used neural networks to create diffusion and a piecewise linear map to generate hash values of different lengths of 128, 256 and 512 bits. Shihong et al.[86] in 2007 proposed a hash function using nonlinear S-boxes and coupled chaotic dynamics to generate a 160-bits hash value. More recently H.Yang et al [103] have published another hash function based on a chaotic map network and Q.Yang et al [105] have published a hash function based on cell neural network. Jun Peng et al.[75] in 2008 used two chaotic systems and a double-pipe structure to design an iterated hash function. In 2008 Long Min et al. [67] used feed-forward mechanism and three different chaotic systems generated from the unified chaotic system to generate 160-bits hash value. Akhavan et al.[1] in 2009 proposed a hash function based on multi-threaded programming. In 2011, Hai Yu[107] constructed a hashing scheme based on chaotic coupled map network(CCMN) which uses input message as coupling coefficient for CCMN and control parameter. Li Gao et al.[26] proposed a hash function based on Tandem-DM[49] structure which uses discretized chaotic map network.

All of these schemes may be considered as keyed hash functions because for a chaotic map, in general, the initial condition and the parameter of the function family have to be chosen apriori which can be treated as secret keys. In the literature, the analysis for keyed hash functions is carried out more extensively.

2.2.2 Keyed hashing schemes

Zhang et al.[108] are the first to propose a chaotic keyed hash function in the literature. This scheme involves an n-dimensional chaotic dynamical system that uses feed forward- feedback nonlinear digital filter. They use chaotic shift keying along with cipher block chaining mode. Xiao et al.[101] proposed a parallellizable hash computation mechanism. They use a chaotic neural network for compression scheme. The scheme of Xiao et al. is cryptanalysed by [30] and the scheme[101] is shown to be prone to forgery attacks and the keys used are found to be weak. X-Y Wang et al.[91] analyzed the scheme proposed by Xiao et al. and demonstrated the weak keys and forgery attacks. A more secure scheme was proposed by Haung et al. [109] that uses nonce numbers as secret keys. Jacques et al.[13] proposed a keyed hash function in 2011 using the iterated hash function approach. Da Li et al.[51] proposed a keyed hash function based on the modified coupled chaotic map lattice using the nearest and long distance couplings. In 2012 Yantao Li et al.[60] proposed a keyed hash function based on a dynamic lookup table of functions. In 2013 A Kanso et al.[36] proposed chaotic keyed hash function based on a single 4-dimensional chaotic cat map which they claim is efficient and and that it can generate a hash value of different lengths like 160, 256, 512, and 1024 bits.

There is immense literature on pseudo random number generators(PRNG) using chaotic functions. PRNG's classically use system parameters in order to induce randomness.

2.2.3 Chaos based PRNG

According to the literature on pseudo-random number generators, Oishi et al. were the first to propose a PRNG using chaotic maps [68] in 1982. Many schemes develop pseudo random number generators using discrete chaotic maps like logistic map [8, 28, 47, 63]. Argenti et al.[9] in an earlier work, propose a stream cipher based on nonlinear dynamical system called Henon map to generate a chaotic signal. For generation of bits, most schemes use intrinsic quantization of the chaotic map. PRNG's are used for encryption to mask the information bits.

In 2001 Stojanovski and Kocarev [83] mathematically analyzed the application of a chaotic piecewise-linear maps as random number generator. In an adjoining paper [84], they lay out the optimum parameters that can be used for generating a random number with lowest redundancy and maximum margin and show that the output bit rate is 1 Mbits/s. Shujun Li et al. [55] perform theoretical analysis on piecewise linear maps and proposed a pseudo random bit generator based on a couple of chaotic systems. In 2003 Kocarev et al.[42] proposed a pseudorandom bit generator using chaotic map and discussed the possibilities of using different chaotic maps for pseudorandom bit generators. Huaping et al.[34] in 2004 proposed a pseudorandom number generator using a one-way coupled chaotic map lattice and showed that it possesses good statistical properties, long periodicity and is fast in generating PRNGs. Li et al.[58] proposed a pseudorandom number generator using piecewise linear maps. In later works, attention is paid to physical realization of the generators using circuit parameters. Wang K et al.[93] proposed a random number generator and realizing it using an analog circuit. In 2006, Wang L et al.[85] proposed a pseudorandom number generator based on z-logistic map. In 2009 Hu Y et al.[31] proposed a true random number generator using computer mouse movement and also gives the chaos based approaches to remove the patterns with similar mouse movements. In 2010 Narendra K Pareek et al.[72] used two chaotic systems which are cross-coupled with each other to generate a pseudorandom bit generator. In 2012 Liu et al[104] proposed an equation and proves it chaotic on imaginary axis. This is used to propose a pseudorandom number generator using a complex number chaotic equation.

It is clear that to propose a secure hash function or a PRNG one has to finely balance the security degradation that is possible due to finite precision arithmetic and the computational complexity of algorithm. The trend in the literature has been to achieve efficiency retaining the security of scheme.

In this thesis we propose to use a simple nonlinear chaotic map namely logistic map along with Bernoulli shift map in order to achieve ergodicity at the same time keeping it computationally inexpensive.

2.2.4 Public key cryptography

There has been very little work in public key cryptography in chaos literature. Ranjan Bose and Kocarev have proposed public key encryption algorithms based on chaotic functions [18, 45].

2.3 Challenges for Chaos Based Cryptography

Some of the main challenges for designing chaos based cryptographic functions according to us, are to generate a non trivial ergodic orbit from a chaotic map and to evolve a bit generation scheme that avoids dynamical degradation.

2.3.1 Bit generation schemes

There exist various methods in literature for generating bits.

- Conversion to binary representation is popular and has been used in the literature[59, 101, 102, 106]. But due to finite precision of the machines, it is not possible to generate non trivial binary representations of length say 80-bits for a 160-bits hash value from a real number. And for integers, it is certainly not possible. Our implementations showed that using double precision we can generate upto 50 non-trivial bits for real numbers and it depends on the word length of the hardware in the case of integers.

- Threshold mechanism uses the intrinsic chaotic binary sequence in the map. For example, since the logistic map has its critical point at 1/2, generally a binary sequence that imitates the underlying dynamics of the map can be obtained by labeling the k-th iterate as 1 if $f^k(x) \geq 1/2$ and otherwise 0. This method of building binary sequence is called a chaotic binary sequence which has been used in the literature extensively [37, 61, 75, 100].

- Quantization is a more general method in which certain subintervals can be labeled as 1 and the others 0 as used in [108].

2.3.2 Dynamical degradation

Shujun Li in his Ph.D. thesis, writes extensively upon the aspect of dynamical degradation for piece-wise linear maps and proposes dynamical indicators. Similar work is necessary to be carried out for non-linear maps. Experimentally we can visualize that the behaviour of non-linear maps like logistic map will be extremely different from its topological conjugate the tent map. We tabulate the orbits obtained for logistic map and tent map at $x = 0.345$ in Table 2.2. It can be seen that tent map which is a piece-wise linear map degrades by 53 iterations.

Our approach

Shinbrot et al. in their seminal paper in *Nature* observe that *"The extreme sensitivity of chaotic systems to tiny perturbations (the butterfly effect) can be used both*

Iteration	$f^n(x)$	
Number	**Logistic map**	**Tent map**
16	0.934188393110789	0.0800000000017462
32	0.0620208685259484	0.880000114440918
48	0.935975785098273	0.3125
50	0.728976596303414	0.75
52	0.662952702674135	1
53	0.89378566676478	0
64	0.922690983181875	0

Table 2.2: A few points on the orbits of logistic map and tent map for x=0.345.

to stabilize regular dynamic behaviours and to direct chaotic trajectories rapidly to a desired state.". We follow this guiding principle in producing ergodic orbits by **intermittent perturbation** so that *dynamical degradation* is avoided and the seed travels along an ergodic orbit of a chaotic map. Hence to avoid dynamical degradation and to capture the ergodicity of a chaotic map, not so small intermittent perturbation is given to the point intermittently as has been suggested in the classical chaos literature[70].

We show that by using higher precision and perturbation of the system variables, we can avoid multiple chaotic systems and achieve an ergodic orbit using a single non-linear map. We generate only a small number of bits in each round to avoid dynamical degradation and use intermittent perturbation and still maintain efficiency by using the 1-dimensional chaotic maps.

The plot in Figure.2.6 shows the distribution of the points generated for logistic map using these ideas and one can see how it can lead to a fine approximation to the underlying invariant density function $\frac{1}{\pi\sqrt{x(1-x)}}$ of the logistic map.

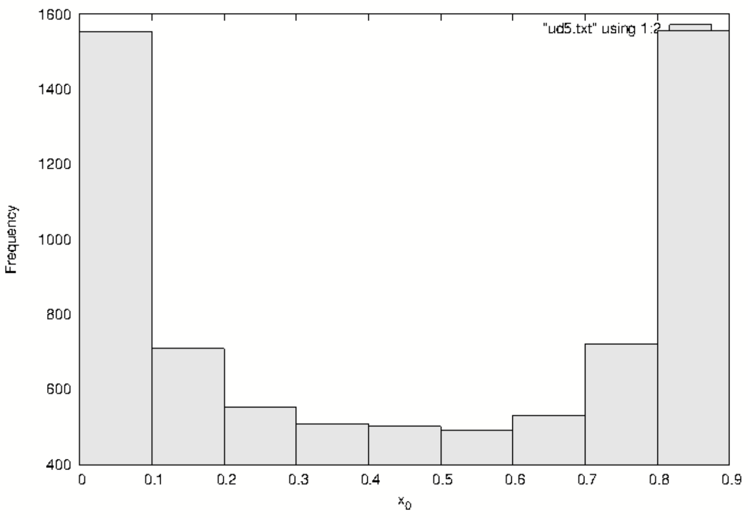


Figure 2.6: Frequency distribution of an ergodic orbit of logistic map.

CHAPTER 3

Chaotic Hash Function Based on Baptista Algorithm

3.1 Introduction

An “ideal” hash function as described by Bellare and Rogaway [17] is like a random oracle in which for each possible x, the value of $h(x)$ is completely random, like looking up its value in a giant book of random numbers. Chaotic functions due to their properties of sensitivity to initial conditions, ergodicity and mixing turn out to be good one-way functions that prove to be useful for designing secure hash functions.

3.1.1 Definition

A secure hash function is a function that takes a variable-length input string and converts it to a fixed-length (smaller) output string called hash value(HV) or message digest(MD). $h : (0,1)^{\star} \rightarrow (0,1)^n$ is such that h satisfies the three security properties: **collision resistance**, **preimage** and **second preimage resistance** [82].

Hash functions are classified into two types, unkeyed and keyed hash functions. Hash functions which use chaotic maps can be treated as keyed hash functions with a secret key. A secure hash function has to satisfy the three properties namely collision resistance, preimage resistance and second preimage resistance.

3.1.2 Properties of a secure hash function

Collision resistance

It is computationally infeasible to find two distinct inputs x, x' which hash to the same output, such that $h(x) = h(x')$.

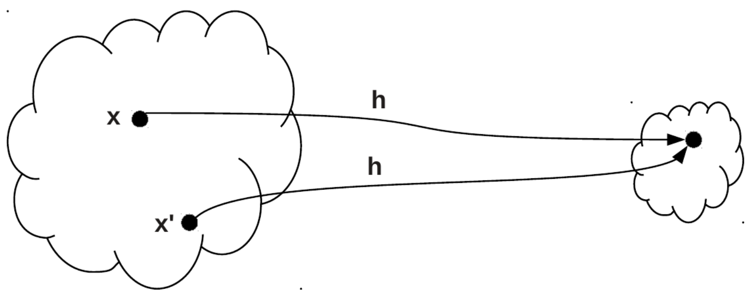


Figure 3.1: Collision resistance

Preimage resistance

It is computationally infeasible to find an input which hashes to a specified output, i.e., for any given output y to find a preimage x' such that $h(x')$=y for which a corresponding input is not known.

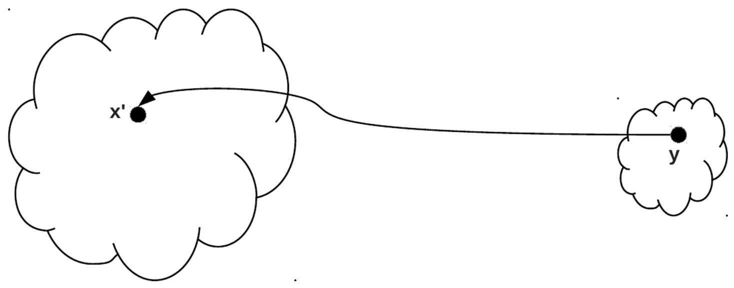


Figure 3.2: Preimage resistance

Second preimage resistance

It is computationally infeasible to find a second input which has the same output as any specified input, i.e., given x to find a second preimage $x' \neq x$ such that $h(x)$ = $h(x')$. Collision resistance implies second preimage resistance of hash functions but collision resistance does not guarantee preimage resistance[76].

3.1.3 Design of classical hash functions

Hash functions in the literature are one-way functions also called compression functions which compute hash values for fixed size domains. The compression functions can be extended to hash functions for infinite domains using a method

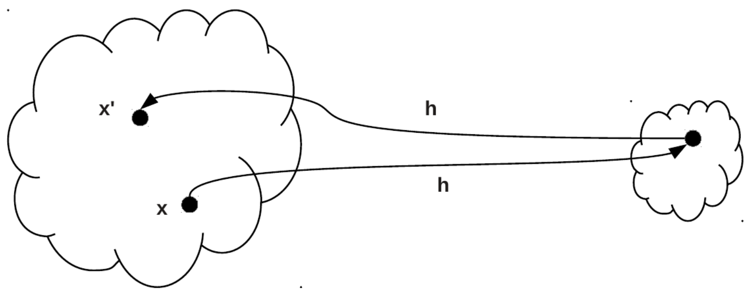


Figure 3.3: Second preimage resistance

called iterated hash functions. If the length of the hash value is m-bits, the preprocessing step converts the input string x to string y using a padding function. Then from the second step onwards, the compression function iteratively compresses the message block by block to finally output the m-bit message digest.

Step1: (Preprocessing:) Transform the given input x to y such that y is a multiple of m and that this transformation is injective.

Step 2: Divide the input into r blocks $B_0, B_1, \dots B_{r-1}$ each of size m.

$$y = (B_0, B_1, \dots B_{r-1})$$

Let IV be the initial vector for the hash function

Step 3:

$$z_0 = Compress\ (B_0, IV)$$

$$z_1 = Compress\ (B_1, z_0)$$

...

$$z_{r-1} = Compress\ (B_{r-1}, z_{r-2})$$

More generally,

$$z_i = Compress\ (B_i, z_{i-1}), 1 \leq i \leq r-1$$

Finally, the hash value is generally taken to be the m-bit string z_{r-1}

The state of the art hash functions like MD5 and SHA1 of classical cryptography follow Merkle-Damgard construction which adopts the iterated hash function approach. Almost all the chaotic hash functions in the literature follow the iterated hash functions scheme[26, 37, 75, 102, 105, 107]. In the algorithms of chaos-based cryptography, chaotic functions have been successfully used as compression functions [26, 37, 75, 101, 102, 105–107].

3.1.4 Challenges of using chaotic functions

Bit generation

Note that in the hash schemes that follow Merkle Damgard construction like MD5 and SHA1, z_i is an intermediate hash value (bit string of length m) that is used as a part of the input for the next step of iteration. Since chaotic functions perform real number computations, z_i's will be real numbers which need to be converted into a bit string separately.

Dynamical degradation of chaos

Due to finite precision of the machines, every orbit will be a periodic orbit. And while iterating a point, it may land in periodic cycles of small periods which leads to a tail of zeros. Surely it is not possible to generate non trivial binary representations of length say 80-bits in order to obtain 160-bits hash value (In our implementations we find that beyond 50-iterations it leads to zeros).

Speed of computation

Chaotic schemes are comparatively slower than the conventional cryptography algorithms. Hence one has to finely balance while designing a secure scheme to keep it computationally simple. In order to strengthen collision resistance, the current trend in chaotic cryptography is to take higher dimensional chaotic maps[105, 108], but these will slow down the computations tremendously.

Hence keeping all these issues in view we propose a generic design for a hash function as in Figure 3.4. We generate only a small number of bits in each round to avoid dynamical degradation and use intermittent perturbation and still maintain efficiency by using the 1-dimensional chaotic maps. **It is important to diffuse the bits in the preprocessing step, in order to avoid near collisions and preimage attacks**.

Comparison with MD5

Table 3.1 gives the comparison between proposed hash function and MD5.

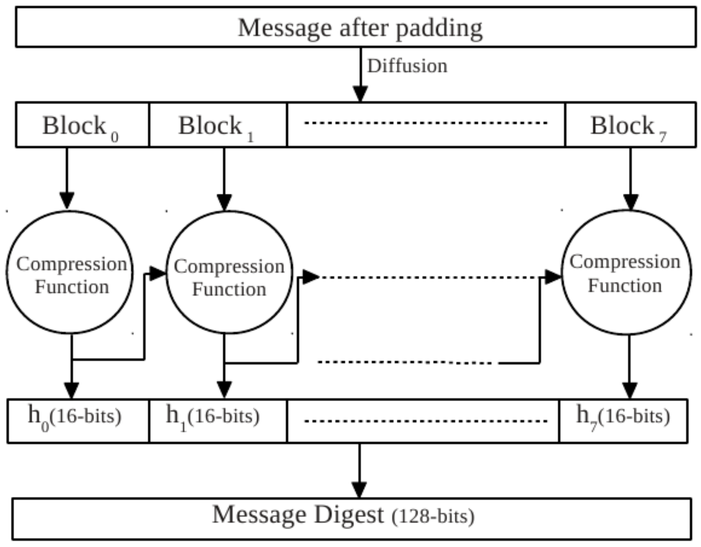


Figure 3.4: Block diagram of proposed hash function

MD5	**Proposed Hash Function**
Padding is done at the last block	Padding is done at the end of message which is then diffused into all blocks
Message is divided into k blocks of fixed length, $B_0, B_1..., B_k$	Message is divided into 8 blocks, $B_0, B_1, \ldots, B_7$
Block size is 512 bits	Block size is of variable length
Hash value = h_k	Hash value = $(h_0h_1h_2h_3h_4h_5h_6h_7)$

Table 3.1: Comparison with MD5

3.2 Background

The seminal paper of Baptista [16] on chaotic cryptography inspires us to propose a one-way hash based on chaotic maps. A thorough analysis of Baptista's symmetric encryption scheme was carried out by Alvarez et al.[5] and they show that Baptista's algorithm is vulnerable to all the four attacks of cipher text only, known plain text, chosen plain text and chosen cipher text. Problems with Baptista's encryption scheme are that ciphertext is not uniformly distributed and its size is larger than the size of plaintext. Encryption is quite slow and information like number of iterations is directly obtained from ciphertext. On the other hand, it was shown in the literature that Baptista's scheme has a lot of potential and could be modified to build a hash function.

K.W.Wong modified Baptista's algorithm by adopting a dynamic look-up table to avoid collisions and preimage attack [97] and then came up with a hashing scheme in 2003 [98]. Size of the hash value is not of fixed size but it varies and depends on the number of entries in the dynamic look-up table. X.Yi [106] in 2005 proposed a hash function based on one dimensional chaotic tent maps which is claimed to be better than Wong's scheme in its computational complexity. He follows the traditional Merkle-Damgard construction for compressing the input into a real number and then takes the binary representation of that real number as hash value. In 2005 Di Xiao [102] proposed one-way hash function using cipher block chaining method with changeable parameter of piece-wise linear map. Bits from the binary representation of real numbers of some fixed iterations are combined to get the 128-bit hash value. In 2006 Shiguo Lian et al.[62] used the diffusion property of neural networks and confusion property of chaotic function to generate hash values of different lengths of 128, 256 and 512 bits. Y Wang et al.[92] proposed a one-way hash function based on two dimensional coupled map lattices formed by logistic map as spatiotemporal chaotic system. Using linear transformation each 8-bit message is mapped to a number in the interval [0, 1] which is given to coupled map lattice as input for iteration. Cipher block chaining mechanism is adopted to strengthen the scheme.

Most of the schemes of recent literature use a multitude of chaotic maps[26, 75, 86, 103, 107], and sometimes neural networks etc to ensure ergodicity [62, 100, 101, 105, 109]. In our design, we show that judicious use of two chaotic maps achieves a secure hash function. The novelty is that we propose the method in a systematic fashion. We consider a basic chaotic hash algorithm based on Baptista's encryption scheme which is discussed in the next section and start strengthening it conducting analysis for security along the way. The hash function is proposed taking advantage of the strengths of Baptista's scheme and it is shown by computational analysis that the performance of the hash function is on par with recent chaotic hash function proposed in the literature without requiring a network of chaotic maps as in [103] thus improving the time complexity of the algorithm.

Baptista's encryption algorithm is discussed in section 3.3.1 and the design of hash function using his scheme in section 3.4 which is further enhanced to be resistant to cryptographic attacks in section 3.5. Performance analysis of the proposed hash function is given in section 3.6.

3.3 Design Motivation

3.3.1 Baptista's encryption algorithm

Baptista divides the input domain into intervals of size ϵ, I_a, where I_a is associated with the character a. The encryption algorithm maps each character a of the message to the number of iterations n that the logistic map takes to reach I_a.

Algorithm 1: Baptista's encryption algorithm.

```
1: Consider a portion $[x_{min}, x_{max}]$ of the attractor of the logistic map
   $F(x) = \lambda x(1-x)$, $\lambda$ the control parameter.
2: Divide the portion into $S$ uniform intervals $I_a, I_b, \ldots$, corresponding to
   characters $a, b$ and so on.
3: Let $P_i, i = 0, 1, 2, 3, \ldots$, be characters of plain text $P$.
4: $x_0 \in (0,1)$ be the initial seed value.
5: Iterate $F$ until $x' \leftarrow F^{n_i}(x_i)$ such that $x' \in I_{P_i}$, the interval corresponding to
   the character $P_i$.
6: $Cipher\ (P_i) = n_i$
7: Repeat Step 5 with $i = i + 1$ and $x_i = x'$ the new seed value for encryption
8: Output $Cipher(P) = (n_0 n_1 n_2 \ldots)$
```

3.4 Design of Hash function Based on Baptista's Scheme

Input message(with appropriate padding) is divided into eight blocks say $B_0, B_1, \ldots, B_7$. Choose the initial value $x_0 \in (0,1)$ and control parameter $\lambda \in (3.8, 4)$. In each block for each character m iterate the logistic map $f^{n_m}(x(m)) \in I_m$, where I_m is the interval associated to character m. Then $f^{n_m}(x(m))$ will be the initial value for the next character in the block. Repeat this till the whole block is processed. Then take binary representation of the value n obtained from $f^{n_m}(x(m))$ of the last character as hash value for that block. This process is carried out for the next block as well by taking the initial seed from the output of the previous block. Finally juxtapose the hash values obtained from all blocks which give the 128-bits hash value for the input message.

3.4.1 Discussion

Some fundamental problems with this design are that it can lead to preimage and second preimage attacks as follows.

Suppose the first block B_0 contains k characters and $h_0 = n$. Let x_0, λ be known. We show how a preimage for n can be constructed. First note that

$$f^{n_{k-1}}(\cdots f^{n_2}(f^{n_1}(f^{n_0}(x_0)))) = f^{n_0+n_1+\ldots n_{k-1}}(x_0).$$

Choose any sequence of integers that ends in n, such as $n_0n_1\ldots n_{k-1}n$. We can construct the preimage of y as follows: Find character p_0 where $f^{n_0}(x_0) \in I_{p_0}$. Now reset x_0 as $f^{n_0}(x_0)$. Then find p_1 such that $f^{n_0+n_1}(x_0) \in I_{p_1}$ and so on. Finally we get p_{k-1} such that

$$f^{n_0+n_1+\ldots n_{k-1}}(x_0) \in I_{n_{k-1}}.$$

Then the required preimage for $y = n$ is $p_0p_1\ldots p_{k-1}$. Following the above scheme, second preimages can be found.

3.4.2 Insights from the above scheme

- This scheme is not *preimage* and *second preimage* resistant. If a hash function is not second preimage resistant, then it will be automatically prone to *collisions*[22].
- *Cipher block − chaining* and *perturbing initial value* for each byte may avoid *preimage* attacks.
- n corresponding to each block present in the hash value leads to preimage attacks.
- The hash value bits of each block are binary representations of possibly small integers which leads to *dynamical degradation*. We need to produce *large n*.
- If we take two plaintexts P1 and P2 which differ only in the last block, i.e., if $B_0, B_1, \ldots, B_6$ are identical, then the corresponding output hash values, i.e., $h_0, h_1, \ldots, h_6$ will be identical. This can be seen as near-collision. To avoid this we need to introduce $diffusion$.

