

100793

**A NEW CRYPTANALYSIS METHOD OF CELLULAR
AUTOMATA BASED ENCRYPTION SYSTEMS**

100793

İC. YÜKSEKÖĞRETİM KURULU
DOKÜMANASYON BİRİMİ

**Ph.D. Thesis by
A. Murat APOHAN, M.Sc.**

504932026

Date of submission : 20 December 1999

Date of defence examination: 25 August 2000

Supervisor (Chairman): Assoc. Prof. Dr. M. Ertuğrul ÇELEBİ *M. Ertuğrul Çelebi*

Members of the Examining Committee Prof.Dr. Bülent ÖRENCİK (İTÜ.) *B. Örencik*

Prof.Dr. Ümit AYGÖLÜ (İTÜ.) *Ü. AYGÖLÜ*

Prof.Dr. Erdal PANAYIRCI (İÜ.) *E. Panayirci*

Prof.Dr. İsmail GÜLOĞLU (METÜ.) *İ. Güloğlu*

AUGUST 2000

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

**HÜRESEL OTOMATA TABANLI ŞİFRELEME
SİSTEMLERİ İÇİN YENİ BİR ŞİFRE ANALİZ YÖNTEMİ**

DOKTORA TEZİ
Y. Müh. A. Murat APOHAN
504932026

Tezin Enstitüye Verildiği Tarih : 20 Aralık 1999
Tezin Savunulduğu Tarih : 25 Ağustos 2000

Tez Danışmanı :

Doç.Dr. M. Ertuğrul ÇELEBİ

Diğer Jüri Üyeleri

Prof.Dr. Bülent ÖRENCİK (İTÜ.)

Prof.Dr. Ümit AYGÖLÜ (İTÜ.)

Prof.Dr. Erdal PANAYIRCI (İÜ.)

Prof.Dr. İsmail GÜLOĞLU (ODTÜ.)

AĞUSTOS 2000

ACKNOWLEDGEMENTS

The study described in this thesis was carried out at TUBITAK National Research Institute of Electronics & Cryptology.

I would like to express my gratitude to Assoc. Prof. Dr. M. Ertugrul ÇELEBİ for his support and guidance.

I am also grateful to Prof. Dr. Çetin K. KOÇ for his guidance and encouragement of this research.

Special thanks are due to Assoc. Prof. Dr. Nezh GEÇKİNLİ for the time he has devoted to reading and commenting.

I wish to thank to my manager Alpaslan BABAOĞLU for his patience and confidence.

August 2000

A. Murat APOHAN

TABLE OF CONTENTS

LIST OF ABBREVIATIONS	v
LIST OF FIGURES	vi
LIST OF TABLES	vii
LIST OF SYMBOLS	viii
SUMMARY	ix
ÖZET	xiv
1. INTRODUCTION	1
2. CRYPTOGRAPHIC TERMINOLOGY AND CONCEPTS	5
2.1. Complexity Theory	6
2.2. A Brief Definition and Concept of Cryptanalysis	10
2.2.1. Perfect Secrecy and Shannon Measure on Security	11
2.2.2. Exhaustive Search	12
2.2.3. Differential and Linear Cryptanalysis of DES	14
2.2.3.1. Differential Cryptanalysis of DES	15
2.2.3.2. Linear Cryptanalysis of DES	17
2.3. Cryptographic Hash Functions	18
2.3.1. General Classification and Security of Hash Functions	19
2.4. Walsh Transforms and Their Properties	21
2.4.1. The Best Affine Approximation of Boolean Functions	23
3. A SURVEY ON CELLULAR AUTOMATA AND ITS CRYPTOGRAPHIC APPLICATIONS	26
3.1. Cryptographic Applications of Cellular Automata	30
3.2. Known Cryptanalysis Methods of Cellular Automata	33
3.2.1. Cryptanalysis of Wolfram Stream Cipher	33
4. THE ZIGZAG ALGORITHM	43
4.1. Brief Description of The Zigzag Algorithm	45
4.1.1 The Definition of Zigzag Algorithm with the Algorithmic Steps	46
4.2. Analysis of The Zigzag Algorithm	50
4.2.1 Analytic Results on Time Complexity Function of Zigzag Algorithm.	51
4.2.2. Computer Aided Analysis of Zigzag Algorithm	56
4.2.2.1. Empirical Derivation of Zigzag Algorithm Complexity	57
4.2.2.2. Analysis of Halting Steps of Zigzag Algorithm	61
4.3. Presentation of Halting Steps of Zigzag Algorithms with The Examples	64
5. APPLICATIONS OF PROPOSED ZIGZAG ALGORITHM	72
5.1. Applications of Zigzag Algorithm to Wolfram's Pseudorandom Number Generator.	72
5.2. Applications of Zigzag Algorithm to Damgard Hash Function	74

CONCLUSIONS	77
REFERENCES	79
APPENDIX A	84
BIOGRAPHY	89



LIST OF ABBREVIATIONS

- CA : Cellular Automata.
BAA : Best Affine Approximation.
NP : Nondeterministic polynomial complexity class.
GF : Galois field.