All these issues are addressed for designing a strong hash function which is given in the next section.

3.4.3 Enhancements to the design

We strengthen the basic design by incorporating diffusion, perturbation and a scheme for generating large integers which are discussed below.

Diffusion

We introduce diffusion by first distributing the bits of each character into eight blocks in some order, so that each block gets the bits from each character i.e, influence of every character is seen in every block as shown in the Figure 3.5. Two plaintexts P1 and P2 which differ only in the last character will get different $h_0, h_1, \ldots, h_7$ values.

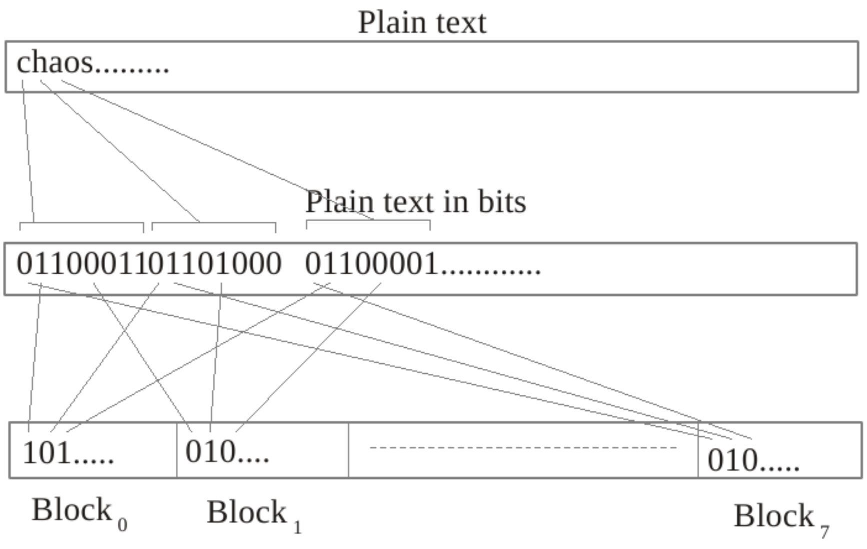


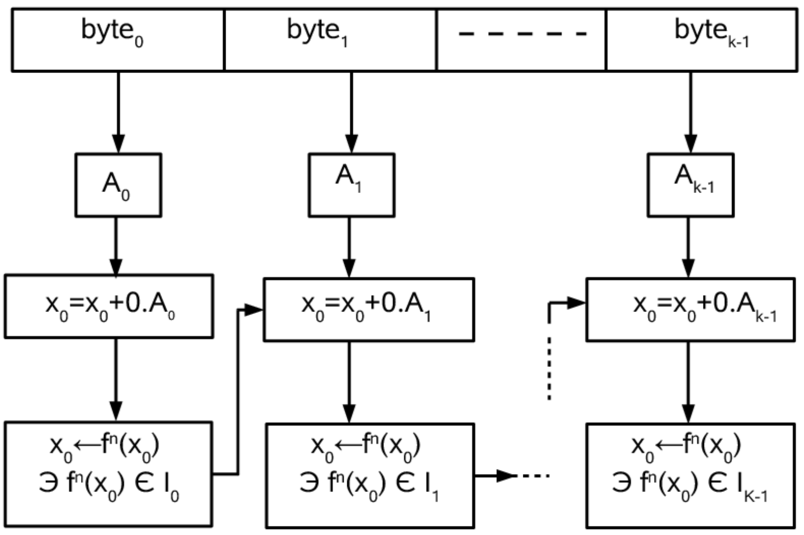
Figure 3.5: Diffusion

Intermittent perturbation

During implementation, dynamical degradation may occur due to short-length chaotic orbits[2]. This weakens the security and desired statistical properties[50, 56]. To overcome dynamical degradation, perform timely perturbation of the chaotic system[52, 78, 110].

Generating large n

It was also found in the experiments the small integers that get generated as hash values lead to collisions. We propose to take a vote by using a tent map to decide if n that emerges at the end of the inner loop is to be chosen as h_n or iterate f further to pick the next possible n.



byte_0
byte_1
byte_{k-1}
A_0
A_1
A_{k-1}
x_0=x_0+0.A_0
x_0=x_0+0.A_1
x_0=x_0+0.A_{k-1}
x_0←f^n(x_0) ∋ f^n(x_0) ∈ I_0
x_0←f^n(x_0) ∋ f^n(x_0) ∈ I_1
x_0←f^n(x_0) ∋ f^n(x_0) ∈ I_{K-1}

Figure 3.6: Perturbing x_0 and evolving a compression function

Tent map T is defined as follows:

$$T(x) = \begin{cases} t * x & \text{if } 0 \leq x \leq \frac{1}{2} \\ t * (1 - x) & \text{if } \frac{1}{2} \leq x \leq 1, n \geq 1 \end{cases}$$

Fix an η, $0 < \eta < 1$. Check if $T^n(x) > \eta$. If yes, accept n else repeat the iteration step of the algorithm to find the next bigger n. This scheme is depicted in the Figure 3.7. In our implementations η is 0.98 and t is 1.99.

Now we propose our algorithm for hash function incorporating all the above enhancements to the basic design.

3.5 Proposed Algorithm : Baptista Hash Function

We name our proposed algorithm which is inspired by Baptista encryption scheme as Baptista hash function.

Preprocessing (Padding): Original input message P is padded such that the length of the message is a multiple of 64. Padding bits are $1000\ldots0$ followed by length of the message i.e., binary($\mid P \mid$).

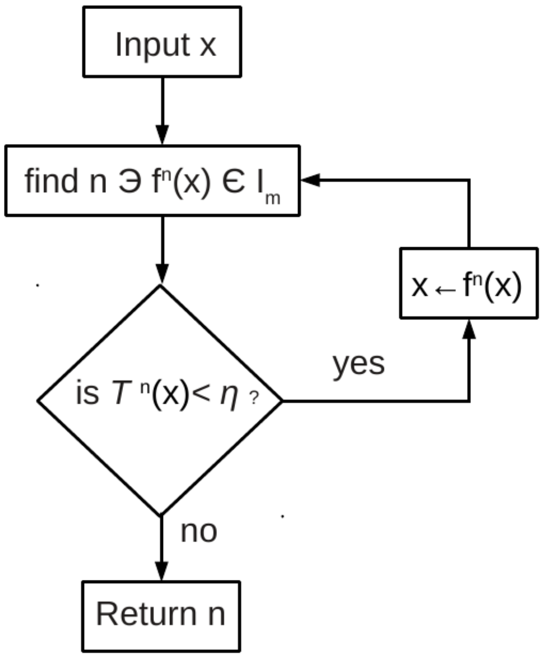


Figure 3.7: Scheme to find large n's using $Vote_{Tent}$ map

3.5.1 Summary

The message is padded such that its length is a multiple of 64, hence length of each block itself is a multiple of eight which makes character computations possible. As eight bits of each character are distributed among eight blocks under diffusion, the influence of every character will be on every block. $\lambda \in (3.8, 4)$ is chosen such that the logistic map is ergodic and possess positive Lyapunov exponent[23]. Note that two integers n_1 and n_2 are generated in steps 10 and 11 using logistic map and $Vote_{Tent}$. The smaller integer n_1 is used for computing the seed of the next block which is kept opaque from the user since $h_i = BinaryRep(n_1) \oplus BinaryRep(n_2)$.

3.5.2 Cryptanalysis

Preimage and second preimage analysis

Suppose the key x_0 is known and the hash value is given, can we construct a preimage? Clearly since the hash value is a concatenation of eight block values, the corresponding integers can be obtained from 16-bits of the hash value. But since $h_i = n_1 \oplus n_2$ for the ith block, neither n_1 nor n_2 can be obtained. Since n_2 is larger than n_1 the first few bits of n_2 may be visible where as no information of n_1 is visible. Thus preimage attack are avoided since n_1 has been used internally

Algorithm 2: Baptista hash function

```
1: **Diffusion**: Distribute the bits of each character in the plaintext into eight
   blocks $B_0, B_1, \dots, B_7$, one bit each in some order.
2: Choose $0 < x_0 < 1$, $\lambda \in (3.8, 4.0)$ the control parameter of the logistic
   equation $f_\lambda(x) = \lambda x(1-x)$ and $I_m$ interval associated to character $m$ and
   set a threshold $0 < \eta < 1$.
3: for Block $B_i = 0$ to 7 do
4:    for Byte $m = 0$ to $k-1$ of the message of a block $B_i$ do
5:       $A(m) =$ ASCII value $(m)$, $a(m) = 0.A(m)$
6:       Perturb $x_0$: $x'(m) = (a(m) + x_0)\ mod\ 1$
7:       Iterate the logistic map until
         $(f^{n_m}(x'(m)) \in I_m)$ && $(n_m = Vote_{Tent}(x'(m)))$
8:       $x_0 \leftarrow f^{n_m}(x'(m))$
9:    end for
10:   $n_1 = n_{k-1}$
11:   Iterate the logistic map until

         $(f^{n_{k-1}}(x'(k-1)) \in I_{k-1})$ && $(n_2 = Vote_{Tent}(x'(k-1)))$

12:   Define for the block $B_i$, $h_i = BinaryRep(n_1) \oplus BinaryRep(n_2)$. Thus a
      block gets hashed into 16 bits.
13: end for
14: The final hash value of the plain text $P$ is obtained by concatenating $h_i$'s i.e
    $h(P) = (h_0h_1h_2h_3h_4h_5h_6h_7)$
```

for cipher block chaining.

Near collision resistance

To illustrate the effect of diffusion which makes the algorithm resistance to near collisions, we take two input messages P_1 and P_2 which are identical except in the last character.

Input message P_1: cryptography
128-bits hash value:
1101001110110000010110101111000001001001111000001101000110010000
0111100110110000010011000100000010100100111010001110111000000100
Hash value in hexadecimal format :
D3B05AF049E0D19079B04C40A4E8EE04

Input message P_2: cryptographe
128-bits hash value:
1001101101000100001110011110000000010111000100000011111110000000
1111111111100000111011110011100010111110001000001110100011111000
Hash value in hexadecimal format :

9B4439E017103F80FFE0EF38BE20E8F8

Due to diffusion step that is carried out, bits of the last character are distributed among all the eight blocks. Influence of each character is on every block. The resultant hash values show that the proposed design is resistant to near collisions. In this example the hash values of two messages differing only in the last character differ by 56-bits.

3.5.3 Algorithm analysis

If M is the length of the message to be hashed, the time taken by the proposed algorithm is of the order of $O(n * M)$, where n is bounded by the value generated by the $Vote_{Tent}$. The number of iterations that a character has to undergo in the worst case is 64,000 which is really huge.

3.5.4 Implementation details

Proposed algorithm is implemented in PERL-language on a PC with 32-bit CPU with linux operating system(openSUSE 11.0). Initial value x_0 is 0.345671, control parameter λ is 4 and η is 0.98. Minimum iteration bench mark is set as n=250 and maximum as $n = 30000$ to retain the randomness of the output n.

3.6 Performance Analysis

The proposed Baptista hash algorithm is implemented and tested for sensitivity, confusion and diffusion properties, further collision analysis is conducted and results are presented in this section.

3.6.1 Sensitivity of the input message

Simulation of proposed hash function is carried out under different cases and their corresponding 128-bit hash values in hexadecimal format are given below.

Case 1: Original message
Input message : How Random is a Coin Toss.
Hash value in hexadecimal format:

1EE02788619011C0A1B886C0DA60C29C

Case 2: Delete the letter "a" from the word "Random"
Input message : How Rndom is a Coin Toss.
Hash value in hexadecimal format:
E818F8305198ADB07B987338D8300CE0

Case 3: Change the character "." to "!"
Input message : How Random is a Coin Toss!
Hash value in hexadecimal format:
B11867A8B2D8A8848200D29C2E708744

Case 4: Insert " " for the word Toss
Input message : How Random is a Coin "Toss".
Hash value in hexadecimal format:
5BCCDDFC7DC84E6094D00D002E18D3F0

From the simulation results it is shown that tiny difference in the input message will cause a huge change in the hash value thus satisfying the sensitivity property.

3.6.2 Statistical analysis of diffusion and confusion

Diffusion and confusion are the two important properties Shannon introduced for the design of hash functions[80]. These two hide the redundancy of the input message. Diffusion spreads the influence of every character in the input over the entire ciphertext in order to hide the statistical structure of plaintexts. Confusion complicates statistical dependance of ciphertext on plaintext using transformation. To satisfy theses two properties the ideal diffusion effect for each bit of hash value must be 50% for any tiny changes in the input message.

In order to evaluate the performance of diffusion and confusion, statistical tests are carried out as proposed in the literature[103]. Let P be the input message and P' denotes the message in which ith bit is toggled. Then $d_H(h(P), h(P'))$ denotes the number of bits changed in output with a 1 bit change in the input. The minimum hamming distance,

$$d_{min} = \min_{1\leq i\leq N} d_H(h(P), h(P'))$$

and maximum hamming distance ,

$$d_{max} = \max_{1\leq i\leq N} d_H(h(P), h(P'))$$

are computed. Further, the average d_{avg} and the standard deviation d_{std} of the distribution of hamming distances over N trials are computed to get a better idea of the distribution. We tabulate the performance of the hash function in terms of these four hamming measures.

We tabulate the performance of the hash function in terms of these four hamming measures $d_{min}, d_{max}, d_{avg}, d_{std}$. Input message of size 780KB is taken for generating hash value say MD which is a 128 bit stream. A bit i is randomly chosen and toggled in the message. Let the hash value of the perturbed input be MD_i.

Experiments are carried out for $N = 10,000$ toggled messages for proposed algorithm. It can be clearly seen from Table 3.2 that the messages obtained by changing one bit in the original message exhibit hash values that have, on average, nearly 64% of the bits different from the original Message Digest. These statistical results shows the diffusion and confusion capabilities are stable and strong.

N	256	512	1024	2048	4096	8192	10000
d_{min}	48	46	45	43	42	42	42
d_{max}	79	80	82	83	83	85	85
d_{avg}	63.43	63.51	63.47	63.49	63.54	63.56	63.56
d_{std}	4.71	4.74	4.81	4.83	4.82	4.84	4.83

Table 3.2: Statistical performance of number of bits changed for Baptista hash function

Figure 3.8 shows that the proposed algorithm exhibits desirable security with the number of changed bits due to a 1-bit toggle in plain text being nearly 63.5 in the 128 bit message digest which is very close to the ideal value of 50% probability. Further all the results achieved are comparable to those of the recent work as shown in the Table 3.5. This shows that the proposed function exhibits strong capability of diffusion and confusion. The space in which the tests are conducted is large enough to indicate that the values obtained by d_{avg} etc lie close to the true values of the distribution.

It is important to note that the proposed chaotic algorithm makes use of only iteration of two maps, the logistic map and the chaotic tent map and achieves nearly 64-bit diffusion where as the scheme proposed by Yang et al.[103] which uses a 16 chaotic map network shows slightly improved results with a significant increase in the computational complexity.

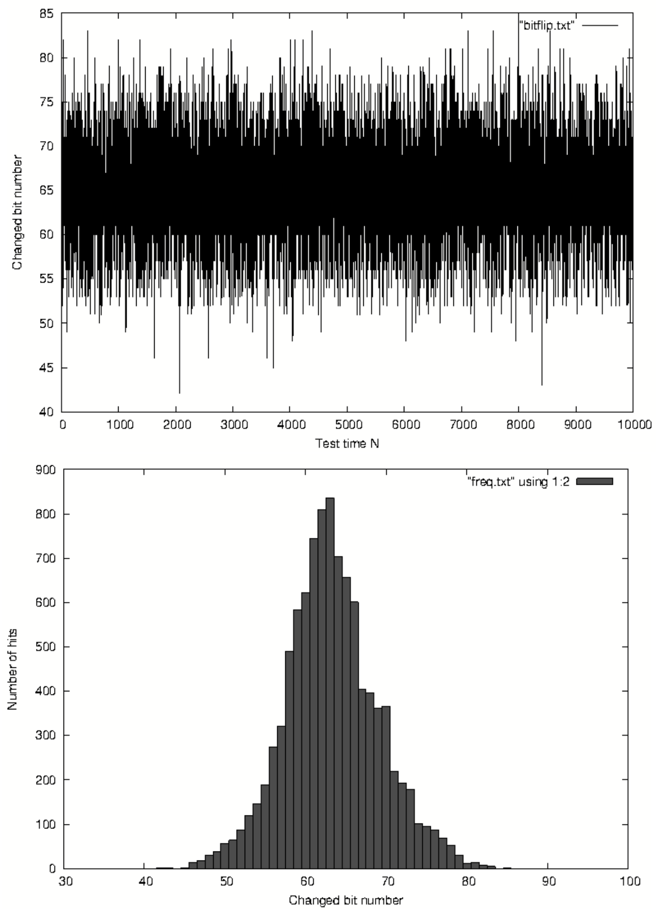


Figure 3.8: Distribution of hamming distance for a 1-bit change in the input message

3.6.3 Looking for a collision in 1-bit neighborhood of message

Providing proof on the collision resistance theoretically is not so easy for chaotic functions. We do collision resistance analysis following the approach used by [102, 103, 108]. The hash value for an input message is generated and stored in ASCII format. Then a bit in the message is selected randomly and toggled. A new hash value is then generated.

$$W = \sum_{i=1}^{N} f(t(e_i), t(e_i{}')) \tag{3.1}$$

where e_i and e_i' are the i^{th} ASCII character of the original and the new hash value, respectively and the function $t(.)$ converts the entries to their equivalent decimal values.

The absolute difference between the two hash values is calculated using the formula:

$$d = \sum_{i=1}^{N} |(t(e_i) - t(e_i'))| \quad (3.2)$$

The two hash values are compared and the number of ASCII characters with the same value at the same location is counted using W in the formula 3.6.3. where

$$f(x,y) = \begin{cases} 1 & \text{x=y} \\ 0 & \text{x} \neq \text{y} \end{cases}$$

$W(i)$ denotes the number of experiments in which exactly i positions have the same characters as the original hash value.

This experiment is carried out for N=10,000 times. The results are given the Table 3.3 The maximum, mean, and minimum values of d are listed in Table 3.3. The experimental values of W are analyzed and we obtained $W(0) = 9325$, $W(1) = 649$, $W(2) = 25$, $W(3) = 1$ and for $W(k) = 0$ for $k > 3$. The distribution W of the number of ASCII characters with the same value at the same location in the hash value is shown in Figure 3.9.

Absolute difference d	Maximum	Minimum	Mean
Proposed scheme	2428	534	1352

Table 3.3: Absolute difference of two hash values

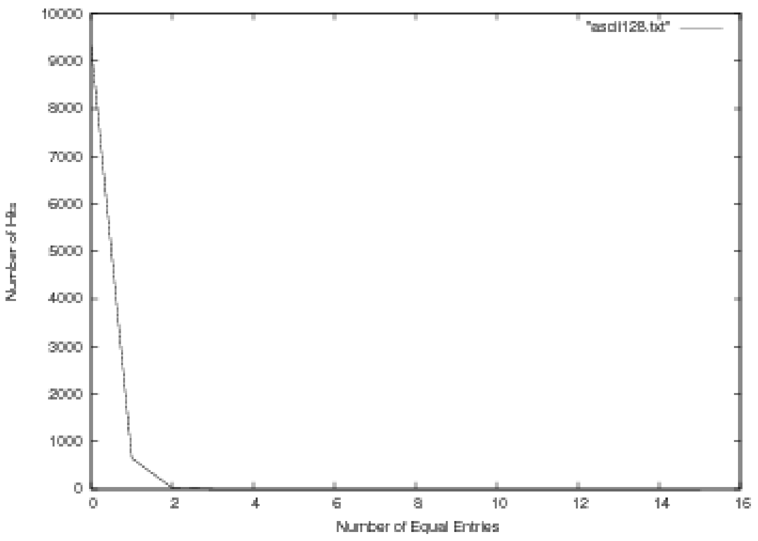


Figure 3.9: Distribution of the number of ASCII characters with the same value at the same location in the hash value

Figure 3.9 shows that the chaotic hash function is performing very well with the maximum number of equal characters in MD obtained being only 3 and the collision probability being quite low.

3.6.4 Looking for a collision in the whole space i.e for a large number of different randomly generated messages

N input messages say $P_1, P_2, P_3, \cdots P_N$ of length 8,00,560 bits are chosen randomly to test for collisions. Hash values are computed for $P_i, i = 1, \ldots, 10,000$ here and it is observed that no two hash values coincide and completely different hash values are obtained for all the 10,000 input messages.

Hash values obtained for a sample of ten input messages is shown in the Table 3.4.

Input	Message Digest
P1	29134CDF15C3C0A15FAE111C2C73735F
P2	7AA104D1485C38FF5C51369E254A1261
P3	FE65864917E1009A421F23913B8302EC
P4	C2E60DEE6A7F379901CF10AF7321C4D5
P5	9EB039E2EF94BAA11C1209A9EB56B1D
P6	1738740F6E430965EE481FC3239C9904
P7	46AF3A1B836078F902005ED981F51F60
P8	6FF5D8DA47D19F93A33B353A2E4256CB
P9	1AC493E81885CF2401CE1CB149B21A76
P10	546509AC2AF93A69067A1E0ACB451CC3

Table 3.4: MD generated for the input messages $P_1, P_2, P_3, \cdots P_{10}$ for collision test of proposed hash function

3.6.5 Comparison with the existing algorithms in the literature

Proposed Baptista hash algorithm is compared with the traditional hash functions MD5 and SHA1 and along with other chaotic hash functions for performance. The average number of bits obtained in the hash value of the 10,000 1-bit toggled messages is computed. Table 3.5 shows the comparison of existing chaos based hash functions[102, 103, 106] along with the traditional hash functions MD5 and

SHA1. X. Yi[106] follows Merkle-Damgard construction for compressing the input message as carried out in traditional schemes MD5 and SHA1. For bit generation Xiao et al. and X. Yi [102, 106] use binary representation. The proposed hash function achieves d_{avg} of 63.56 bits on par with the existing algorithms by using only two chaotic maps where as Yang [103] use a network of 16 maps.

	Proposed algorithm	Yang [103]	Di Xiao [102]	X. Yi[106]	MD5	SHA1
d_{avg}	63.56	64.06	63.98	63.91	64.03	79.86
d_{std}	4.83	4.41	4.43	4.36	4.42	3.91

Table 3.5: Statistical comparison of changed bit number with the algorithms in the literature

From the analysis in Table 3.5 d_{avg} of the proposed Baptista hash function is close to the ideal 50% change bit value depicting stability of diffusion and confusion of the hash function.

3.7 Discussion & Conclusions

The chaotic hash functions that are proposed in the literature generally have a multitude of functions networked together to make the compression function secure. In this chapter, starting with a nice encryption scheme that is proposed by Baptista which is used to design a hash function, it is argued logically why certain plug-ins are required and how these schemes help in making the function secure. A new one-way chaotic hash function is developed based on Baptista's encryption algorithm that gives an output of 128 bit message digest. It is not possible to prove collision resistance theoretically, hence following similar work in literature we present computational results. The proposed Baptista hash function exhibits nearly 50% mixing of bits in the output of message digest on a one-bit change in the input message. The function is further analyzed and shown to possess preimage and second preimage resistance. It is shown that the performance of the hash function is comparable with some of the latest algorithms proposed in the literature. Major drawback of the algorithm is the time taken by $Vote_{Tent}$. The number of iterations that a character has to undergo on the worst case is 64,000 which is really huge.

The algorithm can be adapted to compress a message to hash digest of bigger lengths with slight modifications. The number of blocks n into which message is divided is such that $\frac{\lfloor length(MD) \rfloor}{n} \leq 16$ since binary representation of each integer gives 16 bits. This is a hard constraint and hence not desirable. Also using information like number of iterations in the output may violate security constraints. Hence in the next chapter we rectify these issues and refine the design of the hash function and show that it satisfies all the requirements that are laid out by Shujun Li [2] to evolve into a secure hash function.

In the next chapter, the design of hash function is enhanced with a simple compression scheme and a novel and efficient bit generation scheme. Computational complexity is reduced by removing the use of second chaotic map within the compression function.

CHAPTER 4

Chaotic Keyed Hash Function using Bernoulli Shift Map

4.1 Introduction

In this chapter a keyed hash function is proposed using the iterated function scheme as described in Figure 3.4 of chapter 3. A chaotic hash function can be treated as a keyed hash function since the parameters of the chaotic map can be considered as secret keys. The emphasis in this chapter is placed on the secret key parameters of the hash function in order to conduct a more thorough and elaborate analysis regarding sensitivity of the keys.

Baptista hash function has serious limitations since the bit generation step is very time consuming and bit generation from binary representation of integers is not scalable. Keeping these points in view, we propose to use novel mechanisms to tighten the bit generation step keeping the same over all design as described for Baptista hash function.

4.1.1 Definition

A keyed hash function $H_k()$ is a secure keyed one-way hash function which produces a hash value(HV) or message digest(MD) of length n satisfying the following properties[79, 82]: Given key k and message M it is

1. easy to compute $H_k(M)$,
2. hard to find M, given $H_k(M)$ without knowing key k,
3. hard to find $H_k(M)$, given M without knowing key k,
4. $H_k()$ must exhibit necessary diffusion and confusion in order to thwart statistical attacks,
5. $H_k()$ must generate at least 128-bits hash value in order to thwart birthday attacks,

6. $H_k()$ must have enough key space in order to thwart exhaustive key search,
7. $H_k()$ is keyed collision free. Without knowing k, it is hard to find two distinct messages M and M' that collide under $H_k()$.

$H_k()$ acts on data of arbitrary size using a few secret keys and produces output of fixed size known as hash value or message digest. Formally, a hash function is said to be secure if it satisfies the properties of collision resistance, preimage and second-preimage resistance as defined earlier in 3.1.2.

4.2 Background

The control parameters of a chaotic map are generally used as secret keys for chaos based cryptosystems. Choosing the value of control parameter having a positive value of the Lyapunov exponent plays an important role in providing required confusion and diffusion properties for the chaotic cryptosystem[11]. Some of the cryptosystems fail to choose such a parameter[32, 71, 90] and their vulnerability is proven in[5].

Zhang et al.[108] propose a keyed hash function based on feed forward- feedback nonlinear filter, an nth order chaotic system. They use a nice quantization function to generate 128-bits for each block which are then used for the next block in a typical cipher block chaining fashion to compute the final hash value. Xiao et al.[101] propose a parallellizable hash computation mechanism by computing hash values for each block separately and final hash value is computed by using X-OR operation. They use a chaotic neural network for compression scheme. For each block four output neurons are generated that are transformed using binary representation into four 32-bit strings which are juxtaposed to obtain 128-bits bit string. Clearly they use four output neurons instead of one or two to avoid dynamical degradation. The scheme of Xiao et al. is cryptanalysed by [30, 91] and the scheme[101] is shown to be prone to forgery attacks and the keys used are found to be weak.

A Kanso et al.[37] propose a keyed hash function based on a single chaotic map i.e, skewed tent map and generates a hash value whose length is flexible.

4.3 Design Motivation

Unlike the traditional iterated hash function schemes where the message is divided into 160-bit blocks and intermediate hash values are obtained at these points, chaotic maps generate real numbers at intermediate steps. Hence the message being divided into blocks whose size is a multiple of message digest length is an artificial requirement for chaotic hash functions. Thus we propose a generic design for a hash function as follows:

Step1: Divide the input into 8 blocks of equal bits.

$$M = (B_0, B_2, \dots B_7)$$

Let x_0 be the initial value parameter for the hash function

Step 2:

$$x_{B_0} = Compress\ (B_0, x_0)$$

$$x_{B_1} = Compress\ (B_1, x_{B_0})$$

. . .

$$x_{B_7} = Compress\ (B_7, x_{B_6})$$

More generally,

$$x_{B_i} = Compress\ (B_i, x_{B_{i-1}}), 1 \leq i \leq 7$$

At each round of $Compress$, Baptista hash function outputs bits corresponding to iteration number which is not desirable both from preimage attack point of view and from the perspective of dynamical degradation.In the present scheme, we use real number instead of integers and use all the x_{B_i}'s to generate the MD in order to avoid potential preimage attacks.

We propose an elegant design for a generic chaotic keyed hash function based on iterated hash function approach as described above. The proposed scheme basically compresses each block of data into a real number using a chaotic function to which then, Bernoulli shift map is applied to generate bits which form the message digest. We refer to this scheme as Bernoulli keyed hash function. The block diagram for the keyed hash function is presented in the Figure 4.1.

We claim that one could plug in a chaotic function of the author's choice in the compression block and generate a new hash function.

4.4 Proposed Bernoulli Keyed Hash Function

The complete algorithm for the proposed chaotic keyed hash function is as described in algorithm 3.

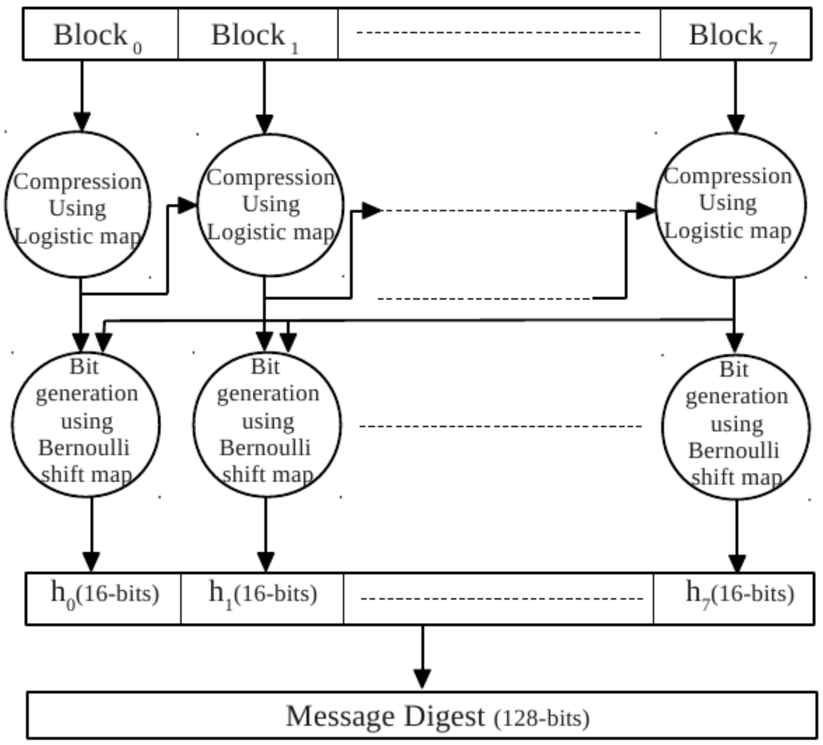

Block $_0$
Block $_1$
Block $_7$
Compression Using Logistic map
Compression Using Logistic map
Compression Using Logistic map
Bit generation using Bernoulli shift map
Bit generation using Bernoulli shift map
Bit generation using Bernoulli shift map
h_0(16-bits)
h_1(16-bits)
h_7(16-bits)
Message Digest (128-bits)

Figure 4.1: Generic block diagram of Bernoulli keyed hash function

Preprocessing (Padding): Original input message P is padded such that the length of the message is a multiple of 64. Padding bits are $1000\ldots0$ binary of length $\mid P \mid$.

Diffusion : Number of blocks is fixed to be eight and the input message is diffused such that each block gets a bit from every byte of the input message to avoid near collisions.

4.4.1 Compression function

The compression block was designed, through a lot of experimentation, in order to ensure that it is not vulnerable to attacks.

The block diagram which elucidates the algorithm of $CompressFunction$ is depicted in Figure 4.2.

4.4.2 Hash Bernoulli : Bit generation step

Bit generation is given in $HashBernoulli$ algorithm 5.

Algorithm 3: Bernoulli Keyed Hash Function

```
1: Input message is padded such that the length of the message is a multiple of 64.
2: **Diffusion**: Divide the input message into 8 blocks. Distribute 8 bits of each character in the plaintext among 8 blocks one bit per each block.
3: **Permutation**: Rearrange the 8 blocks according to a fixed permutation on $\{0, 1, \ldots 7\}$.
4: Input secret keys $x_0 \in (0, 1)$ and $\lambda \in (3.8, 4)$
5: for each Block $B_i, i = 0$ to 7 do
6:    if $(i == 0)$ then
7:       $x_{B_{-1}} = x_0$
8:    end if
9:    $x_{B_i} \leftarrow$ CompressFunction(Block $B_i$, $x_{B_{i-1}}$);
10: end for
11: for each Block $B_i, i = 0$ to 7 do
12:    $h_i \leftarrow$ HashBernoulli($x_{B_i}$);
13: end for
14: Return $h = h_0 h_1 \ldots h_7$ /* message digest */
```

Algorithm 4: CompressFunction(Block B_i, $x_{B_{i-1}}$)

```
1: $x_0 = x_{B_{i-1}}$ /* Initial value for current block */
2: for each Byte $c = 0$ to $k - 1$ do
3:    $x_c = (x_0 + 0.ASCII(c)) mod\ 1$ /* Perturb $x_c$ */
4:    $n = ASCII(c)$ /* Choose integer for iteration */
5:    for $iter = 0$ to $n - 1$ do
6:       $x_c \leftarrow \lambda * x_c * (1 - x_c)$ /* Iterate logistic map $n$ times*/
7:    end for
8:    $x_0 \leftarrow x_c$ /* Initial value for the next character */
9: end for
10: $x_{B_i} = x_0$ /* Initial value for the next block */
11: Return $x_{B_i}$
```

Algorithm 5: HashBernoulli(x_{B_i}, x_{B_7})

```
1: $y_0 = (x_{B_i} + x_{B_7})\ (mod\ 1)$
2: $y \leftarrow \sigma^m(y_0)$ /* Leave first $m$ iterations of $\sigma(y_0)$, where $m < 30$*/
3: for $j = 0$ to 15 do
4:    if $(2 * y > 1)$ then
5:       $h_i(j) = 1$
6:    else
7:       $h_i(j) = 0$
8:    end if
9:    $y \leftarrow 2 * y(mod 1)$
10: end for
11: Return $h_i$
```

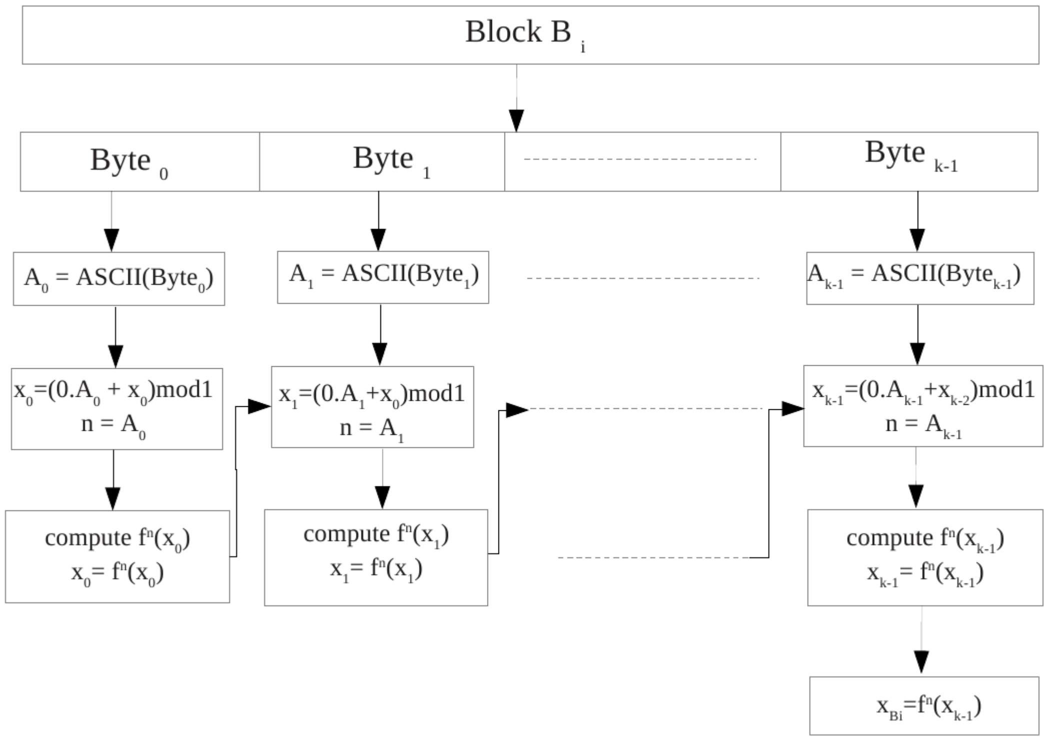


Figure 4.2: Compression function that uses secret key x_0 and λ along with the logistic function $f(x) = \lambda * x * (1 - x)$

By concatenating the 16-bits obtained from the eight blocks we get a hash value of 128-bits. We can also generate a hash value of 160-bits by generating 20-bits using $HashBernoulli$ algorithm for each block.

4.4.3 Summary

We evolve a generic design for weaving the plaintext into a chaotic map in order to achieve a hashing scheme. Input message is padded so that the length of the message is a multiple of 64. Then the message is divided into eight blocks using the concept of diffusion as done in section 3.4.3 of the previous chapter. Each block is then compressed using the chaotic logistic map to obtain a real number. Block chaining method is used to tighten the compression procedure. The novelty of the scheme lies in using the Bernoulli shift map that acts on this real number and iterated to generate a 16-bit hash value for each block. By concatenating all the hash values obtained from the eight blocks, a message digest of 128-bit is obtained. Following the rules laid by S. Li [55], we designed our algorithm. Number of iterations of a chaotic map or number of encryption rounds should not be used as a part of secret key[11]. If they are used as a part of secret key then timing attack can be used to break such a cryptosystem[12]. Taking initial condition and control parameter as a secret key[11] is a good option to avoid bruteforce attack[4]

Compression step

For the method to work, one has to ensure a 'strong' initial seed so that shift map does not terminate quickly by generating zeros. Clearly, periodic points would lead to trivial output. Since technically, every point is a periodic point as a machine cannot generate an infinite decimal, we call points with large orbits as 'machine-aperiodic'. Through the first step of compression scheme which uses the chaotic logistic map, we ensure that a machine-aperiodic initial seed is generated, which is then used by the shift map to generate an almost random 16-bit sequence output.

Perturbation

In this work, it is observed that perturbation of x_0 as each character is scanned, avoids *dynamical degradation* and makes the scheme to be strong. Note that the perturbation factor as well as the number of iterations is chosen based on the character being scanned and hence the scheme is nicely interwoven with the text being scanned. Due to the finite precision of machines, we cannot expect long non-periodic sequences. Hence, we do not generate the entire MD from a single x_{B_i} as it may lead to collisions.

Bernoulli shift map

Mapping hash value, block by block, independently is dangerous as it induces vulnerability to preimage resistance. Hence hash value of each block is computed by adding x_{B_i} obtained from that block to that of the last block as depicted in the *HashBernoulli* algorithm. x_{B_i} obtained from the compression function of each block is perturbed with x_{B_7} obtained from the compression function of the last block to avoid near collisions. We leave some initial iterations of $\sigma(y_0)$, where $y_0 = (x_{B_i} + x_{B_7}) \ (mod\ 1)$. Then Bernoulli shift map is applied on y to extract 16-bits hash value. The shift map exhibits all the hallmarks of chaos. The power of the shift map lies in the fact that after a few shifts, the significant bits of x_0 are lost and the output generated $\sigma^n(x_0)$ will be nearly random[24].

4.4.4 Implementation details

In classical cryptography key size is n-bits. Then every n-bit key is uniformly distributed and equally strong. In chaotic cryptography key space is nonlinear and not equally strong as some of the intervals lead to periodic orbits[4]. According to all the above constraints our secret key consists of two parameters x_0 and λ and their intervals are $x_0 \in (0, 1)$ and $\lambda \in (3.8, 4)$ in order to avoid non-chaotic regions[55]. Implementation details of many chaos-based cryptosystems are neglected. These are very important to analyze the security and performance of the cryptosystem. With out these details it is difficult to evaluate the significance and reliability of the cryptosystem[55].

The implementation is carried out on a PC with 32-bit CPU with linux operating system(openSUSE 11.0) using its inbuilt PERL programming language which can handle upto 15 decimal places.

4.4.5 Algorithm analysis

It should be clarified that non-chaotic algorithms like MD5 and SHA1 will certainly be much faster than the chaotic algorithms. If M is the bit-length of the message to be hashed, the time taken by the proposed algorithm and those proposed in the literature[101, 108] are all of the order of $O(n * M)$, where n is the number of iterations per character and M is the length of P.

The number of iterations per character n for our algorithm is in the range (0, 255). By amortized analysis, we can show that n is less than 125. In comparison, n for [108] is 10 and n for [101] is $(208 + 13\tau)/64$ where $\tau = 50$, which is nearly 13.

It should be noted that for most of the schemes, M is a multiple of the length of MD which is 128*$\mid P \mid$, whereas our scheme uses $M = 64* \mid P \mid$ independent of the length of the MD. Therefore, for longer MD's our algorithm proves to be more efficient than the present schemes. In practical implementation, our algorithm generates message digest in 0.016 seconds for an input message of size 512 bytes and 0.027 seconds for an input message of size 14 KB.

4.5 Performance Analysis

The Bernoulli keyed hash function is analyzed for sensitivity, confusion and diffusion properties and collision resistance is discussed below.

4.5.1 Illustrating each step of proposed method

Illustrating all the steps of Bernoulli keyed hash function on a single character as input message.

x_0=0.345671

λ=4

Plaintext - *a*

Binary representation of plaintext - 01100001

Padding -

01100001 10000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000001

Diffusion -

01000000 00000000 10000000 00000000 10000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 10000000 00000001

Permutation -

00000000 00000000 00000000 00000000 10000000 00000000
00000000 00000000 10000000 00000000 10000000 00000001
01000000 00000000 00000000 00000000

Final $x_0's$:

x_{B_0}=0.566510911103551

x_{B_1}=0.402966248470744

x_{B_2}=0.667395033595001

x_{B_3}=0.323547448404084

x_{B_4}=0.990343786178691

x_{B_5}=0.40255349801499

x_{B_6}=0.907348667317384

x_{B_7}=0.0855904161720338

Hash value obtained from each block:

$h0$=1100000001110011

$h1$=0100100000110010

$h2$=0000111010011110

$h3$=1111010100001001

$h4$=1100000110110010

$h5$=1101101111111111

$h6$=1100010100000101

$h7$=0100101000000111

4.5.2 Sensitivity of input message

Simulation of proposed hash function is carried out for different cases and their corresponding 128-bit hash values in hexadecimal format are given below.

Case 1: Original message
Input message:
Cryptography is the study of transforming information in order to make it secure from unintended recipients or use.
Hash value(128-bits):
1011001001011000111111011011101000111101101011101000111000000101
1011010110000101011001111101100111110001111110000000010000110111
Hash value in hexadecimal format:
B258FDBA3DAE8E05B58567D9F1F80437

Case 2: Change the letter ”i” to ”I” in the word information
Input message:
Cryptography is the study of transforming Information in order to make it secure from unintended recipients or use.
Hash value(128-bits):
0100001101110111100011101101100111001110110011010001111100100100
1111001000111010011001010101001000010011110011100010011001110100
Hash value in hexadecimal format:
43778ED9CECD1F24F23A655213CE2674

Case 3: Insert ””
Input message:
”Cryptography” is the study of transforming information in order to make it secure from unintended recipients or use.
Hash value(128-bits):
0100010001011111000111100111000001110011110100000000110011100000
1000110011010101011011110000011010011110010010011001001010111111
Hash value in hexadecimal format:
445F1E7073D00CE08CD56F069E4992BF

Case 4: Replace character ”.” by ”!” at the end of the message
Input message:
Cryptography is the study of transforming information in order to make it secure from unintended recipients or use!
Hash value(128-bits):
0100001000101100011010001011001100111010101101000100111110010011

1111011100000111101000001101101100001110000000011010110110011000
Hash value in hexadecimal format:
422C68B33AB44F93F707A0DB0E01AD98

Case 5: Replace the letter ”i” by ”1” in the word is
Input message:
Cryptography 1s the study of transforming information in order to make it secure from unintended recipients or use.
Hash value(128-bits):
0110100010010011010000001000110010111100001111001101111101000101
0110001000001101100010111011011110110010001101100111000010101101
Hash value in hexadecimal format:
6893408CBC3CDF45620D8BB7B23670AD

Case 6: Delete the letter ”i” in is
Input message:
Cryptography s the study of transforming information in order to make it secure from unintended recipients or use.
Hash value(128-bits):
1011110100001010110011010001111100010100101100110101111101010010
0000010010100101001001001001111001010011000011010100001010111000
Hash value in hexadecimal format:
BD0ACD1F14B35F5204A5249E530D42B8

The graphical display of the binary sequence of hash values is shown in Figure 4.3.

From the simulation results it is shown that tiny difference in the input message will cause a huge change in the hash value thus satisfying the sensitivity property.

4.5.3 Distribution of x_{B_i}'s obtained from each block

Here we show the distribution of $x'_{B_i}s$ over the interval (0, 1). Figure 4.4 below shows the distribution of x_{B_i}'s obtained from each block with different x_0's. Plots in Figure 4.4 show that the distribution of x_{B_i}'s follows the underlying invariant density function of logistic map over the interval (0,1) even when they are chosen from different domains.

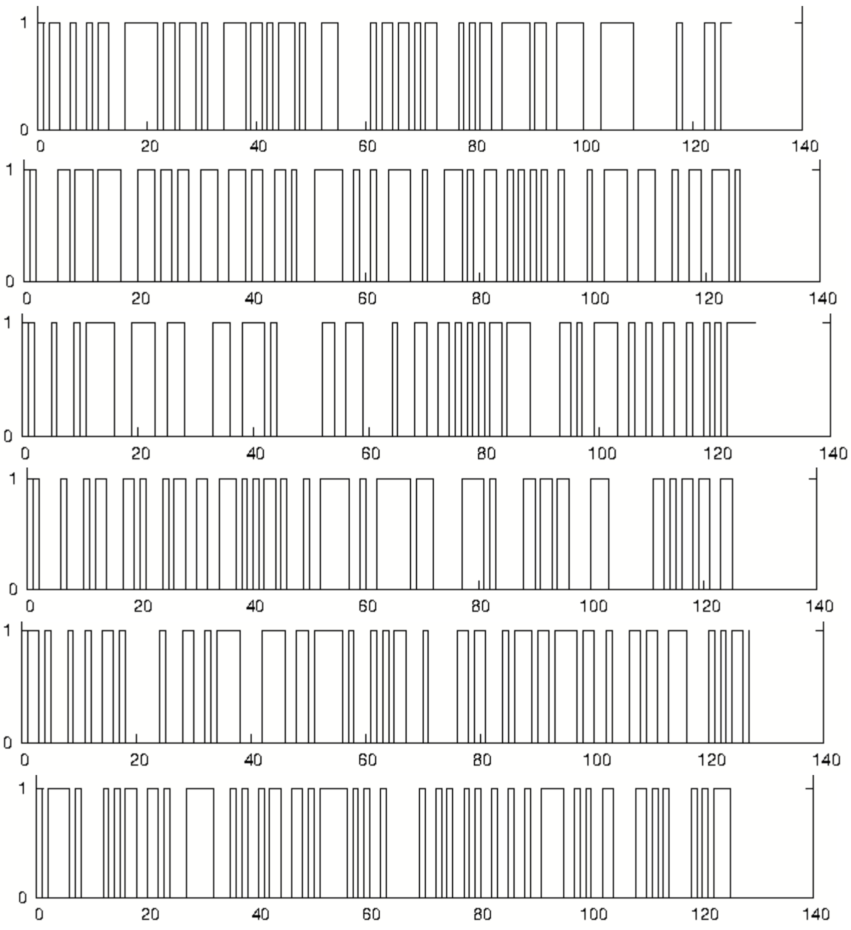


Figure 4.3: Hash values of messages with minute difference (Case 1 to 6)

4.5.4 Statistical analysis of diffusion and confusion

Shannon introduced the properties of diffusion and confusion for hiding the redundancy of a plaintext[80]. Diffusion and confusion are two important elements in the design of one-way hash function which requires the hash values be evenly distributed sensitive to the plaintext. For a hash value the ideal diffusion effect should be that any tiny change in the secret key or input message results in about 50% change of each bit in probability.