LIST OF TABLES

	<u>Page Number</u>
Table 2.1: Execution time of different problems.	8
Table 2.2: Effect of improved technology on several polynomial and exponential time algorithms.	9
Table 2.3: The necessary amount of workload for differential cryptanalysis.	16
Table 4.1: Experimental results of execution time versus automata size.	58
Table 4.2: Transformed experimental results for linear probabilistic model.	59
Table 4.3: The invertible functions for different size of cellular automata and theirs percentile with respect to whole set of functions.	62
Table 4.4: The success ratio of zigzag algorithm at the different halting steps.	63
Table 4.5: The experimental and expected success ratio of zigzag algorithm.	63
Table 4.6: Affine functions of CA30.	65

LIST OF FIGURES

	<u>Page Number</u>
Figure 2.1: The general scheme of symmetric cryptographic system.	6
Figure 3.1: Global State transition diagram of rule 18 and periodic boundary cellular automata (Wolfram et. al, 1984).	29
Figure 3.2: Triangle determined by $s_{i-n}(t), \dots, s_{i+n}(t)$.	36
Figure 3.3: The space-time diagram of the computation (Wolfram, 1986).	40
Figure 4.1: One-dimensional cellular automata with the radius one updating function.	44
Figure 4.2: Schematic representation of the first four steps of zigzag algorithm.	45
Figure 4.3: The experimental values for the size of the search space.	55
Figure 4.4: Plot of the experimental results.	59
Figure 4.5: Plot of experimental results and curve of the calculated complexity function.	61
Figure 5.1: Schematic representation of Damgard hash function.	75
Figure 5.2: Schematic representation of zigzag algorithm.	75

LIST OF SYMBOLS

$O(\cdot)$: Asymptotic Upper Bound.

α : Probability of agreement.

n : Size of cellular automata.

P, p : Plaintext.

C, c : Ciphertext.

K, k : Encryption key.

$f(x)$: Boolean function.

$S_f(w)$: Walsh transformation of Boolean function f .

$\oplus$: Logical exclusive or operation.

$\vee$: Logical or operation.

S_k : Internal state of cellular automata at the k 'th step.

$E[\cdot]$: Expected value.

A NEW CRYPTANALYSIS METHOD OF CELLULAR AUTOMATA BASED ENCRYPTION SYSTEMS

SUMMARY

Cryptography is a scientific discipline that is concerned for the protection of information transmitted through communications networks or stored in electronic media. It can also be used for message authentication, digital signature and personal identification for authorizing electronic funds transfer, credit card transactions, protecting programs against viruses. A basic problem in cryptography is devising procedures to transform messages (plaintext) into cryptograms (ciphertext) that can withstand intense cryptanalysis (the techniques used by opponents to penetrate encrypted communications and recover the original information).

The world of cryptology has searched a measure for the security of the encryption algorithm. Shannon is the first person that gave the mathematical model for the security of cryptographic system. Obtaining plaintext from the ciphertext without having encryption key and/or algorithm is called cryptanalysis. The general assumption that is usually made is that the opponent, the cryptanalyst, knows the cryptosystem with all details being used but not the key. This is usually referred to as Kerckhoff's principle. Of course, if cryptanalyst does not know the cryptosystem being used, that will make his task more difficult. However, the security of a cryptosystem is not based on the assumption that cryptanalyst does not know what system is being employed. Hence, cryptosystem is designed under Kerckhoff's principle. Different types of attacks can be enumerated as follows:

1. Ciphertext Only: The opponent possesses a string of ciphertext, c .
2. Known Plaintext: The opponent possesses a string of plaintext p , and corresponding ciphertext c .
3. Chosen Plaintext: The opponent has obtained temporary access to the encryption machinery. Hence, he can choose a plaintext string, p , and construct the corresponding ciphertext string, c .
4. Chosen Ciphertext: The opponent has obtained temporary access to the decryption machinery. Hence, he can choose a ciphertext string, c , and construct the corresponding plaintext string, p .

Work factor measures what is needed to carry out a specific analysis or attack against a cryptographic algorithm. In practice, there is no universally accepted, fixed set of parameters used to express the work factor. However, it is frequently measured in one or more of the following: cryptanalyst hours, number of mathematical or logical operations, computing resources, necessary special hardware and calendar time. To be useful, the work factor should be expressed using parameters, which can be translated for the purpose of comparison into a common base, such as cost in dollars.

In an exhaustive attack, an attempt is made to recover the plaintext or key by using direct search methods. For example, in key exhaustion, a known plaintext is enciphered with a trial key and result is compared for equality with the known corresponding ciphertext. If only ciphertext is available, it can be decrypted with the trial key and the resulting plaintext can be inspected to see if it makes any sense. In this way, it can be determined if the trial key is a candidate for the unknown key or not.