Statistical analysis is carried out according to the literature [103, 108]. Let P be the input message and P'_i denote the message in which ith bit is toggled, i is randomly chosen. Then $d_H(h(P,K), h(P'_i,K))$ denotes the number of bits changed in output with a 1 bit change in the input. The minimum hamming distance and maximum hamming distance are computed.

$$d_{min} = \min_{1\leq i\leq N} d_H(h(P,K), h(P_{i'},K)),$$

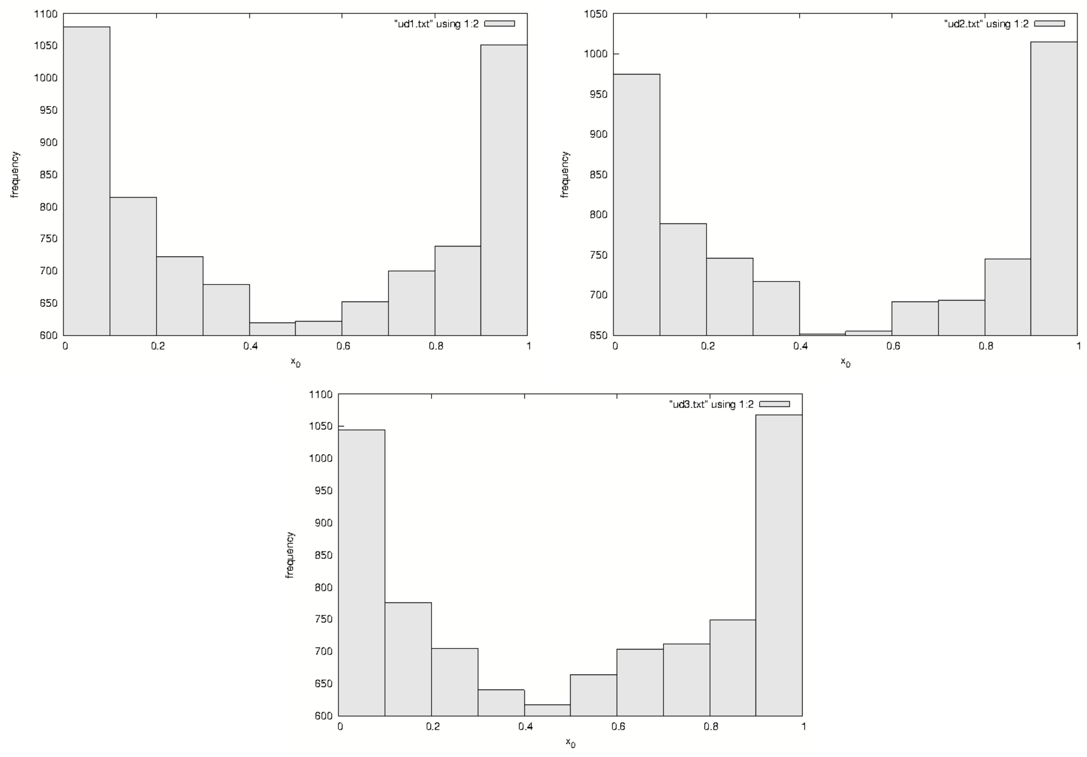


Figure 4.4: Choosing x_0 in (0, 0.33), (0.33, 0.66) and (0.66, 1)

$$d_{max} = \max_{1 \leq i \leq N} d_H(h(P, K), h(P_{i'}, K)).$$

Further, the average d_{avg} and the standard deviation d_{std} of the distribution of hamming distances over N trials are computed. We tabulate the performance of the hash function in terms of four hamming measures.

$h(P, K)$ is denoted as the keyed hash function with keys K which acts on any plain text P to produce the hash value h. The analysis of a keyed hash function is done in two steps. First, fixing the key space, we analyze the hash function for sensitivity of input, preimage and collision resistance. Subsequently, the key space is tested for sensitivity.

Statistical analysis of confusion and diffusion for small input messages

We show that even very small messages of length from 1 byte onwards, on flipping a bit in the input message produce a significantly different message digest which is noted by the hamming distance d_{avg} in the Table 4.1 which is seen to be approximately of the order 65-bits obtained by averaging over the number of runs. In each run a different bit is flipped and the comprehensive statistical performance of the hash function for variable length input messages is shown in the Table 4.1.

Message Length (bytes)	Number of runs	d_{min}	d_{max}	d_{avg}	d_{std}
1	4	57	73	65.5	7.33
2	8	55	76	67.6	6.3
4	16	60	71	66.13	3.2
8	32	57	78	66.32	4.56
16	64	52	76	65.08	5.2
32	128	52	76	65.04	5.2
64	256	51	77	65.14	5.74
128	512	49	80	65.57	6.01
256	1024	51	80	66.66	4.79
512	2048	47	84	66.4	5.7

Table 4.1: Statistical analysis of changed bit number for small length messages

Table 4.1 shows that the capability of diffusion and confusion is stable and thus resistant to statistical attacks.

Statistical analysis of confusion and diffusion in 1-bit neighbourhood of a large input message

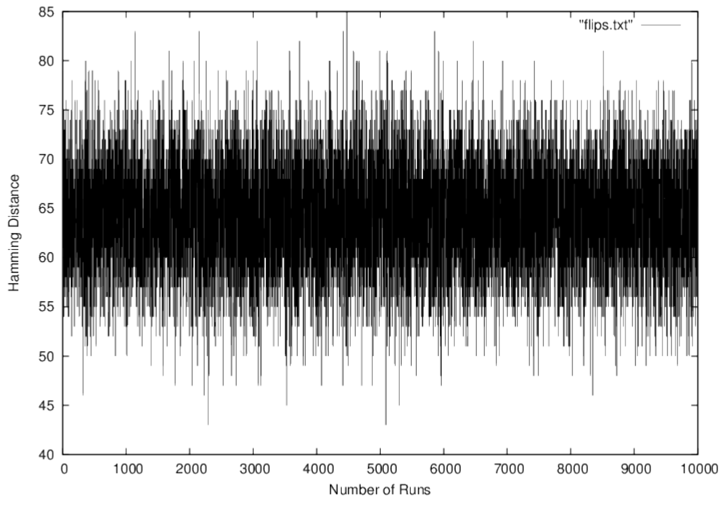
Now a large input text is considered of length around 3 million characters. In each run, a single bit is flipped randomly, and such experiments are carried for 10,000 times. The hamming distance of the hash value obtained from original input data and the modified input data is shown in Table 4.2. Clearly, the keyed hash function based on logistic map based compression followed by Bernoulli shift map is seen to exhibit a confusion of more than 64 bits on average within a 128 bit message digest which is a significant improvement on the results obtained by Zhang et al. [108] whose average results show less than 64 bits which is below the ideal 50% confusion. Figure 4.5 shows the distribution of hamming distance for the above experiment which is seen to concentrate around the ideal value of 50% probability. The best results obtained in the literature are those of Yang et al.[103] who obtain d_{avg} of 64.06 for their chaotic hash function (not keyed).

Figure 4.5 shows that the number of bits changed for 1-bit toggled message concentrates around the ideal value 64-bits indicating the capability of diffusion and confusion is stable and strong for the proposed function.

Table 4.3 gives the statistical analysis of confusion and diffusion of each block. 16-bits hash value h_i obtained from each block is analyzed according to statistical analysis carried out in the literature[102, 103, 108]. If d_{min} is 0 then h_i of original

N	256	512	1024	2048	4096	8192	10000	10000 (Zhang[108])
d_{min}	49	47	47	43	43	43	**43**	44
d_{max}	82	82	85	85	85	85	**85**	85
d_{avg}	**64.26**	**64.23**	**64.22**	**64.34**	**64.27**	**64.21**	**64.18**	63.96
d_{std}	5.97	5.85	5.84	5.84	5.72	5.67	**5.75**	4.32

Table 4.2: Statistical analysis of changed bit number for a large plain text with 1 bit toggled randomly with number of trials N for a maximum of N=10,000



message and the 1-bit toggled message is the same indicating collision at block level. For the proposed function d_{min} is 1 or 2 indicating that collision do not happen at block level and d_{max} value 16 indicates that capability of diffusion and confusion is maximum. The d_{avg} values obtained from the experiment can be seen to concentrate on the ideal value 8-bits of h_i. These results indicate that the proposed algorithm has strong and stable capabilities of confusion and diffusion even at the block level.

In the literature, it is usual to conduct a bit flip analysis. But the following extensive analysis at byte level, block level as well as a whole space level has not been reported for any hashing scheme in the literature to the best of our knowledge.

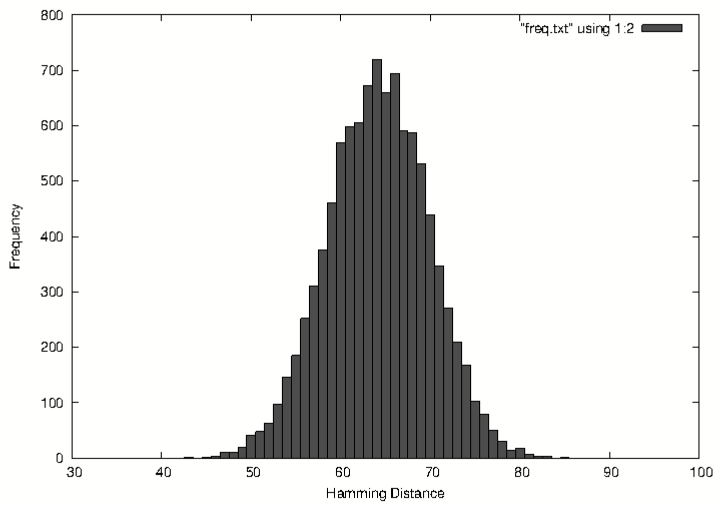


Figure 4.5: Distribution of the hamming distance for keyed hash function

	h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7
d_{min}	2	1	1	1	1	2	1	1
d_{max}	15	15	15	16	16	15	16	16
d_{avg}	8.01	8.07	8.09	7.93	8.01	8.01	8.02	8.00
d_{std}	1.93	1.97	1.96	1.97	1.97	1.98	1.98	1.97

Table 4.3: Statistical analysis of changed bit number for each block in 1-bit neighbourhood of a large input message

Statistical analysis of confusion and diffusion in one-byte neighbourhood of large input message

The input is the same as the large plain text chosen in section 4.5.4. The message is altered by changing a randomly chosen character in the input and this experiment is repeated for 10000 times. The hamming distance of the hash values of the original input data from the modified input data are shown in Table 4.4.

Table 4.4 shows d_{avg} is around the ideal value 64-bits. This indicates the stability of confusion and diffusion capabilities of the proposed keyed hash function and also its resistant to statistical attacks.

Statistical analysis of confusion and diffusion for each block by one-byte flip in the input message

Consider the large plain text chosen in section 4.5.4. Just by altering one character in the plain text, we study how this change percolates to hash value of each block.

N	256	512	1024	2048	4096	8192	10000
d_{min}	49	46	46	46	43	43	43
d_{max}	79	80	80	83	83	85	85
d_{avg}	64.27	64.09	64.12	64.26	64.17	64.21	64.18
d_{std}	5.97	5.67	5.49	5.48	5.70	5.67	5.64

Table 4.4: Statistical analysis of confusion and diffusion for large text at the level of character. Changing a character in the plain text and assess the associated change in the hash value with the results tabulated for the experiments carried out N times

Change a randomly chosen character in the input and repeat this experiment for 10000 times. Let h_i be the 16-bit hash value obtained at each block from original input data, and h'_i be the hash value obtained at each block from the modified input data. The distribution of hamming distance between h_i and h'_i is analyzed in Table 4.5.

	h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7
d_{min}	2	1	2	1	2	2	1	2
d_{max}	15	15	15	16	16	15	16	16
d_{avg}	8.01	8.08	8.1	7.93	7.99	8.01	8.03	8.00
d_{std}	1.93	1.97	1.95	1.97	1.97	1.97	1.98	1.97

Table 4.5: Statistical analysis of confusion and diffusion analysis for each block by one-byte flip in the input message

Note that $d_{min} > 0$, which means no collisions even at the block level and $d_{avg} > 7$, clearly showing superior collision resistance performance. The d_{avg} value obtained is close to ideal value 8-bits of h_i. These results indicate that the proposed keyed hash function has strong capabilities of confusion and diffusion even at each block.

4.5.5 Collision analysis in the whole space

We have taken 3,194,807 different input messages which is roughly around 2^{21} and observed that all the message digests are collision free. In order to prove resistance to birthday attack, we need to run this experiment for 2^{64} messages. Table 4.6 shows the hash values obtained for a sample of ten input messages. Since a collision resistant hash function is automatically second preimage resistant [22], we do not provide any additional argument for second preimage resistance.

We have done collision analysis test by taking 65,536 i.e, 2^{16} different messages

Input	Message Digest
P1	A70B03DAF146D449E5B0644DA69C79A7
P2	9EB8C53875EE00003DD78B844ACB24C6
P3	2A333B4D316ADFDDFF874AD1EF6B8AE0
P4	C715D4A2F4ACFD81019774DFD2072EAE
P5	80F2209AFFFFAA40EB2755BD5EB76B6C
P6	1915FA2B2061274E3868B66B14500CD8
P7	2839FF120AEE76FEC494E68B5CD2412B
P8	9662D4F716EDF7E5B793293C87E82477
P9	D580930E85C2FC7F717D18AB7DC681D7
P10	E0E5530CFD4B1AB8150EF901869E15AC

Table 4.6: MD generated for the input messages $P_1, P_2, P_3, \cdots P_{10}$ using keyed hash function

with length of MD to be of 32-bits, i.e, 4-bits are generated by $HashBernoulli$ Algorithm 5 for each block. If the size of the hash value is less then probability of finding a collision is high. We have found first collision at 57,014. From this experiment we say that the hash value size must be at least 128-bits. Table 4.7 shows the hash values obtained for a sample of ten input messages.

Input	Message Digest(32-bits)
P1	10011111110011101111111000000001
P2	00001101010010101010101100101111
P3	10110100011100110001111111010111
P4	00110101011101100111111011110000
P5	11110110110000100100111110100001
P6	00100100101010100010000000100000
P7	11111011110011100101001101000110
P8	00010111100011110000110111100001
P9	00000011111000111110000010000000
P10	01010101100111010100011010001011

Table 4.7: MD of length 32-bits generated for the input messages $P_1, P_2, P_3, \cdots P_{10}$ using keyed hash function

4.6 Cryptanalysis

In this section, arguments for near collision and preimage resistance of the proposed keyed hash function are given along with illustrating the same examples.

4.6.1 Near collision resistance

In Algorithm 3, the step of introducing diffusion is important since transporting the bits of each character into all the eight blocks breaks the character dependency and induces randomness. Perturbing x_{B_i} with x_{B_7} obtained from compression function also induces randomness. These two steps helps in avoiding near collisions. For elucidation purposes, we run the algorithm on a small example of two strings both identical except in the last character .

Example: Input messages: M1: diffusion and M2: diffusioa
Now the seeds generated per block are shown in the Table 4.8. Hamming distance between the hash values obtained for the two messages M1 and M2 is 66-bits.

	M1	M2	HammingDistance(M1,M2)
Input Message	diffusion	diffusioa	
x_{B_0} h_0	0.300515286219458 0010101111100110	0.341837620904593 0001010110001010	9
x_{B_1} h_1	0.226854645819571 1011111000110100	0.391716960526644 0010100100011100	7
x_{B_2} h_2	0.159125325268756 0110001101011111	0.158485545885274 0101010011100100	11
x_{B_3} h_3	0.997162277736213 1000100110111010	0.936393259069408 1110100010111011	4
x_{B_4} h_4	0.548602918771423 0011011010010101	0.926948492078734 0011110011011000	6
x_{B_5} h_5	0.41843677247604 1110110001001111	0.601108508805559 1001001111011001	11
x_{B_6} h_6	0.798239135920256 1101011100111000	0.93005804369404 0110101111111110	9
x_{B_7} h_7	0.95355843394418 1110001100111110	0.502971111056028 0001010110110111	9
Message Digest	2BE6BE34635F89BA 3695EC4FD738E33E	158A291C54E4E8BB 3CD893D96BFE15B7	66

Table 4.8: Hash values for two strings identical except in the last character. x_{B_i}'s and their corresponding h_i's for each block are shown. The last row gives the message digests of M1 and M2 which are seen to differ in 66-bits.

4.6.2 Preimage resistance

Given a hash value y for the proposed hash function, it is to be shown that it is computationally infeasible to produce an input (preimage) x such that $h(x) = y$. A clear vulnerability is found if the first 16 bits of the message digest y correspond to x_{B_0} of the first block. Therefore, the x_{B_0} is further perturbed with x_{B_7}. Then we leave some iterations of Bernoulli shift map before generating 16-bits. How close is the hash value thus obtained to the actual message digest? Let the decimal equivalent of h_0 be equal to x'_{B_0}. Clearly x'_{B_0} may be close to x_{B_0}. The whole algorithm can then be run using x'_{B_0} in the place of x_{B_0} from second block onwards for computing the remaining 112-bits of the hash value.

Thus it is important to see if guessing x_{B_0} will anyway jeopardize the security of the hash function. We conduct the sensitivity experiment on several different plaintexts by taking $xB_0{}'$ in the place of xB_0 to assess the impact of this attack. A sample of the results is shown in the Table 4.9. It is observed that the property of sensitivity to initial conditions of the chaotic compression function makes even a small error in the initial value generates a completely different message digest. Table 4.9 shows that a preimage attack is not feasible if the initial value x_0 is guessed approximately, with the hamming distance of the message digest obtained being on average 56.2 bits within 112 bits of the message digest thus ensuring the security of the scheme.

Input	Hamming Distance (112 bits)
P1	58
P2	60
P3	59
P4	57
P5	51
P6	55
P7	50
P8	63
P9	53
P10	61

Table 4.9: Preimage attack on the proposed keyed hash function.

It is well known that hash functions which satisfy Merkle-Damgard construction are collision resistant if the compression block is collision resistant [82]. For chaotic functions, we claim that if the parameters of λ and x_0 are appropriately chosen, then the compression block and hash function will be collision resistant. We validate this claim by experimental analysis.

4.7 Security of Key Space

Now in the following experiments we focus our energies on analyzing security of the key space. Firstly we check for the dependence of the hash function on the keys and empirically show that different keys do give rise to different message digests. Then we analyze for the sensitivity of the keys by perturbing the keys a little which is shown to produce a dramatic change in the MD due to the properties of chaotic maps.

4.7.1 Varying x_0 in the whole space

Fix the value of λ. The other parameter x_0 is randomly chosen from the interval (0, 1) and hash value of the keyed hash function is computed. This experiment is repeated for 10000 times with different x'_0. The hamming distance $d_H(h(P, K), h(P, K'))$ is plotted in the Figure 4.6 below. The experiments clearly show that h has strong diffusion and confusion capabilities and the average hamming distance is concentrated around the ideal 64 bits.

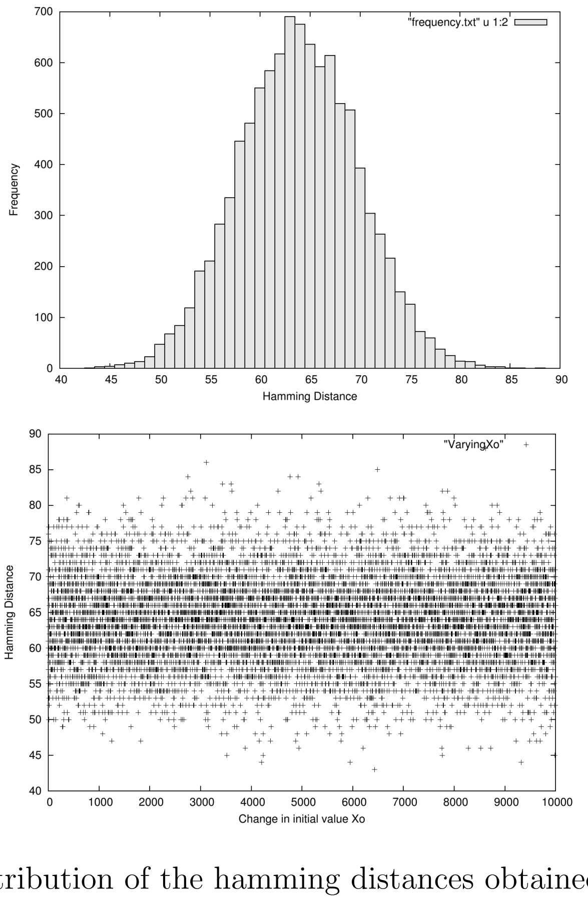


Figure 4.6: Distribution of the hamming distances obtained by varying x_0

4.7.2 Varying λ in the whole space

The key parameter λ is randomly chosen in the interval (3.8, 4) and hash value of the keyed hash function is computed. This experiment is repeated for 10000 times with different λ' for a fixed x_0. The Hamming distance $d_H(h(P,K), h(P,K'))$ is plotted in the Figure 4.7 below. Plots show that average hamming distance is concentrated around the ideal 64 bits.

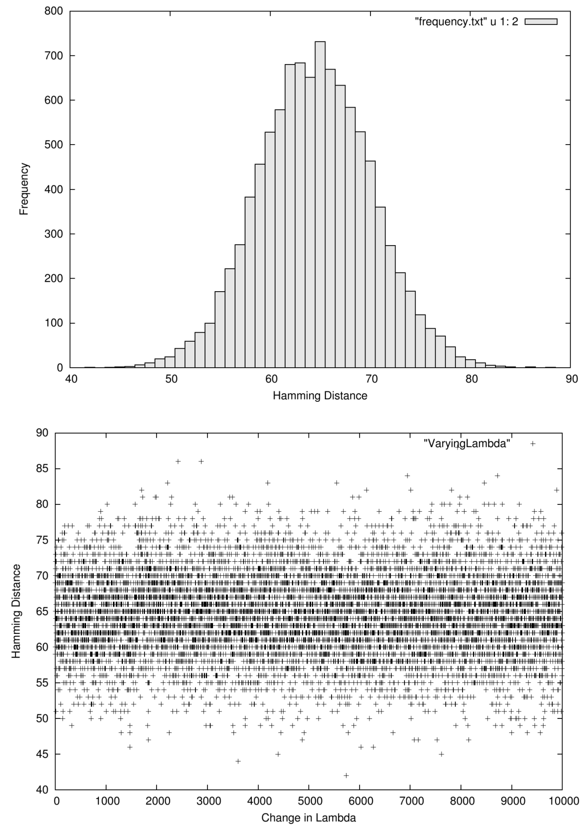


Figure 4.7: Distribution of the hamming distances obtained by varying λ randomly in the interval (3.8, 4)

4.8 Sensitivity to Keys

We try to satisfy the requirement as specified by S. Li [55] in Rule 6 and we show the avalanche effect between two hash values obtained for two slightly different keys K and K'.

4.8.1 Sensitivity of x_0

The key parameter x_0 is changed with negligible difference to, say x_0' and hash value of the keyed hash function is computed. A sample of results of the hamming distance $d_H(h(P,K),h(P,K'))$ is presented in Table 4.10.

x_0'	$d_H(h(P,K),h(P,K'))$
$x_0 + 10^{-06}$	66
$x_0 + 10^{-07}$	61
$x_0 + 10^{-08}$	64
$x_0 + 10^{-09}$	57
$x_0 + 10^{-10}$	68
$x_0 + 10^{-11}$	69
$x_0 + 10^{-12}$	54
$x_0 + 10^{-13}$	72
$x_0 + 10^{-14}$	62
$x_0 + 10^{-15}$	62

Table 4.10: Hamming distance for different x_0 values very close to $x_0 = 0.345671$.

4.8.2 Sensitivity of λ

The key parameter λ is infinitesimally varied to obtain $K' = (x_0, \lambda')$ and hash values of the keyed hash function are computed with K and different K'. The sample results tabulated in Table 4.11 show the average hamming distance between these corresponding hash values.

λ'	$d_H(h(P,K),h(P,K'))$
$\lambda + 10^{-06}$	63
$\lambda + 10^{-07}$	74
$\lambda + 10^{-08}$	68
$\lambda + 10^{-09}$	62
$\lambda + 10^{-10}$	63
$\lambda + 10^{-11}$	59
$\lambda + 10^{-12}$	71
$\lambda + 10^{-13}$	70
$\lambda + 10^{-14}$	49
$\lambda + 10^{-15}$	60

Table 4.11: Hamming distance corresponding to different perturbations of λ values for $\lambda = 3.897653$

4.9 Comparative Study

Table 4.12 shows the comparative study of the statistical analysis of the hash functions in the literature. Proposed Bernoulli keyed hash function is compared with chaotic hash functions[102],[108] and keyed hash functions[1, 101] and traditional hash functions MD5 and SHA1. Most of the chaotic schemes use piecewise linear chaotic maps for their compression schemes and binary representation for bit generation. Proposed function use only one chaotic function i.e, logistic map in compression scheme and Bernoulli shift map for bit generation. d_{avg} is considerably high for the proposed function.

Scheme	**Chaotic map used**	**Bit generation**	**Length of MD(bits)**	d_{avg}
Baptista hash function3.5	Logistic map and tent map	Binary representation	128	63.56
Bernoulli keyed hash function4.4	Logistic map	Bernoulli shift map	128	**64.18**
Ref.[102]	Piecewise linear map	Binary representation	128	64.06
Ref.[108]	Piecewise linear map	Quantization function	128	63.98
Ref.[101]	Piecewise linear map	Binary representation	128	64.01
Ref.[1]	Piecewise non-linear map	Binary representation	128	63.92
MD5	–	–	128	64.03
SHA1	–	–	160	79.86

Table 4.12: Comparison of the proposed hash functions with a few sehemes from the literature for their statistical analysis

4.10 Adapting the hash function to longer MD

The algorithm can be adapted to compress a message to hash value of bigger lengths with slight modifications. Each of the blocks can generate upto 32 bits after applying the shift map by ignoring about 10 bits. Hence, the current design requires that the number of blocks is to be chosen such that $\lfloor length(MD) \rfloor < 32$. Therefore, 8 blocks suffice for generating a 160 bit or 256 bit message digest. But for longer message digests of lengths 512 or 1024 bits, the message may have to be divided into 32 blocks which would mean that in the initial processing step the length of the message needs to be padded such that it is a multiple of $32 \times 8 = 256$.

4.11 Conclusions

The literature is abound with hash functions based on chaotic networks. There tends to be ad-hocism creeping into the schemes and the reasons for their strength become less apparent. The main contribution of this chapter is that a generic design for constructing a secure hash function is proposed and implemented to make the whole construction transparent. A preprocessing step of mixing the input bits is carried out. Then, the well-studied logistic map is used for compressing each block of data into a real number and the chaotic Bernoulli shift map is utilized to generate the message digest. We show that using Bernoulli map in this novel way achieves high amount of mixing and thus ensures collision resistance. Extensive validation is carried out at byte, block and the whole message level for collision resistance and the proposed keyed hash function is shown to achieve higher security than many recent schemes proposed in the literature. Performance and security of these two hash functions is reported in the results.

Next chapter presents an application of these two proposed hash functions to construct integrity check value.

CHAPTER 5

Application : Integrity Check Value

5.1 Introduction

We use the hash functions proposed (Baptista and Bernoulli hash functions) in the previous chapters 3 and 4 to construct an integrity check value. A keyed hash function or message authentication code(MAC) is a family $h_k : \ k \in K$ of hash functions, where K is a key space of h [19]. A hash function is a function that takes a variable-length input string and converts it to a fixed-length(smaller) output string called hash value or message digest. $h : (0,1)^\star \rightarrow (0,1)^n$ is defined such that h satisfies the three security properties: **collision resistance**, **preimage** and **second preimage resistance** [82].

5.1.1 Definition

Integrity check value(ICV) is an authentication field in IPSec Authentication Header(AH) and Encapsulating security payload(ESP) as shown in Figure 5.1.

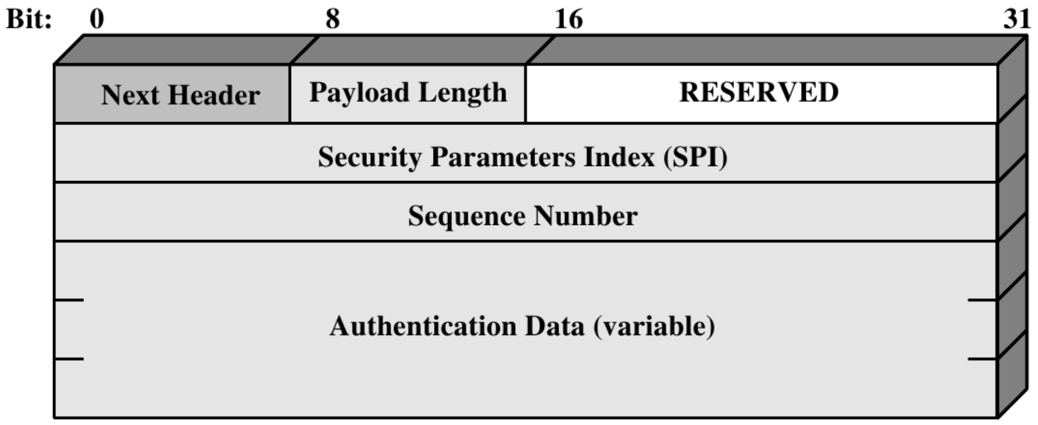

Bit:
0
8
16
31
Next Header
Payload Length
RESERVED
Security Parameters Index (SPI)
Sequence Number
Authentication Data (variable)

Figure 5.1: Authentication Header.

The ICV is a truncated version of message authentication code(MAC) produced by MAC algorithm HMAC-MD5-96(RFC2403)[64] or HMAC-SHA1-96(RFC2404)[65].

The term ICV is referred as Keyed Hash function or Message Authentication Code. The full HMAC value is calculated and first 96 bits are considered, which is the default length for the authentication data field in IPSec Authentication Header and Encapsulating security payload. The truncated hash value serves the purpose of reduced bandwidth.

HMAC is a secret key authentication algorithm. HMAC provides data origin authentication and data integrity. If only the sender and receiver know the HMAC key, this provides both data integrity and data origin authentication for packets sent between the two parties; if the HMAC is correct, this proves that it must have been added by the source[64]. HMAC-MD5-96 is used in ESP and AH. HMAC-MD5-96 produces a 128-bit authenticator value. This 128-bit value can be truncated as described in RFC 2104[48]. For use with either ESP or AH, a truncated value using the first 96 bits must be supported. While sending, the truncated value is stored within the authenticator field. After receiving, the entire 128-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field[64].

It is well-known that keyed hash functions are directly useful to be used as MAC. Chaotic keyed hash functions have been proposed in the literature by Zhang et al. based on nth-order chaotic system [108] and by Xiao et al., [101] based on chaotic neural network. These algorithms then can be theoretically used as integrity check value algorithms. There is no explicit evaluation of chaotic hash function algorithms to be used as integrity check value algorithms in the literature.

Initial value x_0 and control parameter λ are taken as secret key parameters. We consider the first 96 bits of the hash value as integrity check value and conduct a systematic analysis to prove its strength. Considering only first 96-bits of the hash value may increase the probability of collision[48]. We assess how the truncation of a MD of length 128 to say 96 bits influences its performance without lowering security and report our results.

5.2 Analysis of Collision Resistance

In this section, collision resistance analysis for a tiny change in the input message for the proposed hash functions(3.5 and 4.4) is carried out for the first 96-bits of the hash value. This properties have to satisfy these conditions to qualify them as robust keyed hash functions.

5.2.1 Collision resistance analysis for ICV of Baptista hash function

We perform quantitative analysis on collision resistance according to Wong's method [103], by calculating the number of ASCII characters with the same value at the same location using W in the formula 5.1 in the hash value whose input messages differ by one bit. ASCII character of the original and the new hash value are compared and the collision test performed for 10,000 times.

$$W = \sum_{i=1}^{N} f(t(e_i), t(e_i{}')) \tag{5.1}$$

where

$$f(x, y) = \begin{cases} 1 & \text{x=y} \\ 0 & \text{x} \neq \text{y} \end{cases}$$

e_i and e_i' are the i^{th} ASCII character of the original and the new hash value, respectively. We found in the section 3.6.3 that W is distributed as $W(0) = 9325$, $W(1) = 649$, $W(2) = 25$, $W(3) = 1$ and $W(k) = 0$ for $k > 3$ for a 128-bit hash value.

Now the hash value is truncated to 96-bits, i.e, first 96-bits of the hash value are considered. The number of ASCII characters with the same value at the same location using the formula 5.1 for the first 96-bits of the hash values whose input messages differ by one bit is calculated. The experimental values of W are obtained as $W(0) = 9464$, $W(1) = 512$, $W(2) = 23$, $W(3) = 1$ and $W(k) = 0$ for $k > 3$ for a 96-bit hash value.

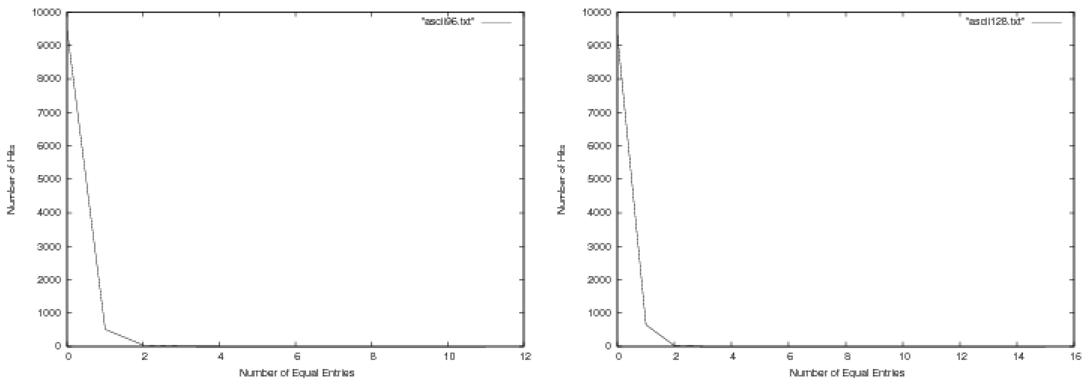


Figure 5.2: Distribution of the number of ASCII characters with the same value at the same location in the first 96-bits of the hash value and 128-bit hash value for Baptista hash function.

Figure 5.2 shows the distribution of the number of ASCII characters with the same value at the same location in the hash value of first 96-bits of the hash value and 128-bit hash value. This shows that the proposed function is performing well with the maximum number of equal characters in hash values obtained being only 3 and the collision probability being quite low in spite of the truncation of the message digest.

5.2.2 Collision resistance analysis for ICV of Bernoulli keyed hash function

Experiment that is carried out in 5.2.1 is repeated for Bernoulli keyed hash function. The experimental values of W are distributed as $W(0) = 9405$, $W(1) = 577$, $W(2) = 18$ and $W(k) = 0$ for $k > 3$ for a 128-bit hash value.

Now the hash value is truncated to 96-bits, i.e, first 96-bits of the hash value are considered. The number of ASCII characters with the same value at the same location for the first 96-bits of the hash values whose input messages differ by one bit is calculated. The experimental values of W obtained here are $W(0) = 9433$, $W(1) = 552$, $W(2) = 15$ and $W(k) = 0$ for $k > 3$ for a 96-bit hash value.

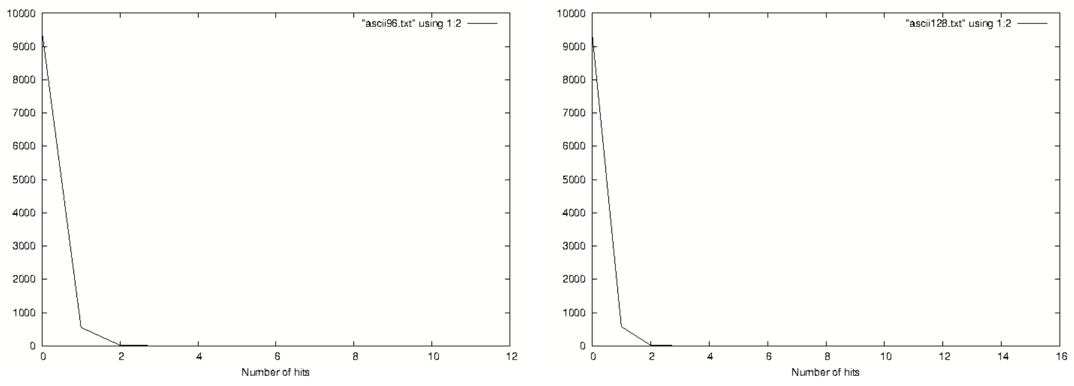


Figure 5.3: Distribution of the number of ASCII characters with the same value at the same location in the first 96-bits of the hash value and 128-bit hash value for the Bernoulli keyed hash function.

Figure 5.3 shows the distribution of the number of ASCII characters with the same value at the same location in the hash value of first 96-bits of the hash value and 128-bit hash value. This shows that the proposed function is performing well with the maximum number of equal characters in hash values obtained being only 2 and the collision probability being quite low.

5.3 Comparison of the Proposed Hash Functions with MD5 and SHA1

Experiment in which the number of identical bytes that occur in the same positions in hash values is calculated i.e collision analysis for the proposed Baptista hash function, Bernoulli keyed hash function, MD5 and SHA1. Hash values generated by toggling a bit randomly 2048 times are stored in ASCII. Table 5.1 shows the frequency of the number of ASCII characters with the same value at the same location of proposed functions, MD5 and SHA1. The maximum number of positions matched is 3 for Baptista hash function and remaining positions after 3 are all 0's. Table 5.1 shows that the proposed functions are on par with traditional hash functions MD5 and SHA1 collision probability is quite low.

$No.of positions$ $matched$	Frequency (Baptista hash function)	Frequency (Bernoulli keyed hash function)	Frequency (MD5)	Frequency (SHA1)
0	1927	1949	1948	1975
1	82	79	73	56
2	39	20	27	17
3	2	0	0	0

Table 5.1: Frequency of the number of ASCII characters with the same value at the same location of proposed hash functions, MD5 and SHA1.

5.4 Sensitivity of Control Parameter

In this section statistical analysis of diffusion and confusion and collision resistance analysis is carried out with a tiny change in the control parameter.

5.4.1 Sensitivity analysis of control parameter for Baptista ICV and Bernoulli ICV

Here we analyze the strength of the proposed hash function with respect to control parameter. x_0 is not altered. Let $h(P)$ be the original hash value and $h(P')$ be the hash value obtained by varying the control parameter from $\lambda + 10^{-7}$ to $\lambda + (2048 * 10^{-7})$. Then $d_H(h(P), h(P'))$ denotes the number of bits changed in output with a change in the control parameter. The d_{avg} over 2048 trials is 63.51.

When the first 96-bits for Baptista ICV are considered d_{avg} is 47.34 which is very close to the ideal value and for Bernoulli ICV d_{avg} is 48.03 which is equal to the ideal value of 50% probability. Table 5.2 and 5.3 depicts the space in which the tests are conducted is large enough to indicate that the values obtained by d_{avg} etc lie close to the true values of the distribution.

N	256	512	1024	2048
d_{min}	36	36	36	36
d_{max}	60	60	63	63
d_{avg}	47.64	47.53	47.43	47.34
d_{std}	4.6	4.65	4.62	4.64

Table 5.2: Statistical analysis of diffusion and confusion of first 96-bit of 128-bit hash value $h(P')$ obtained by varying the control parameter from $\lambda + 10^{-7}$ to $\lambda + (2048 * 10^{-7})$, where $\lambda = 3.87$, generating 2048 trials for Baptista ICV.

N	256	512	1024	2048
d_{min}	38	39	39	39
d_{max}	65	68	68	69
d_{avg}	47.93	48.98	48.01	48.03
d_{std}	4.43	4.5	4.49	4.51

Table 5.3: Statistical analysis of diffusion and confusion of first 96-bit of 128-bit hash value $h(P')$ obtained by changing the control parameter from $\lambda + 10^{-7}$ to $\lambda + (2048 * 10^{-7})$, where $\lambda = 3.897653$, generating 2048 trials for Bernoulli ICV.

Table 5.2 shows that d_{avg} is close to 48-bits and Table 5.3 shows that d_{avg} is equal to 48-bits which is the ideal value of 50% probability. This also indicates the proposed hash functions possess strong capabilities of diffusion and confusion and is resistant to statistical attacks even when first 96-bits of the hash value is considered.

ASCII character of the original hash value $h(P)$ and the new hash value $h(P')$ obtained by changing the control parameter λ are compared on the collision test performed for 2048 times. The distribution of the number of ASCII characters with the same value at the same location in the hash value is shown in Figures 5.4 and 5.5.

Figures 5.4 and 5.5 shows the collision probability of proposed hash functions when first 96-bits are considered is 2 which is quite low.

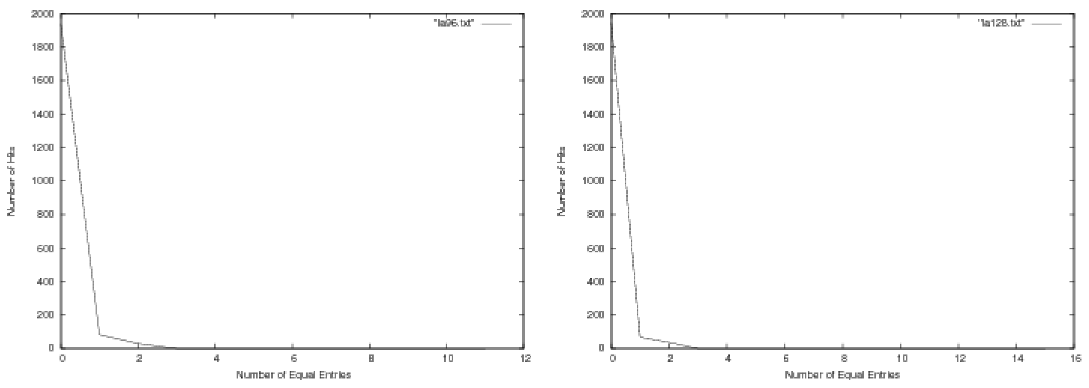


Figure 5.4: Distribution of the number of ASCII characters with the same value at the same location in the first 96-bits of the hash value and 128-bit hash value as the control parameter λ is varied for Baptista ICV.

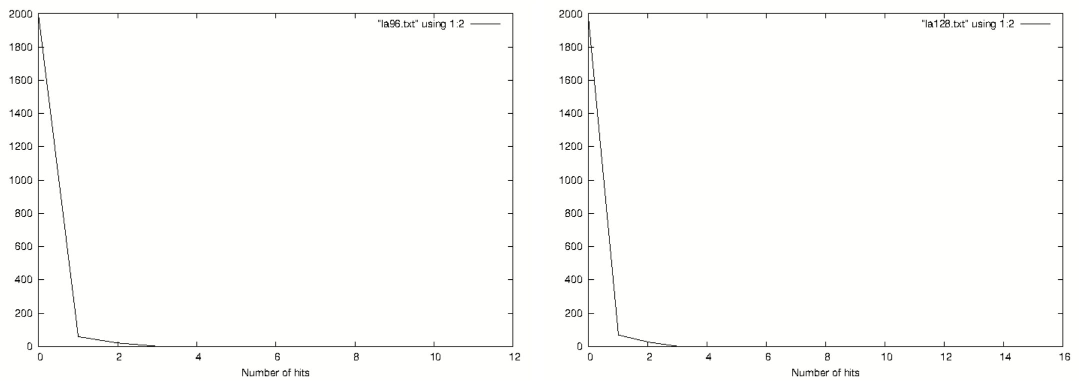


Figure 5.5: Distribution of the number of ASCII characters with the same value at the same location in the first 96-bits of the hash value and 128-bit hash value as the control parameter λ is varied for Bernoulli ICV.

5.5 Sensitivity of Initial Value

Statistical analysis of diffusion and confusion and collision resistance analysis is carried out with a tiny change in the initial value .

5.5.1 Sensitivity analysis of initial value for Baptista ICV and Bernoulli ICV

Here we analyze the strength of the proposed hash function with respect to initial value. λ is not altered. Let $h(P)$ be the original hash value and $h(P')$ be the hash value obtained by varying the initial value from $x_0 + 10^{-7}$ to $x_0 + (2048 * 10^{-7})$. Then $d_H(h(P), h(P'))$ denotes the number of bits changed in output with a change in x_0. When the first 96-bits of 128-bit hash value is considered d_{avg} is 47.81 for Baptista ICV which is very close to the ideal value and 48.07 for Bernoulli ICV which is equal to the ideal value of 50% probability. Tables 5.4 and 5.5 depicts the space in which the tests are conducted is large enough to indicate that the values obtained by d_{avg} etc lie close to the true values of the distribution.

N	256	512	1024	2048
d_{min}	35	35	35	35
d_{max}	60	60	63	64
d_{avg}	47.24	47.46	47.76	47.81
d_{std}	4.67	4.8	4.76	4.74

Table 5.4: Statistical analysis of diffusion and confusion of first 96-bit of 128-bit hash value $h(P')$ obtained by varying the initial value from $x_0 + 10^{-7}$ to $x_0 + (2048 * 10^{-7})$ for Baptista ICV.

N	256	512	1024	2048
d_{min}	38	38	39	39
d_{max}	65	67	68	69
d_{avg}	47.97	48.81	48.03	48.07
d_{std}	4.42	4.5	4.49	4.51

Table 5.5: Statistical analysis of diffusion and confusion of first 96-bit of 128-bit hash value $h(P')$ obtained by varying the initial value x_0 is varied from $x_0 + 10^{-7}$ to $x_0 + (2048 * 10^{-7})$ for Bernoulli ICV.

Values of d_{avg} in Tables 5.4 and 5.5 indicate that the proposed hash functions possess strong capabilities of diffusion and confusion and is resistant to statistical attacks even when first 96-bits of the hash value is considered.

ASCII character of the original hash value $h(P)$ and the new hash value $h(P')$ obtained by changing the initial value x_0 are compared on the collision test performed for 2048 times. The distribution of the number of ASCII characters with the same value at the same location in the hash value is shown in Figures 5.6 and 5.7.

Figure 5.5 shows the collision probability of proposed hash functions is 2 which is quite low when first 96-bits of the 128-bits hash value is considered .

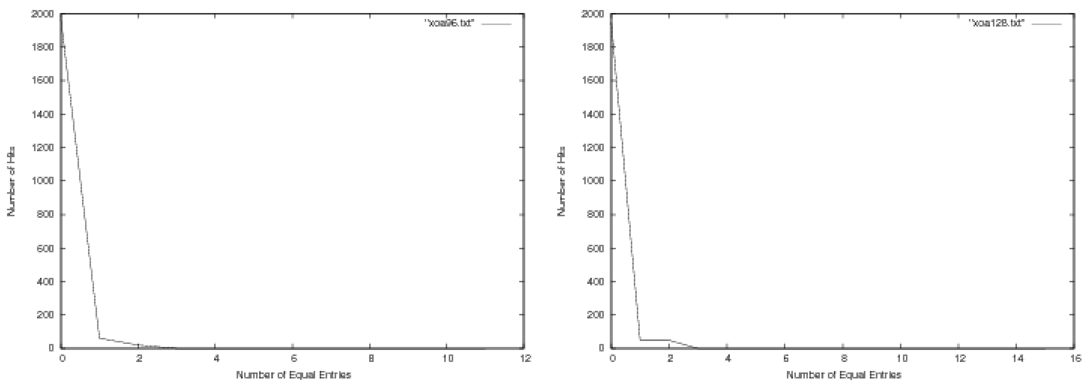


Figure 5.6: Distribution of the number of ASCII characters with the same value at the same location in the first 96-bits of the hash value and 128-bit hash value as the initial value x_0 is varied for Baptista ICV.

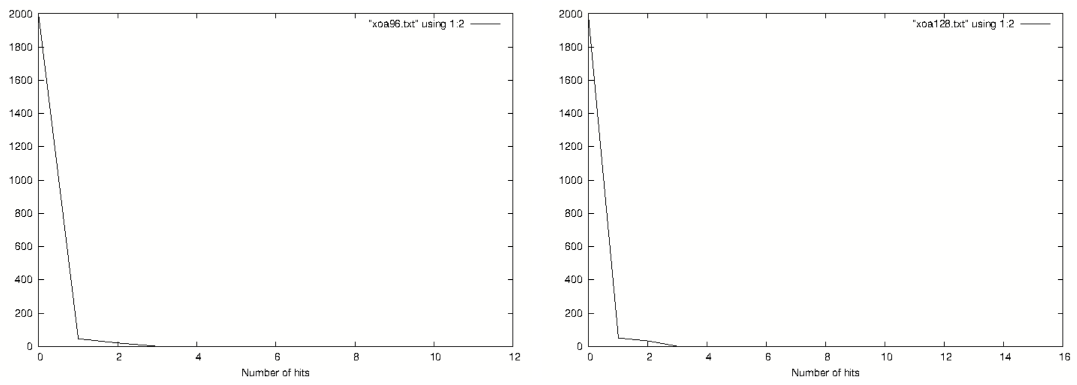


Figure 5.7: Distribution of the number of ASCII characters with the same value at the same location in the first 96-bits of the hash value and 128-bit hash value as the initial value x_0 is varied for Bernoulli ICV.

5.6 Conclusions

In this chapter we show that proposed hash functions 3.5 and 4.4 give an output of 128-bits, in which the first 96-bits of hash value can be used as an ICV. Statistical analysis of confusion and diffusion of Baptista and Bernoulli hash functions is carried out for first 96-bits of the hash value with respect to tiny change in the input message, initial value and control parameter. Results show the capabilities of confusion and diffusion is robust and secure i.e d_{avg} is close to ideal value of 50% probability. On the similar lines collision resistance analysis is carried out with respect to tiny change in the input message, initial value and control parameter and collision probability is quite low. The keyed hash function is tested by varying the two secret keys infinitesimally and shown to achieve 50% variation in the output message digest and hence is robust against collisions and statistical attacks.

CHAPTER 6

Pseudo Random Bit Generator using Logistic and Bernoulli Shift Maps

6.1 Introduction

We propose pseudo random bit generator(PRBG) algorithms by adopting the underlying ideas of Bernoulli keyed hash function in chapter 4. Further we conduct experiments for Shunjun Li's PRBG's[55] with different chaotic maps in order to generate pseudo random key streams.

6.1.1 Background

Much work has been done in skillfully using piecewise linear maps in order to produce randomness. Stojanovski et al. mathematically analyzed the application of chaotic piecewise-linear maps to produce random number generators [83, 84]. Shujun Li in his Ph.D. thesis, writes extensively upon the aspect of dynamical degradation of piece-wise linear maps and proposes dynamical indicators to measure this property[57]. Further, he proposes a novel scheme utilizing a couple of piecewise linear maps to produce a pseudo random bit generator [55]. In 2003 Kocarev et al.[42] proposed a pseudorandom bit generator using chaotic map and discussed the possibilities of using different chaotic maps as pseudorandom bit generator. The pseudo random number generated by A.B. Orue et al. [69] is the X-OR of pseudo random numbers generated by three sawtooth piecewise linear maps with dynamical variation of coefficients and perturbation of the least significant bits. By doing so algebraic attacks may be avoided and period of the output generator is increased to the least common multiple of periods of three individual generators. In 2010 Narendra K Pareek et al.[72] use two chaotic systems which are cross-coupled with each other and their orbits are compared for generating a pseudorandom bit sequence similar to the scheme proposed in [55].

Kohda et al. define many types of binary sequences from the trajectories of chaotic maps using different thresholding mechanisms[46]. They emphasize that chaotic binary sequences of piece-wise linear maps may not retain pseudo random properties due to finite precision of the computer. Nonlinear ergodic maps like logistic map and Chebyshev maps prove to be good candidates for pseudo random generators. Many chaos based pseudo random number generators were proposed which use a single non-linear chaotic map [8, 28, 47, 69, 93] or a couple of chaotic maps [34, 55, 72]. Some of these schemes were shown to be vulnerable to the chosen plaintext attack [6, 7, 81]. To avoid such a situation, it has been suggested that the initial conditions and control parameters have to be perturbed frequently [54]. Perturbation is done after some iterations to avoid dynamical degradation as suggested by Parker et al.[73] thus ensuring the determination of key to be difficult.

During these studies, it becomes clear that using one chaotic map may lead to insecure schemes. Our main claim throughout the thesis has been that judicious use of perturbation along with the shift map leads us to ergodic orbits which can be potentially used for different cryptographic applications. We experiment with Shujun Li's scheme called CCS-PRBG by taking different chaotic maps. We propose a fast coupling scheme by taking logistic and Bernoulli map and then a composition of these two maps to get a novel PRBG scheme. We test the pseudo randomness of the key streams obtained using the standard NIST test suite.

6.1.2 NIST statistical test suite

NIST statistical test suite[77] is used to test the strength of random and pseudorandom number generators and their suitability of using in cryptographic applications. For each of these tests, p-value (probability) is calculated. A p-value ≥ 0.01 would mean that the sequence would be considered to be random with a confidence of 99%. A p-value < 0.01 would mean that the sequence is non-random with a confidence of 99%.

Fifteen statistical tests were developed, implemented and evaluated by NIST namely, simple statistics like frequency mono-bit and block tests; runs test and longest runs of 1's in a block; testing for periodic patterns using rank test, DFT(Discrete Fourier Transform) test; overlapping matching tests and testing aperiodicity using non-overlapping template matching tests, Maurer's universal statistical test, entropy test etc.A Following are the parameter adjustments of various tests:

1. Block Frequency Test - block length(M): 128
2. NonOverlapping Template Test - block length(m): 9
3. Overlapping Template Test - block length(m): 9

4. Approximate Entropy Test - block length(m): 10
5. Serial Test - block length(m): 16
6. Linear Complexity Test - block length(M): 500

More details of the NIST test suite is given in the Appendix.

6.2 Motivation : PRBG Based on Shujun Li's Scheme

In 2001 Shujun Li et al. [55] proposed a pseudo random bit generator based on couple chaotic systems(CCS-PRBG) to provide high security. In this scheme, pseudo random bits are generated by comparing the corresponding values of orbits of two chaotic systems. Let

- p_1 and p_2 be two control parameters and
- x_1 and x_2 be two initial values.
- Both the chaotic maps are synchronized and the keystream $K(i) = g(F_1(i), F_2(i))$ is generated where

$$g(F_1, F_2) = \begin{cases} 1, & \text{if } F_1 > F_2 \\ \textit{no output}, & \text{if } F_1 = F_2 \\ 0, & \text{if } F_1 < F_2 \end{cases}$$

Conditions laid down for the keystream to be pseudorandom are

1. $F_1(x_1, p_1)$ and $F_2(x_2, p_2)$ are surjective maps defined on a same interval $I = [a, b]$
2. $F_1(x_1, p_1)$ and $F_2(x_2, p_2)$ are ergodic on I with unique invariant density functions
3. $F_1(i)$, $F_2(i)$ are asymptotically independent

Hence the challenge is to generate $F_1(i)$, $F_2(i)$ such that the pair are mutually independent in their dynamical behaviour.

6.2.1 Experiment 1: CCS-PRBG with one chaotic map

Shujun Li's scheme is tested for different chaotic maps and it is found to work well with one chaotic map by taking two different initial seeds. We take logistic map with the same control parameter p with different initial seeds x_1 and x_2.
Let F run with x_1 be called F_1 and F run with x_2 be called F_2.
$F_1(x_1, p) = p * x_1 * (1 - x_1)$,
$F_2(x_2, p) = p * x_2 * (1 - x_2)$
Experiment 1 is implemented in two ways.

Variation 1

Shujun Li scheme is implemented using $F_1(x_1, p)$ and $F_2(x_2, p)$. After generating 100 different keystreams of different lengths from 100 to 10000000 bits, randomness testing is performed using NIST statistical test suite. Table 6.1 shows the results for the uniformity of p-values and the proportion of passing sequences.

Statistical test case: Here the key stream length is 10000000 and number of bit streams is 100. Table 6.1 gives the final Analysis Report of NIST Statistical test suite. $c_1, c_2 \ldots c_{10}$ correspond to the frequency of $p - values$, $p - value$ is obtained via the application of chi-square test and *proportion* indicates the number of binary sequences that are passed.

Analysis of results: For a keystream of length 100-bits NIST statistical test suite shows that except frequency, block frequency and runs all the other tests fail. This is due to the small length of the keystream i.e. minimum bit length sequence must be greater than 100-bits. The keystream of length 10000-bits passed all the tests except Maurer's universal statistical test, approximate entropy test, random excursions test and random excursions variant test due to minimum bit length requirements of the tests. For keystreams of length 10000000-bits, as shown in Table 6.1 all the statistical tests are passed. This analysis shows the strength of proposed pseudo random number generator as it passes all the NIST statistical tests and that it is suitable for cryptographic applications.

Variation 2

Same experiment is carried out as in 6.2.1 with timely perturbation of the seed at some intervals and leaving a few pre-iterations. 100 different key streams of length 100000 bits are generated and tested for randomness with NIST statistical test suite. Surprisingly except rank, FFT, universal, random excursions, random excursions variant and linear complexity all the other tests fail. From the analysis

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	p-value	proportion	statistical test
13	11	12	13	10	10	6	12	10	3	0.419021	98/100	Frequency
9	9	12	10	8	11	8	15	11	7	0.834308	100/100	BlockFrequency
13	9	13	8	8	11	8	10	11	9	0.946308	98/100	CumulativeSums
11	12	13	9	11	12	8	12	5	7	0.719747	99/100	CumulativeSums
10	7	9	6	13	14	8	11	9	13	0.678686	98/100	Runs
4	14	6	13	7	13	13	8	15	7	0.115387	100/100	LongestRun
10	10	13	8	5	4	5	17	10	18	0.011791	98/100	Rank
15	9	10	7	5	19	10	5	11	9	0.051942	100/100	FFT
9	13	8	6	10	14	17	8	7	8	0.262249	100/100	NonOverlappingTemplate
15	12	7	12	12	9	7	6	9	11	0.595549	99/100	OverlappingTemplate
8	8	6	11	10	13	8	8	15	13	0.574903	98/100	Universal
10	8	13	12	8	9	14	7	11	8	0.816537	99/100	ApproximateEntropy
8	6	5	11	7	9	9	11	11	10	0.752361	87/87	RandomExcursions
8	3	9	10	8	9	14	10	8	8	0.448892	87/87	RandomExcursionsVariant
10	15	11	7	10	9	7	8	18	5	0.129620	100/100	Serial I
11	13	12	7	8	11	6	12	16	4	0.213309	99/100	Serial II
16	12	7	11	7	7	8	15	7	10	0.304126	99/100	LinearComplexity

Table 6.1: Keystream generated by Logistic map with different seeds and keystream of length 10^7 is seen to pass all the tests of the NIST statistical test suite showing the proportion of passing sequences to be nearly 100.

it is clear that the orbits should be mutually independent and should not evolve identically along the iterations. Failing aperiodic test indicates short periodic cycles have been found.

6.2.2 Experiment 2: CCS-PRBG with logistic and tent maps

We implement Shujun Li's [55] algorithm choosing F_1 as logistic mp and F_2 as tent map; the parameters $p_1 = 4.0, p_2 = 0.98$ and the initial values are randomly chosen to be $x_1 = 0.35281, x_2 = 0.63283$. The resulting key streams are tested with NIST statistical test suite.

Logistic Map $F_1(x_1, p_1) = p_1 * x_1 * (1 - x_1)$,

Tent Map

$$F_2(x_2, p_2) = \begin{cases} 2 * p_2 * x_2 & \text{if } 0 \le x_2 < \frac{1}{2} \\ 2 * p_2 * (1 - x_2) & \text{if } \frac{1}{2} \le x_2 \le 1 \end{cases}$$

Variation 1

No perturbation of the seed is carried out. After generating 100 different key streams of different lengths from 100 to 1000000 bits, randomness testing is performed using NIST statistical test suite. The results show that the key streams generated are very weak. For a keystream of length 100-bits, NIST statistical test suite shows that except cumulative sums and runs test, all the other tests fail. The keystream failed the frequency and block frequency tests even though they satisfy the minimum bit length requirement. For a keystream of length 10000-bits, only discrete fast fourier transform, non overlapping template, linear complexity tests are passed. For keystreams of length 1000000-bits, except rank all the statistical tests fail even though they satisfy minimum bit length requirements. The results are not satisfactory. Report shows p-value is < 0.01 for almost all the tests which indicates that the generated key stream is not random.

Variation 2

Here we left pre-iterations and perturbed seed at some predefined intervals while generating the key stream. 100 different key streams of length 1000000-bits are generated and tested with NIST statistical test suite to determine their randomness. Except rank and linear complexity tests all the other fail. This indicates that the generated key streams are not random. From the analysis it is clear that the orbits of the two maps got synchronized. Most of the key streams have periodic patterns indicating short periodic cycles and failing frequency test indicates that the distribution is not uniform.

It is clear that the scheme of Shujun Li[55] cannot be applied so easily. One has to choose the initial conditions carefully such that they satisfy the 3rd condition of the scheme which is that the orbits generated are mutually independent.

Every orbit of a chaotic map F, using a threshould t, can be converted to a binary sequence. i.e,

$$B(x_i) = \begin{cases} 0, & \text{if } x_i < t \\ 1, & \text{if } x_i \geq t \end{cases}$$

It is well known that the shift map acts on this space as a chaotic map and captures the underlying dynamics of any chaotic map. One can interpret Shujun Li's PRBG scheme as, the bit generated is 1 when the underlying chaotic dynamics of the two maps is out of sync, otherwise it generates a 0. There are methods in the literature to synchronize two chaotic maps[70], but to enforce two orbits of different chaotic maps not to be synchronized, no algorithm is available. In this context we propose four different PRBG's that use only one chaotic orbit and sampling the orbit using Bernoulli shift map for generating bits, thus generating a keystream.

6.3 Proposed PRBG's

We adopt the general approaches of our hashing schemes to generate PRBG. The chaotic maps we choose for our PRBG are logistic map and Bernoulli shift map for bit generation. The definition of Bernoulli map is extended to include the bit that is generally truncated and is defined as, $S(x) = (2x\ mod 1, B(x)), \quad x \in [0, 1)$, where $S_1(x) = 2x\ mod\ 1$ and $S_2(x) = B(x)$,

$$B(x) = \begin{cases} 1, & \text{if } 2x > 1 \\ 0, & otherwise \end{cases}$$

The mapping is often called a shift map. Here instead of truncating the integer part we store the bit in $B(x)$. We investigate Bernoulli shift map for generating bits.

In our experimentations we choose the initial seed as $x_0 = 0.751 \in (0, 1)$ and λ as 4 for which the logistic map is proved to possess positive Lyapunov exponent[23]. We propose to generate bits from the orbits of logistic map in four different ways using the binary representation of points of the orbit as

1. First bit of every point in the orbit.
2. Start sampling the orbit after p iterations. Output the first bit of m points, ignore the next l points of the orbit then again take the first bit of m points and so on.
3. Repeat 2 with additionally perturbing the seed after choosing first bits of m points of the orbit.
4. Extract the 16-th bit of the point rather than the first bit in 3.

PRBG's 1 to 4 adopt the above schemes in the order given.

6.3.1 PRBG 1

In this method we choose an initial seed and iterate the logistic map n times, where n is the bit length of the key stream. At each iteration generate a bit using Bernoulli shift map. Algorithm for the proposed method is given below.

Algorithm 6: PRBG 1

```
1: Choose an initial value $0 < x_0 < 1$, control parameter $\lambda$=4 for the logistic
   map $f(x) = \lambda x(1-x)$, $x = x_0$.
2: for $i = 0$ to $n-1$ do
3:     $K(i) = S_2(x_i)$  $/* shiftbit(B(x_i)) */$
4:     $x_{i+1} \leftarrow f(x_i)$
5: end for
6: Output $K$.
```

Keystreams of different lengths from 100 to 100000 are generated and tested with NIST statistical test suite. The keystreams that satisfy the minimum length requirement passed all the tests. Table 6.2 shows the final analysis report of 100 keystreams of length 1000000.

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	p-value	proportion	statistical test
10	11	6	6	6	7	10	12	19	13	0.085587	96/100	Frequency
10	10	14	6	13	7	11	7	10	12	0.699313	99/100	BlockFrequency
9	10	7	9	9	11	8	9	12	16	0.759756	98/100	CumulativeSums
12	7	9	5	7	6	13	13	15	13	0.236810	99/100	CumulativeSums
9	6	13	7	9	14	16	8	12	6	0.262249	99/100	Runs
15	12	5	11	9	10	10	6	11	11	0.595549	99/100	LongestRun
12	9	10	8	9	16	5	10	7	14	0.383827	100/100	Rank
16	7	8	5	10	8	7	12	13	14	0.236810	98/100	FFT
9	10	7	13	12	10	12	12	7	8	0.883171	100/100	NonOverlappingTemplate
16	9	5	13	10	8	9	10	11	9	0.554420	94/100	OverlappingTemplate
11	7	9	15	10	7	5	15	10	11	0.383827	100/100	Universal
15	17	7	9	6	11	8	9	11	7	0.236810	98/100	ApproximateEntropy
7	4	8	9	9	13	11	7	15	5	0.122325	87/88	RandomExcursions
5	5	12	6	15	11	15	7	5	7	0.021262	87/88	RandomExcursionsVariant
14	10	8	15	12	6	7	12	10	6	0.401199	99/100	Serial I
12	12	9	9	16	14	4	2	12	10	0.055361	99/100	Serial II
8	15	10	9	7	11	12	10	13	5	0.554420	99/100	LinearComplexity

Table 6.2: Analysis report of PRBG 1(the uniformity of p-values and the proportion of passing sequences)

From the report of Table 6.2 it is clear that the keystreams generated are random as their p-values are > 0.01.

6.3.2 PRBG 2

In this method before generating the bit stream, a few iterations (100) of the logistic map are ignored. Also at predefined intervals, some fixed number of iterations are ignored after bit generation. Algorithm for the proposed PRBG 2 is given below:

Algorithm 7: PRBG 2

```
1: Choose an initial value $0 < x_0 < 1$, control parameter $\lambda$=4 for the logistic
   map $f(x) = \lambda x(1-x)$.
2: Ignore pre-iterations of $f$, $x_0 \leftarrow f^p(x_0)$.
3: for $j = 0$ to $n-1$ do
4:    for $i = 0$ to $m-1$ /* Generate $m$-bits */ do
5:       $K(i) = S_2(x_i) \quad /*shiftbitofS*/$;
6:       $x_{i+1} \leftarrow f(x_i)$
7:    end for
8:    Ignore $l$ iterations of $f$.
9: end for
10: Output $K$.
```

Keystreams of different lengths from 100 to 10000000 are generated and tested with NIST statistical test suite for randomness. Keystreams satisfying minimum length requirement passed all the statistical tests. Table 6.3 shows the final analysis report of 100 keystreams of length 10000000(=10^7).

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	p-value	proportion	statistical test
12	11	9	9	14	12	9	7	5	12	0.678686	98/100	Frequency
8	13	8	6	9	9	19	14	8	6	0.085587	99/100	BlockFrequency
11	7	12	16	10	9	10	4	15	6	0.171867	97/100	CumulativeSums
15	6	10	11	10	11	12	11	6	8	0.657933	97/100	CumulativeSums
10	14	4	7	7	11	12	19	9	7	0.055361	98/100	Runs
11	13	10	9	7	9	14	10	6	11	0.798139	100/100	LongestRun
12	8	10	8	13	8	13	6	16	6	0.334538	99/100	Rank
11	11	12	8	12	13	8	5	7	13	0.637119	100/100	FFT
9	9	9	15	10	13	6	6	18	5	0.071177	99/100	NonOverlappingTemplate
14	13	9	12	10	10	9	8	5	10	0.739918	98/100	OverlappingTemplate
9	6	11	9	13	4	16	10	12	10	0.319084	98/100	Universal
10	10	13	12	11	11	10	13	3	7	0.514124	100/100	ApproximateEntropy
10	8	12	7	8	6	9	9	12	8	0.800471	89/89	RandomExcursions
9	7	7	15	7	14	7	6	9	8	0.220448	88/89	RandomExcursionsVariant
8	5	16	19	11	10	9	11	7	4	0.021999	99/100	Serial I
8	15	10	9	11	10	11	9	12	5	0.719747	100/100	Serial II
11	7	5	13	8	12	15	12	7	10	0.437274	100/100	LinearComplexity

Table 6.3: Analysis report of PRBG 2(the uniformity of p-values and the proportion of passing sequences)

From Table 6.3 it is clear that the keystreams are random and can be used for cryptographic applications.

6.3.3 PRBG 3

In this method a few pre-iterations of the logistic map are left and **perturbation of the seed after $m = 24$ iterations** is carried out. Perturbation that is applied to x_i is very small equal to $4 * 10^{-11}$ at i-th iteration. Keystreams of different lengths are generated and tested with NIST statistical test suite. Algorithm for PRBG 3 is given below.

Algorithm 8: PRBG using Method 3

```
1: Choose an initial value $0 < x_0 < 1$, control parameter $\lambda$ =4.0 for the logistic
   map $f(x) = \lambda x(1 - x)$.
2: Ignore $p$ pre-iterations of $f$, $x_0 \leftarrow f^p(x_0)$.
3: for $i = 1$ to $n$ do
4:   for $i = 0$ to $m - 1$ do
5:     $K(i) = S_2(x_i)$;
6:     $x_{i+1} \leftarrow f(x_i)$
7:   end for
8:   Perturb $x_i$
9: end for
10: Output $K$.
```

Table 6.4 gives the final analysis report of NIST statistical test suite for 100 keystreams of length 1000000. Keystreams generated are found to be random.

13	10	6	14	13	5	14	6	8	11	0.262249	98/100	Frequency
7	13	11	18	7	5	8	15	9	7	0.075719	99/100	BlockFrequency
15	7	12	6	13	8	10	6	10	13	0.419021	98/100	CumulativeSums
12	8	13	6	13	10	12	7	10	9	0.779188	99/100	CumulativeSums
11	10	7	14	6	6	14	8	12	12	0.474986	99/100	Runs
13	18	8	8	7	6	11	7	9	13	0.181557	100/100	LongestRun
15	11	11	7	5	15	10	6	11	9	0.319084	97/100	Rank
16	13	9	13	4	9	12	5	9	10	0.202268	98/100	FFT
16	10	8	12	9	8	8	8	8	13	0.637119	99/100	NonOverlappingTemplate
11	6	7	10	11	7	9	12	16	11	0.554420	97/100	OverlappingTemplate
14	12	8	13	15	6	9	6	10	7	0.350485	100/100	Universal
8	12	11	7	14	12	9	8	12	7	0.779188	100/100	ApproximateEntropy
7	4	4	6	6	7	6	5	9	6	0.949602	59/60	RandomExcursions
4	8	6	3	8	6	6	6	7	6	0.931952	59/60	RandomExcursionsVariant
10	7	17	12	15	12	9	9	4	5	0.080519	100/100	Serial I
15	11	9	13	7	9	6	11	13	6	0.455937	98/100	Serial II
14	17	11	11	7	5	11	7	8	9	0.236810	99/100	LinearComplexity

Table 6.4: Analysis report for PRBG 3 (the uniformity of p-values and the proportion of passing sequences)

6.3.4 PRBG 4

This is similar to Method 3, but instead of extracting the first bit of the seed, using Bernoulli shift map, we extract the 16-th bit. Algorithm for the proposed method 4 is given below.

Algorithm 9: PRBG using Method 4

```
 1: Choose an initial value $0 < x_0 < 1$, control parameter $\lambda$ =4.0 for the logistic
    map $f(x) = \lambda x(1 - x)$.
 2: Ignore $p$ pre-iterations of $f$, $x_0 \leftarrow f^p(x_0)$.
 3: for $i = 1$ to $n$ do
 4:    for $i = 1$ to $m$ do
 5:       $K(i) = S_2^{16}(x_i)$;/*Extract 16th bit*/
 6:       $x_i \leftarrow f(x_{i-1})$
 7:    end for
 8:    Perturb $x_i$
 9: end for
10: Output $K$.
```

Keystreams of different lengths are generated and tested for randomness. All the keystreams that satisfy the minimum length requirement satisfy the NIST statistical tests. Table 6.5 gives final analysis report of NIST statistical test suite for keystreams of length 1000000. Report shows the generated keystreams are random.

7	10	6	10	12	10	8	11	14	12	0.798139	100/100	Frequency
13	9	17	7	7	8	15	6	11	7	0.153763	97/100	BlockFrequency
6	8	14	9	10	7	10	10	15	11	0.616305	99/100	CumulativeSums
10	6	8	7	10	11	18	8	13	9	0.289667	100/100	CumulativeSums
9	12	6	9	11	8	9	13	12	11	0.897763	100/100	Runs
12	7	11	5	14	9	7	17	8	10	0.224821	99/100	LongestRun
11	8	7	14	11	8	8	15	9	9	0.678686	97/100	Rank
12	12	8	9	13	10	8	8	10	10	0.964295	99/100	FFT
10	9	4	10	11	15	9	10	14	8	0.494392	100/100	NonOverlappingTemplate
10	7	13	13	7	9	7	12	7	15	0.494392	96/100	OverlappingTemplate
9	13	4	9	11	11	10	7	11	15	0.494392	99/100	Universal
10	13	10	13	10	10	8	4	12	10	0.719747	99/100	ApproximateEntropy
4	9	12	3	6	11	8	6	2	4	0.037157	65/65	RandomExcursions
5	8	10	7	8	8	5	4	6	4	0.689019	65/65	RandomExcursionsVariant
8	7	9	10	16	8	9	8	12	13	0.616305	100/100	Serial I
7	9	8	8	12	8	14	8	14	12	0.678686	100/100	Serial II
6	15	8	4	10	8	12	16	8	13	0.129620	100/100	LinearComplexity

Table 6.5: Analysis report for PRBG 4 (the uniformity of p-values and the proportion of passing sequences)

6.4 Conclusions

Shujun Li's scheme CCS-PRBG is tested for different chaotic maps. Shujun Li's scheme with one chaotic map and different initial seeds yields good results. We propose four new schemes that uses logistic and Bernoulli maps. This combination is very fast since Bernoulli is simply shifting a bit in the corresponding binary representation of the point which can be implemented in hardware.

We proposed PRBG's 1 to 4 that use the interplay between chaotic dynamics and shift dynamics using a composition of logistic and Bernoulli maps. We show the pseudo randomness of the key streams obtained using NIST statistical test suite. Therefore we conclude that the proposed generators are suitable for cryptographic applications.

In the next chapter we use the keystreams generated by these methods for building a stream cipher. We study the distributions of the characters in the ciphertext for their fitness to uniform distribution for all the ciphers using keystreams generated by using these four PRBG's.

CHAPTER 7

Application : Stream Ciphers

7.1 Introduction

In this chapter we use the chaotic pseudo random bit generators PRBG 1 to 4 proposed in the previous chapter 6 for generating a keystream. Stream cipher simply X-OR's the plaintext bits with the keystream one at a time to get ciphertext. If the keystream achieves randomness, it acts almost as the ideal one-time pad in encryption.

Stream ciphers are traditionally based on Linear shift feedback registers(LFSR). It is well known that stream ciphers are much faster than block ciphers, at the same time vulnerable if the keystream is not pseudo random. Since LFSR's are inherently linear, there is an effort to introduce non-linearity by taking more than 2 LFSR's and introducing irregular clocking mechanism. Hence it is clear that non-linear maps have a significant role to play in generating pseudo random bit streams.

Confusion and diffusion are two main principles for designing any encryption algorithm. Diffusion hides the statistical structure of plaintext over ciphertext. Confusion complicates the statistical dependance of ciphertext on plaintext. Diffusion is achieved through mixing property of a chaotic map i.e any set of nonzero initial conditions will be spread eventually as the system evolves over the whole phase space[21]. The influence of a single plaintext digit is spread out over many ciphertext digits.

Baptista proposed chaos based encryption scheme[15] which is found to be weak as the distribution of ciphertext characters is found to be not uniform and prone to all attacks. K.W. Wong et al.[99] modified this scheme by adding a random number to the ciphertext and claims that the distribution of ciphertext is uniform. In 2002, Wong [96] proposes to use a dynamic look-up table instead of a random number. In 2004 Shujun Li et al.[53] enhances Baptista's encryption scheme[15] using pseudo-random number generator to rectify the problems in the encryption scheme and provides countermeasures to make it resistant to attacks. Encryption algorithm must possess properties of diffusion, confusion, sensitivity to changes in secret key and plaintext and mixing[80].

In this chapter, we present four stream ciphers based on the 4 PRBG's proposed in the Chapter 6 using the standard encryption scheme as stated below.

7.1.1 Encryption and decryption

Encryption is carried out by performing X-OR of bits of the plaintext and keystream one at a time.
Plaintext P
Key stream K
Ciphertext C
Encryption : $C(i) = P(i) \oplus K(i)$
Decryption : $P(i) = C(i) \oplus K(i)$

Stream ciphers generated using $PRBG$'s in previous chapter and the distribution of ciphertext characters is discussed in section 7.2. Chi-square goodness of fit test is carried out for those stream ciphers in section 7.2.3. Section 7.3 depicts the avalanche effect obtained in the ciphertexts generated with keystreams whose initial seeds are slightly perturbed.

7.2 Stream Ciphers

Confusion and diffusion properties of chaos based encryption schemes depend on the equi probability of characters in ciphertext for any pseudo-random sequence[11]. If the distribution of characters in the ciphertext is non-uniform then we can get the information about plaintext[5]. Non-equiprobability increases encryption and decryption time[33, 95] as seen in Baptistas encryption scheme[15]. For generating the key stream K we use the pseudo random bit generators proposed in the previous chapter 6. We encrypt a plaintext P whose length is 1000 characters and obtain the ciphertext C, where, $C = P \oplus K$. In the following sections, we analyze each of the stream ciphers obtained with PRBG's for the randomness properties of the cipher.

7.2.1 Stream cipher 1

Ciphertext is constructed using the keystream generated by PRBG 1 and the distribution of characters in the ciphertext is plotted. Figure 7.1 shows that the distribution of plaintext characters is narrow and mostly confined to the range 97 - 120.

Now we apply PRBG 1 to the plain text P to obtain Stream cipher 1 which

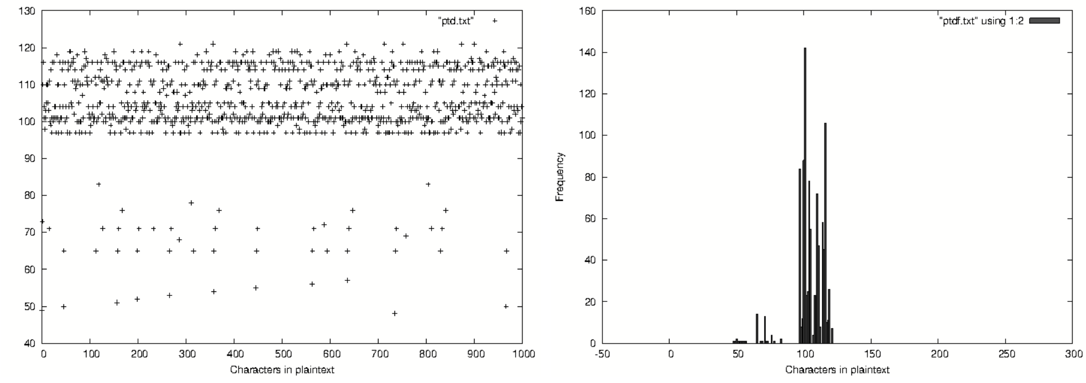


Figure 7.1: Distribution of the input plaintext characters can be seen to be restricted to a narrow range

is plotted in the Figure 7.2. Figure 7.2 visually shows that the distribution of the characters in ciphertext is uniform, thus satisfying the properties of confusion and diffusion as given by Shannon[80].

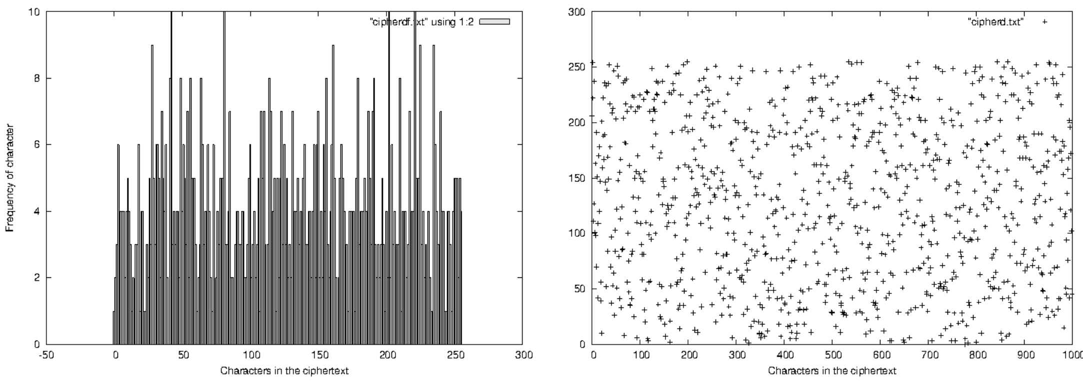


Figure 7.2: Distribution of the ciphertext characters for stream cipher 1

7.2.2 Stream ciphers 2, 3 and 4

Keystreams generated by PRBG methods 2, 3 and 4 have been used to produce stream ciphers 2, 3 and 4 respectively. Now in Figures 7.3, 7.4 and 7.5, the frequency distribution and the scatter plots of the ciphers obtained using these schemes are shown. It is to be noted that the same plain text P is fixed for all the experiments whose distribution is plotted in Figure 7.1. Visually all the plots 'look' to be uniform distributions.

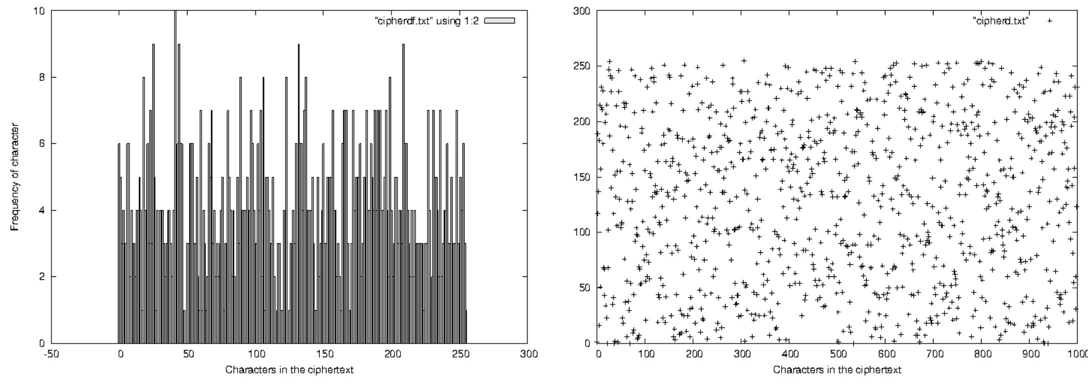


Figure 7.3: Distribution of stream cipher 2 using PRBG 2

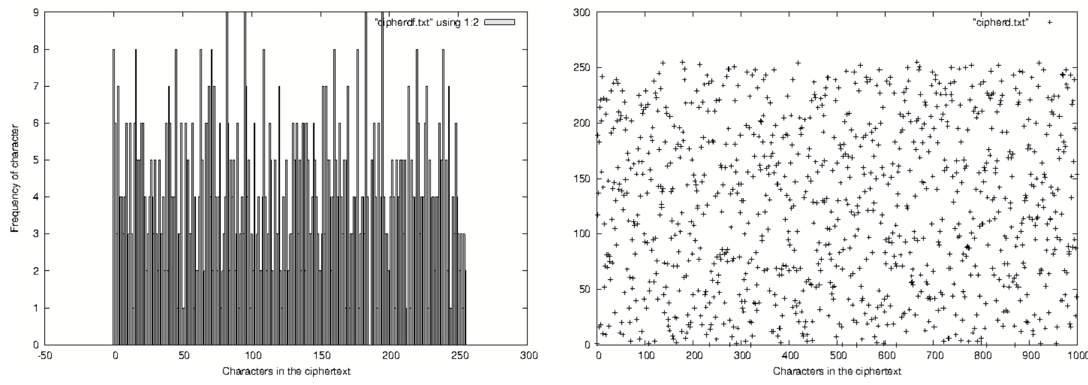


Figure 7.4: Distribution of ciphertext characters in stream cipher 3

7.2.3 Chi-Square fitness test

Though the distributions of the cipher text characters visually look uniform, they need to be validated using the standard fitness test. Chi-square goodness of fit test is used to compute the deviation of the input distribution to the ideal uniform distribution. This experiment tests if the input distribution matches the uniform distribution by chance. The p-value given by ChiTest is such that lower the probability value, higher is the significance of the match.

Stream cipher 4 emerges to be a strong cipher. Recall that it is produced

Method	p-value
Stream cipher 1	0.244
Stream cipher 2	0.633
Stream cipher 3	0.680
Stream cipher 4	0.003

Table 7.1: Clearly only stream cipher 4 is passing the ChiTest and the matching of the others with the uniform distribution is not found to be significant

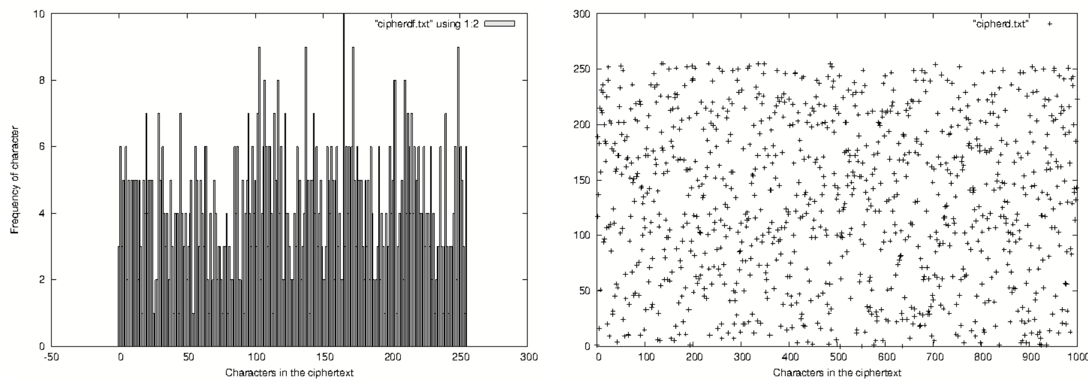


Figure 7.5: Distribution of ciphertext in stream cipher 4

using PRBG 4 which samples the ergodic orbit of the logistic map. This method is strong since it applies perturbation after every 100 iterations and samples 16th bit of the orbit using Bernoulli shift map. The initial seed cannot be obtained since 100 iterations are ignored in the beginning before generating the bits which is a fool-proof scheme against attacking the initial seed.

7.3 Avalanche Effect : Key Sensitivity Analysis

Avalanche effect is seen if any change in key results in changing approximately half of the output bits in the ciphertext. A good encryption scheme spreads the influence of a single key digit over many ciphertext digits[41]. Here we have taken initial seed x_0 as a secret key parameter. A small change in x_0 will make a considerable change in the ciphertext. Mixing property is shown in the parameter space of x_0 in the chaotic map. This effect is shown in the Tables 7.2, 7.3, 7.4 and 7.5. Slightly different keys give different ciphertexts. In order to prove the avalanche effect, we calculate the difference in the cipher after every 1000 bits to show the influence of changed initial seed x'_0 over the entire ciphertext. Following are the details of encryption:

- Plaintext P is of size 8000 bits.
- Initial seed x_0 is changed with negligible difference to, say x'_0.
- K and K' are the key streams generated with initial seeds x_0 and x'_0.
- $C = E(P, K) = P \oplus K$ and $C' = E(P, K') = P \oplus K'$.
- $d_H(C, C')$ is the hamming distance between the two ciphertexts.

In the following sections key sensitivity analysis is carried out by making a tiny change in the initial seed value.

7.3.1 Stream cipher 1

Stream cipher C is constructed using the keystream K generated by PRBG of method 1 taking x_0 as initial seed. Then make a tiny change in x_0 and call it as x'_0. Construct the stream cipher C' with the keystream K' generated using the initial seed x'_0. Calculate $d_H(C, C')$ which gives the number of bits changed for a tiny change in the key. 50% of change indicates good avalanche effect. Table 7.2 depicts the influence of x'_0 over the entire ciphertext at different lengths.

x'_0	Length of bits in ciphertext							
	1000	2000	3000	4000	5000	6000	7000	8000
$x_0 + 10^{-7}$	505	1010	1491	2004	2520	3000	3488	3995
$x_0 + 10^{-8}$	497	989	1482	1991	2503	3001	3496	4015
$x_0 + 10^{-9}$	471	1001	1520	2040	2551	3061	3545	4057
$x_0 + 10^{-10}$	482	1007	1514	1998	2508	3008	3501	4028
$x_0 + 10^{-11}$	472	975	1446	1934	2440	2949	3471	3969

Table 7.2: Hamming distance $d_H(C, C')$ for different x'_0 values very close to $x_0 = 0.751$ for stream cipher 1

From the Table 7.2, it can be seen that the number of bits changed in C' with a tiny change in initial seed $x_0 = x_0 + 10^{-7}$ at 1000-bit length is 505-bits. This indicates a 50% of change in the output resulting in good avalanche effect. On an average 50% change is found in the ciphertexts obtained from different x'_0s. Note that by 10^{-11}, due to dynamical degradation, the effect of diffusion is reducing giving the number of changed bits to be less than 50%.

7.3.2 Stream cipher 2

Experiments carried out in cipher 1 is repeated using the PRBG 2. Table 7.3 gives the influence of x'_0 over the entire ciphertext at different lengths. From the Table 7.3 on an average 50% change is found in the ciphertexts obtained from different x'_0s which indicates good avalanche effect.

7.3.3 Stream cipher 3

Table 7.4 gives the influence of x'_0 over the entire ciphertext at different lengths for stream cipher 3. Good avalanche effect is achieved as shown in the Table 7.4.

x_0'	Length of bits in ciphertext							
	1000	2000	3000	4000	5000	6000	7000	8000
$x_0 + 10^{-7}$	532	1042	1554	2039	2559	3034	3523	4005
$x_0 + 10^{-8}$	512	1012	1518	2017	2502	3002	3517	4038
$x_0 + 10^{-9}$	517	1056	1559	2061	2554	3081	3579	4070
$x_0 + 10^{-10}$	515	1022	1533	2043	2559	3038	3537	4010
$x_0 + 10^{-11}$	506	984	1485	1993	2487	2962	3461	3953

Table 7.3: Hamming distance $d_H(C, C')$ for different x_0' values very close to $x_0 = 0.751$ for stream cipher 2

x_0'	Length of bits in ciphertext							
	1000	2000	3000	4000	5000	6000	7000	8000
$x_0 + 10^{-7}$	517	990	1478	1976	2512	2977	3467	3965
$x_0 + 10^{-8}$	498	1019	1517	2017	2528	2997	3490	3994
$x_0 + 10^{-9}$	497	1003	1493	1993	2500	2985	3493	3987
$x_0 + 10^{-10}$	487	998	1518	2003	2503	2971	3489	3979
$x_0 + 10^{-11}$	486	988	1480	1990	2504	3004	3498	3987

Table 7.4: Hamming distance $d_H(C, C')$ for different x_0' values very close to $x_0 = 0.751$ for cipher 3

7.3.4 Stream cipher 4

The avalanche effect can be seen in the cipher text as the changed bit numbers are tabulated at different lengths for stream cipher 4 in Table 7.5, by perturbing the seed.

x_0'	Length of bits in ciphertext							
	1000	2000	3000	4000	5000	6000	7000	8000
$x_0 + 10^{-7}$	497	998	1488	1985	2512	3010	3529	4037
$x_0 + 10^{-8}$	501	1009	1530	2035	2529	2988	3505	3992
$x_0 + 10^{-9}$	506	1002	1500	1991	2505	3011	3512	4009
$x_0 + 10^{-10}$	500	1008	1512	2010	2524	3010	3520	4014
$x_0 + 10^{-11}$	500	997	1496	1957	2470	2965	3477	3998

Table 7.5: Hamming distance $d_H(C, C')$ for different x_0' values very close to $x_0 = 0.751$ for stream cipher 4

7.4 Conclusions

We are using keystreams generated by pseudo random bit generators PRBG 1-4 (6.3.1, 6.3.2, 6.3.3 and 6.3.4) proposed in the Chapter 6 for constructing different stream ciphers. We show that the characters of the ciphertexts are uniformly distributed for stream cipher 4 using Chi-square goodness of fit test. It should be noted that PRBG's 1-3 show good performance with NIST statistical test suite for pseudo randomness, even though they do not seem to give satisfactory results for uniform distribution test. However good avalanche effect is seen in the ciphertext obtained for all the stream ciphers when the initial seed is varied by a very small quantity. Also we observe the avalanche effect at various lengths of the cipher text for all the stream ciphers.

CHAPTER 8

Conclusions and Future Work

Motivation for the thesis work has been to simplify the design of chaos based algorithms so that they meet the twin goals of (a) reducing the computational complexity and (b) are amenable for cryptanalysis. We propose in this thesis that judicious application of one non-linear map like the classical logistic map intertwined with Bernoulli shift map can generate "truly" random bit sequences that can be used to produce secure hash functions and PRBG's which are then applied to generate integrity check values and stream ciphers.

8.1 Conclusions

We propose two chaotic hashing schemes called Baptista hash function and Bernoulli hash function in Chapters 3 and 4. We adopt the iterated hash functions approach in which chaotic maps are chosen as compression functions. We deviate from the existing schemes proposed in the literature which produce intermediate message digests at each step of compression. We argue that that this is really an artificial requirement for chaos based schemes, unnecessarily adding to computational requirements. We also show that there is really no need to pad the input message to be a multiple of the length of the message digest. Hence we evolve a generic design for a hash function using which we propose two hash functions. One major advantage of this design is that it is adaptable to produce message digests of any given length.

In Chapter 3, a new one-way chaotic hash function is developed based on Baptista's encryption algorithm that gives an output of message digest. We present collision analysis of Baptista hash function and computational analysis of the message digest for confusion and diffusion properties. The proposed Baptista hash function is shown to exhibit nearly 50% mixing of bits in the output of message digest on a one-bit change in the input message. The function is further analyzed and shown to possess preimage resistance. It is shown that the performance of the hash function is comparable with some of the latest algorithms proposed in the literature. There are limitations to this algorithm since the bits generated are from integers which degrade faster than real numbers due to finite precision. Also the speed of the algorithm is not satisfactory. Culling the fine points of the design, we propose an elegant hash function which is almost our 'signature' hash function in Chapter 4.

Bernoulli keyed hash function is proposed in Chapter 4, with an efficient compression scheme and a novel and efficient bit generation scheme. Computational complexity is tremendously reduced for this scheme compared to Baptista hash function. The time complexity of the algorithm is of the order $O(n * M)$, on par with the hashing schemes proposed in the literature[101, 108], where n is the number of iterations per character and M is the length of the input. We show that the hidden constant will be much lesser for our scheme for larger message digests. Extensive validation is carried out at byte, block and the whole message level for collision resistance and Bernoulli keyed hash function. Further, statistical analysis shows that Bernoulli hash function exhibits an average hamming distance of 64.18 bits of diffusion and confusion, much higher than many recent schemes proposed in the literature. Cryptanalysis arguments for this scheme are spelt out to show that it is resistant to preimage and second preimage attacks.

In the next chapter an application of these two proposed hash functions is discussed for constructing Integrity check value (ICV). We show that Baptista and Bernoulli hash fuctions of 3.5 and 4.4 which give an output of 128-bits, can be truncated to extract the first 96-bits of hash value to be used as an ICV. Statistical analysis of confusion and diffusion of Baptista and Bernoulli hash functions is carried out for first 96-bits of the hash value with respect to tiny changes in the input message, initial value and control parameter. Results show the capability of confusion and diffusion is robust and secure. Collision resistance analysis is also carried out and it is shown that the collision probability is quite low. The keyed hash function is tested by varying the two secret keys infinitesimally and shown to achieve 50% variation in the output message digest and hence is robust against collisions and statistical attacks.

In Chapter 6, we propose pseudo random bit generators following our general theme of constructing an ergodic orbit of logistic map generated using intermittent perturbation. By sampling the orbit using Bernoulli shift map in four different ways, 4 pseudo random bit generators(PRBG) are proposed. We also present our experiments with Shujun Li's CCS-PRBG by taking different chaotic maps. The pseudo randomness of the key streams is tested with the standard NIST test suite. We understand that pseudo randomness cannot be proved, but on the other hand, weak key streams can certainly be showed up by the NIST test suite. The key streams generated by the proposed PRBG's pass all the tests of the test suite.

In the last chapter, as an application of PRBG's we propose chaotic stream ciphers. Goodness-of-Fit tests are conducted for these ciphers. One of the encryption algorithms is shown to generate a cipher in which the distribution of the characters show significant correlation to the uniform distribution. It is also shown that the ciphers exhibit good avalanche effect when the initial seed is varied infinitesimally and we can also clearly observe the avalanche effect at different points within the ciphertext as the ciphertext is being generated. This indeed vindicates our hypothesis that stream of bits obtained from orbits of logistic map using perturbation and Bernoulli shift map exhibits randomness.

8.2 Future Directions

If chaos based algorithms are to be taken seriously by the cryptography community, two aspects have to be strengthened. One of the crucial issues is the large time taken by chaos based algorithms. It is true that the chaos community has been considering the piece-wise linear maps instead of non-linear maps for this reason. Yet, the schemes do not meet the speeds of the state of the art algorithms like MD5 and SHA1. We need to focus on this aspect in order to speeden up the proposed algorithms. One advantage of our hash functions is that the time taken does not significantly increase for larger message digests. We have to focus our attention on optimizing the parameters in order to improve efficiency without jeopardizing security. Another significant research direction is that of parallelizing the proposed algorithms. Parallelizing the hash functions can significantly increase efficiency and there are already a few parallel chaotic keyed hash functions that have been proposed in the literature[61, 101]. Since our design is amenable for parallelization, it seems a worth while direction to proceed further.

The second crucial issue is that of crypt analysis of chaos based algorithms. There is a severe theoretical lacuna in this regard. Shujun Li in his Ph.D. thesis [57], writes extensively upon the aspect of dynamical degradation and proposes dynamical indicators for piece-wise linear maps. Similar work is necessary to be carried out for non-linear maps. It is heartening to note that much of the recent work does concentrate on crypt analysis of chaos based cryptographic algorithms [10, 30, 88, 91]. We choose logistic map for all our schemes mainly due to its simplicity and well-understood dynamics, eventhough its underlying invariant density function does not generate uniform distribution. It is worthwhile to explore other simple non-linear maps which exhibit uniform distribution from implementation perspective.

Chaotic symmetric encryption schemes in the literature adopt synchronization approaches[83]. We need to incorporate synchronization into our schemes so that the proposed encryption schemes can be useful for secure communication.

Appendices

APPENDIX A

A.1 NIST statistical test suite

NIST statistical test suite is used to test the strength of random and pseudorandom number generators and their suitability of using in cryptographic applications. For each of these tests, P-value (probability) is calculated. A P-value ≥ 0.01 would mean that the sequence would be considered to be random with a confidence of 99%. A P-value < 0.01 would mean that the conclusion was that the sequence is non-random with a confidence of 99%.

Fifteen statistical tests were developed, implemented and evaluated by NIST. The following describes each of the tests.

1. Frequency (Monobits) Test

   The focus of the test is the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to $\frac{1}{2}$, that is, the number of ones and zeroes in a sequence should be about the same. It is recommended that each sequence to be tested consist of a minimum of 100 bits.

2. Test for Frequency within a Block

   The focus of the test is the proportion of zeroes and ones within M-bit blocks. The purpose of this test is to determine whether the frequency of ones is an M-bit block is approximately M/2. It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., length of the bit string, n $\geq$ 100). Note that n $\geq$ MN. The block size M should be selected such that M $\geq$ 20, M $> .01n$ and N $<$ 100.

3. Runs Test

   The focus of this test is the total number of zero and one runs in the entire sequence, where a run is an uninterrupted sequence of identical bits. A run of length k means that a run consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow. It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., length of the bit string, n $\geq$ 100).

4. Test for the Longest Run of Ones in a Block

   The focus of the test is the longest run of ones within M-bit blocks. The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. Note that an irregularity in the expected length of the longest run of ones implies that there is also an irregularity in the expected length of the longest run of zeroes. Long runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests. The length of each block. The test code has been pre-set to accommodate three values for M: M = 8, M = 128 and M = 10^4 in accordance with the following values of sequence length, n:

Minimumn	*M*
128	8
6272	128
750,000	10^4

5. Random Binary Matrix Rank Test

   The focus of the test is the rank of disjoint sub-matrices of the entire sequence. The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence.The probabilities for M = Q = 32 have been calculated and inserted into the test code. Other choices of M and Q may be selected, but the probabilities would need to be calculated. The minimum number of bits to be tested must be such that n $\geq$ 38MQ (i.e., at least 38 matrices are created). For M = Q = 32, each sequence to be tested should consist of a minimum of 38,912 bits.

6. Discrete Fourier Transform (Spectral) Test

   The focus of this test is the peak heights in the discrete Fast Fourier Transform. The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness. It is recommended that each sequence to be tested consist of a minimum of 1000 bits (i.e., n $\geq$ 1000).

7. Non-overlapping (Aperiodic) Template Matching Test

The focus of this test is the number of occurrences of pre-defined target substrings. The purpose of this test is to reject sequences that exhibit too many occurrences of a given non-periodic (aperiodic) pattern. For this test and for the Overlapping Template Matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. For this test, when the pattern is found, the window is reset to the bit after the found pattern, and the search resumes. The test code has been written to provide templates for m = 2, 3,...,10. It is recommended that m = 9 or m = 10 be specified to obtain meaningful results. Although N = 8 has been specified in the test code, the code may be altered to other sizes. However, N should be chosen such that N $\leq$ 100 to be assured that the P-values are valid. Additionally, be sure that M > 0.01 n and N = n/M.

8. Overlapping (Periodic) Template Matching Test

   The focus of this test is the number of pre-defined target substrings. The purpose of this test is to reject sequences that show deviations from the expected number of runs of ones of a given length. Note that when there is a deviation from the expected number of ones of a given length, there is also a deviation in the runs of zeroes. Runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests. For this test and for the Non-overlapping Template Matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. For this test, when the pattern is found, the window again slides one bit, and the search is resumed. The values of K, M and N have been chosen such that each sequence to be tested consists of a minimum of 106 bits (i.e., n $\geq$ 106). Various values of m may be selected, but for the time being, NIST recommends m = 9 or m = 10.

9. Maurer's Universal Statistical Test

   The focus of this test is the number of bits between matching patterns. The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information. An overly compressible sequence is considered to be non-random. It is recommended that each sequence to be tested consist of a minimum of 387,840 bits.

10. Linear Complexity Test

    The focus of this test is the length of a generating feedback register. The purpose of this test is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterized by a longer feedback register. A short feedback register implies non-randomness.

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits.

11. Serial Test

    The focus of this test is the frequency of each and every overlapping m-bit pattern across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. The pattern can overlap. It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits.

12. Approximate Entropy Test

    The focus of this test is the frequency of each and every overlapping m-bit pattern. The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence.

13. Cumulative Sum (Cusum) Test

    The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence. The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the random walk should be near zero. For non-random sequences, the excursions of this random walk away from zero will be too large. It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e.,length of the bit stream n $\geq$ 100).

14. Random Excursions Test

    The focus of this test is the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is found if partial sums of the (0,1) sequence are adjusted to (-1, +1). A random excursion of a random walk consists of a sequence of n steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number of visits to a state within a random walk exceeds what one would expect for a random sequence. It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits (i.e.,length of the bit stream n $\geq 10^6$).

15. Random Excursions Variant Test

    The focus of this test is the number of times that a particular state occurs in a cumulative sum random walk. The purpose of this test is to detect deviations from the expected number of occurrences of various states in the random walk. It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits (i.e.,length of the bit stream n $\geq 10^6$).

REFERENCES

[1] A. Akhavan, A. Samsudin, and A. Akhshani. Hash function based on piece-wise nonlinear chaotic map. *Chaos, Solitons and Fractals*, 42(2):1046–1053, 2009.

[2] G. Alvarez and S. Li. Some basic cryptographic requirements for chaos-based cryptosystems. *International Journal of Bifurcation and Chaos*, 16(8):2129–2151, 2006.

[3] G. Alvarez, F. Monotoya, G. Pastor, and M. Romera. Chaotic cryptosystems. *IEEE Int. Carnahan Conf. Security Technology*, pages 332–338, 1999.

[4] G. Alvarez, F. Montoya, and G. Pastor. Crypt-analysis of a discrete chaotic cryptosystem using external key. *Physics Letters A*, 319:334–339, 2003.

[5] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Cryptanalysis of an ergodic chaotic cipher. *Physics Letters A*, 311:172–179, 2003.

[6] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Cryptanalysis of dynamic look-up table based chaotic cryptosystems. *Physics Letters A*, 326:211–218, 2004.

[7] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Keystream cryptanalysis of a chaotic cryptographic method. *Computer Physics Communications*, 156:205–207, 2004.

[8] M. Andrecut. Logistic map as a random number generator. *International Journal of Modern Physics B*, 12:921–930, 1998.

[9] F. Argenti, S. Benzi, E. D. Re, and R. Genesio. Stream cipher system based on chaotic maps. *Proc. SPIE 4122, Mathematics and Applications of Data/Image Coding, Compression, and Encryption III*, 10, 2000.

[10] D. Arroyo, G. Alvarez, J. M. Amigo, and S. Li. Cryptanalysis of a family of self-synchronizing chaotic stream ciphers. *Communications in Nonlinear Sciences and Numerical Simulation*, 16(2):805–813, 2011.

[11] D. Arroyo, G. Alvarez, and S. Li. Some hints for the design of digital chaos-based cryptosystems: Lessons learned from cryptanalysis. *IFAC*, 2(1):171–175, 2009.

[12] D. Arroyo, R. Rhouma, G. Alvarez, S. Li, and V. Fernandez. On the security of a new image encryption scheme based on chaotic map lattices. *Chaos*, 18(033112):1–16, 2008.

[13] J. M. Bahi, J. F. Couchot, and C. Guyeux. Performance analysis of a keyed hash function based on discrete and chaotic proven iterations. *INTERNET 2011 : The Third International Conference on Evolving Internet*, pages 52–57, 2011.

[14] J. M. Bahi and C. Guyeux. Hash functions using chaotic iterations. *Journal of Algorithms & Computational Technology*, 4(2):167–181, 2010.

[15] M. S. Baptista. Cryptography with chaos. *Physics Letters A*, 240:1–2, 1998.

[16] M.S. Baptista. Chaos in cryptography. *Phys. Lett. A*, 240:50–54, 1998.

[17] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocals. *ACM conference on computer and communications security*, ACM press:62–73, 1993.

[18] R. Bose. Novel public key encryption technique based on multiple chaotic systems. *Phys. Rev. Lett*, 95(098702):1–4, 2005.

[19] J. A. Buchmann. *Introduction to cryptography.* Springer, 2001.

[20] Werndl Charlotte. What are the new implications of chaos for unpredictability? *The British Journal for the Philosophy of Science*, 60(1):195–220, 2009.

[21] I. P. Cornfeld, S. V. Fomin, and Ya. G. Sinai. *Ergodic theory.* Berlin: Springer, 1982.

[22] H. Delfs and H. Knebl. *Introduction to cryptography.* Springer, 2002.

[23] R. L. Devaney. *An introduction to chaotic dynamical systems.* Addison-Wesley, 1989.

[24] J. Ford. How random is a coin toss: Chaos in nonlinear systems. *Proceedings of the Ergodic Year at Maryland, Ed. by Anatole Katok, Birkhauser Series: Progress in Mathematics*, 1980.

[25] J. Fridrich. Symmetric ciphers based on two-dimensional chaotic maps. *Int. J. Bifurcation and Chaos*, 8(6):1259–1284, 1998.

[26] L. Gao, X. Wang, and W. Zhang. Chaotic hash function based on tandem-dm construction. *IEEE*, pages 1745–1749, 2011.

[27] J. Gleick. *CHAOS: Making a new science.* Penguin, 1987.

[28] J. A. Gonzalez and R. Pino. Random number generator based on unpredictable chaotic functions. *Computer Physics Communications*, 120:109–114, 1999.

[29] M. Gotz, K. Kelber, and W. Schwarz. Discrete-time chaotic encryption systemspart i: Statistical design approach. *IEEE Trans. Circuits and SystemsI*, 44(10):963–970, 1997.

[30] W. Guo, X. Wang, D. He, and Y. Cao. Cryptanalysis on a parallel keyed hash function based on chaotic maps. *Physics Letters A*, 373:3201–3206, 2009.

[31] Y. Hu, X. Liao, K.W. Wong, and Q. Zhou. A true random number generator based on mouse movement and chaotic cryptography. *Chaos Solitons and Fractals*, 40:2286–2293, 2009.

[32] F. Huang and Z. H. Guan. Cryptosystem using chaotic keys. *Chaos, Solitons and Fractals*, 23:851–855, 2005.

[33] F. Huang and Z. H. Guan. A modified method of a class of recently presented cryptosystems. *Chaos, Solitons & Fractals*, 23(5):1893–1899, 2005.

[34] L. Huaping, S. Wang, and H. Gang. Pseudo-random number generator based on coupled map lattices. *International Journal of Modern Physics B*, 18:2409–2414, 2004.

[35] G. Jakimoski and L. Kocarev. Analysis of some recently proposed chaos-based encryption algorithms. *Phys. Lett. A*, 291:381–384, 2001.

[36] A. Kanso and M. Ghebleh. A fast and efficient chaos-based keyed hash function. *Commun Nonlinear Sci Numer Simulat*, 18:109–123, 2013.

[37] A. Kanso, H. Yahyaoui, and M. Almulla. Keyed hash function based on a chaotic map. *Information Sciences*, 186:249–264, 2012.

[38] B. Hasselblatt. A. Katok. *A first course in dynamics: With a panorama of recent developments.* Cambridge University Press. ISBN 0-521-58750-6, 2003.

[39] K. Kelber and W. Schwarz. General design rules for chaos based encryption system. *Proc. of Nolta 2005*, pages 465–468, 2005.

[40] H. S. Kellert. *In the wake of chaos: Unpredictable order in dynamical systems.* University of Chicago Press. p. 32. ISBN 0-226-42976-8, 1993.

[41] L. Kocarev. Chaos-based cryptography: a brief overview. *IEEE Circuits and Systems Magazine*, 1(3):6–21, 2001.

[42] L. Kocarev and G. Jakimoski. Pseudorandom bits generated by chaotic maps. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 50:123–126, 2003.

[43] L. Kocarev, G. Jakimoski, T. Stojanovski, and U. Parlitz. From chaotic maps to encryption schemes. *IEEE Int. Symposium Circuits and Systems*, 4:514–517, 1998.

[44] L. Kocarev and S. Lian. *Chaos-based cryptography: Theory, algorithms and applications*. Springer, 2011.

[45] L. Kocarev and Z. Tasev. Public-key encryption based on chebyshev maps. *IEEE Trans. Circuits Syst I*, pages 28–31, 2003.

[46] T. Kohda and A.Tsuneda. Statistics of chaotic binary sequences. 1997.

[47] V. V. Kolesov, R. V. Belyaev, and G. M. A. Voronov. Digital random-number generator based on the chaotic signal algorithm. *Journal of Communications Technology and Electronics*, 46:1258–1263, 2001.

[48] H. Krawczyk, M. Bellare, and R. Canetti. Rfc2104 - hmac: Keyed-hashing for message authentication. 1997.

[49] X. Lai and J. L. Massey. Hash function based on block ciphers. *EUROCRYPT92, LNCS, Springer*, 658:55–70, 1992.

[50] C. Li, S. Li, D. Zhang, and G. Chen. Cryptanalysis of a chaotic neural network based multimedia encryption scheme. *Advances in Multimedia Information Processing, Lecture Notes in Computer Science (Springer-Verlag)*, 3333:418–425, 2004.

[51] D. Li, G. Hu, and S. Wang. A keyed hash function based on the modified coupled chaotic map lattice. *Commun Nonlinear Sci Numer Simulat*, 17:2579–2587, 2012.

[52] S. Li, G. Chen, and X. Mou. On the dynamical degradation of digital piecewise linear chaotic maps. *Int. J. Bifurc. Chaos*, 15:3119–3151, 2005.

[53] S. Li, G. Chen, K. W. Wong, X. Mou, and Y. Cai. Baptista-type chaotic cryptosystems: Problems and countermeasures. *Physics Letters A*, 332:368–375, 2004.

[54] S. Li, C. Li, G. Chen, N. G. Bourbakis, and K. T. Lo. A general quantitative cryptanalysis of permutation-only multimedia ciphers against plain-text attacks. *Signal Processing: Image Communication*, 23(3):212–223.

[55] S. Li, X. Mou, and Y. Cai. Pseudo-random bit generator based on couple chaotic systems and its application in stream-ciphers cryptography. *Lecture Notes in Computer Science*, 2247:316–329, Progress in Cryptology INDOCRYPT 2001.

[56] S. Li, X. Mou, Z. Ji, and J. Zhang. Cryptanalysis of a class of chaotic stream ciphers. *J. Electronics & Information Technology*, 25:473–478, 2003.

[57] Shujun Li. *Analyses and new designs of digital chaotic ciphers*. Ph.D Dissertation, 2006.

[58] X. M. Li, H. B. Shen, and X. L. Yan. Characteristic analysis of a chaotic random number generator using piece-wise-linear map. *Journal of Electronics and Information Technology*, 27:874–878, 2005.

[59] Y. Li, S. Deng, and D. Xiao. A novel hash algorithm construction based on chaotic neural network. *Neural Comput & Applic*, 20:133–141, 2011.

[60] Y. Li, D. Xiao, and S. Deng. Keyed hash function based on a dynamic lookup table of functions. *Information Sciences*, 214:56–75, 2012.

[61] Y. Li, D. Xiao, S. Deng, Q. Han, and G. Zhou. Parallel hash function construction based on chaotic maps with changeable parameters. *Neural Computing and Applications*, 20(8):1305–1312, 2011.

[62] S. Lian, Z. Liu, Z. Ren, and H. Wang. Hash function based on chaotic neural networks. *Proceedings of International Symposium on Circuits and Systems*, pages 237–240, 2006.

[63] T. Lin and L. O. Chua. New class of pseudo-random number generator based on chaos in digital filters. *International Journal of Circuit Theory and Applications*, 21:473–480, 1993.

[64] C. Madson and R. Glenn. Rfc2403 - the use of hmac-md5-96 within esp and ah. 1998.

[65] C. Madson and R. Glenn. Rfc2403 - the use of hmac-sha-1-96 within esp and ah. 1998.

[66] R. A. J. Matthews. On the derivation of a chaotic encryption algorithm. *Cryptologia XIII*, pages 29–42, 1989.

[67] L. Min, P. Fei, and C. Guan-Rong. Constructing a one-way hash function based on the unified chaotic system. *Chinese Physics B*, 17(10):3588–3595, 2008.

[68] S. Oishi and H. Inoue. Pseudo-random number generators and chaos. *Transactions of the Institute of Electronics and Communication Engineers of Japan E*, 65:534–541, 1982.

[69] A. B. Orue, G. Alvarez, A. Guerra, G. Pastor, M. Romera, and F. Montoya. Trident, a new pseudo random number generator based on coupled chaotic maps. *Advances in Intelligent and Soft Computing*, 85:183–190, 2010.

[70] E. Ott. *Chaos in dynamical systems.* Cambridge University Press, 1993.

[71] N. K. Pareek, V. Patidar, and K. K. Sud. Discrete chaotic cryptography using external key. *Physics Letters A*, 309:75–82, 2003.

[72] N. K. Pareek, V. Patidar, and K. K. Sud. Random bit generator using chaotic maps. *International Journal of Network Security*, 10(1):32–38, 2010.

[73] A. T. Parker and K. M. Short. Reconstructing the keystream from a chaotic encryption scheme. *IEEE Trans. Circuits Syst I*, 48:624–630, 2001.

[74] Peitgen, Jurgens, and Saupe. *Chaos and fractals : New frontiers of science.* Springer, 1992.

[75] J. Peng, D. Zhang, Y. Liu, and X. Liao. A double-piped iterated hash function based on a hybrid of chaotic maps. *IEEE*, pages 358–365, 2008.

[76] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. *Fast Software Encryption (Springer-Verlag)*, 2004.

[77] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. *NIST Special Publication 800-22 Rev.1a: A statistical test suite for random and pseudorandom number generators for cryptographic applications.* NIST, Apr. 2010.

[78] T. Sang, R. Wang, and Y. Yan. Perturbance-based algorithm to expand cycle length of chaotic key stream. *Electron. Lett*, 34:873–874, 1998.

[79] B. Schneier. *Applied cryptography.* 2nd ed. New York: Wiley, 1996.

[80] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949.

[81] E. Solak. Cryptanalysis of a chaos-based image encryption algorithm. *Physics Letters A*, 373:1357–1360, 2009.

[82] D.R. Stinson. *Cryptography: Theory and practice.* CRC Press, 1995.

[83] T. Stojanovski and L. Kocarev. Chaos-based random number generators - part i: Analysis. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48:281–288, 2001.

[84] T. Stojanovski, J. Pihl, and L. Kocarev. Chaos-based random number generators - part ii: Practical realization. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48:382–385, 2001.

[85] L. Wang, F. P. Wang, and Z. J. Wang. Novel chaos-based pseudo-random number generator. *Acta Physica Sinica*, 55:3964–3968, 2006.

[86] S. Wang and G. Hua. Hash function based on chaotic map lattices. *American Institute of Physics*, 17:1–9, 2007.

[87] S. Wang and H. Yu. How to break md5 and other hash functions. *Proc. EUROCPYPT 2005 Advances in Cryptology, LNCS*, 3494:19–35, 2005.

[88] X. Wang, W. Guo, W. Zhang, M. Khurram Khan, and K. Alghathbar. Cryptananlysis and improvement on a parallel keyed hash function based on chaotic neural network. *Telecommun Syst*, 52:515–524, 2013.

[89] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full sha-1. *Proc. CRYPTO 2005 Advances in Cryptology, LNCS*, 3621:17–36, 2005.

[90] X. Wang and C. Yu. Cryptanalysis and improvement on a cryptosystem based on a chaotic map. *Computers and Mathematics with Applications*, 57:476–482, 2009.

[91] X. Wang and J. Zhao. Cryptanalysis on a parallel keyed hash function based on chaotic neural network. *Neurocomputing*, 73:3224–3228, 2010.

[92] Y. Wang, X. Liao, D. Xiao, and K. Wong. One-way hash function construction based on 2d coupled map lattices. *Inform Sci*, 178:1391–1406, 2008.

[93] Y. Wang, H. Shen, and X. Yan. Design of a chaotic random number generator. *Chinese Journal of Semiconductors*, 26:2433–2439, 2005.

[94] R. G. Watts. *Global warming and the future of the earth.* Morgan & Claypool. p. 17, 2007.

[95] J. Wei, X. Liao, K. Wong, T. Zhou, and Y. Deng. Analysis and improvement for the performance of baptista's cryptographic scheme. *Physics Letters A*, 354:101–109, 2006.

[96] K. W. Wong. A fast chaotic cryptographic scheme with dynamic look-up table. *Phys. Lett. A*, 298:238–242, 2002.

[97] K. W. Wong. Modified baptista type chaotic cryptosystem. *Phys. Lett. A*, 298:238–242, 2002.

[98] K. W. Wong. A combined chaotic cryptographic and hashing scheme. *Phys Lett A*, 307:292–298, 2003.

[99] W. Wong, L. Lee, and K. Wong. A modified chaotic cryptographic method. *Computer Physics Communications*, 138:234–236, 2001.

[100] D. Xiao and X. Liao. A combined hash and encryption scheme by chaotic neural network. *Advances in Neural Networks, Lecture Notes in Computer Science*, 3174:633–638, 2004.

[101] D. Xiao, X. Liao, and Y. Wang. Parallel keyed hash function construction based on chaotic neural network. *Neurocomputing*, 72:2288–2296, 2009.

[102] D. Xiao, X.F. Liao, and S.J. Deng. One-way hash function construction based on the chaotic map with changeable parameter. *Chaos, Solitons & Fractals*, 24:65–71, 2005.

[103] H. Yang, K. W. Wong, X. Liao, Y. Wang, and D. Yang. One-way hash function construction based on chaotic map network. *Chaos, Solitons and Fractals*, 41(5):2566–2574, 2009.

[104] L. Yang and T. Xiao-Jun. A new pseudorandom number generator based on a complex number chaotic equation. *Chin. Phys. B*, 21(9):1–8, 2012.

[105] Q. Yang, T. Gao, L. Fan, and Q. Gu. Analysis of one-way alterable length hash function based on cell neural network. *Journal of Information Assurance and Security*, 5:196–200, 2010.

[106] X. Yi. Hash function based on chaotic tent maps. *IEEE Transactions on Circuits and Systems-II: Express Briefs*, 52(6):354–357, 2005.

[107] H. Yu, Y. Lu, X. Yang, and Z. Zhu. One-way hash function construction based on chaotic coupled map network. *IEEE*, pages 193–197, 2011.

[108] J. Zhang, X. Wang, and W. Zhang. Chaotic keyed hash function based on feedforward-feedback nonlinear digital filter. *Phys Lett A*, 362(5-6):439–448, 2007.

[109] H. Zhongquan. A more secure parallel keyed hash function based on chaotic neural network. *Communications in nonlinear science and numerical simulation*, 16:3245–3256, 2011.

[110] K. J. Cerm . Digital generators of chaos. *Phys. Lett. A*, 214:151–160, 1996.