The work factor of an exhaustive attack, which is directly proportional to the number of trials, is easily determined even when the number of trials is so large that the attack is not feasible. This is not the case with some other attacks from a definition of the cryptographic algorithm solved for the variable or variables representing the known message or key. One way to thwart this purely mathematical attack is to construct the algorithm so that each plaintext bit is a sufficiently complex mathematical function of the ciphertext and plaintext. If the mathematical equations describing the algorithms operation are so complex that, an analytical attack cannot be successful, then a work factor for this method cannot be calculated. In that case, one usually says that the algorithm cannot be broken in the practical sense.

Walsh functions and Walsh transforms have a wide range of applications to signal processing, image processing and communication as well as logic design and analysis. Rueppel have investigated the applications of spectral techniques to cryptology by analyzing 2.nd S-box of DES (Data Encryption Standard) algorithm. Presentation of Walsh transform can be given as follows: Let w and z are two vectors in $GF(2)^n$. For all w and z in $GF(2)^n$, the Walsh functions are defined as: $Q(w, z) = (-1)^{w \cdot z}$ where $w \cdot z = w_1 z_1 \oplus \dots \oplus w_n z_n$, and $\oplus$ denotes modulo-2 addition (exclusive-or operation). The Walsh transform is defined as $S_f(w) = 2^{-n} \sum_{z \in GF(2)^n} (-1)^{f(z)} Q(w, z)$. The following formula about how to determine the best affine approximation (BAA) of Boolean functions was developed by Rueppel with an information approach $P_f(wx) = \frac{1}{2} + \frac{1}{2} S_f(w)$. BAA is used to find affine approximation to a nonlinear updating function of the cellular automata at the proposed zigzag algorithm.

Cellular Automata were introduced by John von Neuman, following a suggestion of Ulam, to provide a more realistic model for the behavior of complex systems.

A one-dimensional cellular automaton consists of a linearly connected array of n cells, each of which takes the value of 0 or 1, and a Boolean function $f(x)$ with q variables. The value of the cell x_i is updated in parallel (synchronously) using this function in discrete time steps as $x_i' = f(x)$ for $i = 1, 2, \dots, n$. The boundary conditions are usually handled by taking the index values modulo n , i.e., the linearly connected array is actually a circular register. The parameter q is usually an odd integer, i.e., $q = 2r + 1$, where r is often named the radius of the function $f(x)$; the new value of the i th cell is calculated using the value of the i th cell itself and the values of r neighboring cells to the right and left of the i th cell. The cellular automaton for $q=3$ is illustrated in Figure 1.

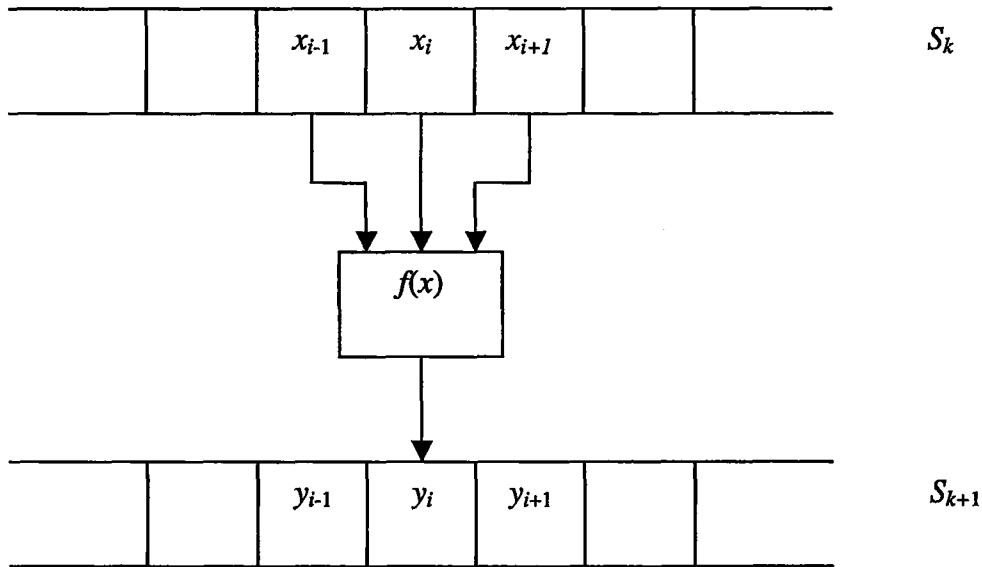


Figure 1: One-dimensional cellular automata with the radius one updating function.

Since there are n cells, each of which takes the values of 0 or 1, there are 2^n possible state vectors. Let S_k denote the state vector at the time step k . Starting from an initial state vector S_0 , the cellular automaton moves to the states S_1, S_2, S_3 , etc., at time steps $k = 1, 2, 3$, etc. The state vector S_k takes values from the set of n -bit binary vectors as k advances, and the state machine will eventually cycle, i.e., it will reach a state S_{k+P} which was visited earlier $S_k = S_{k+P}$. The period P is a function of the initial state, the updating function, and the number of cells. The cells alter their states synchronously on discrete time steps according to the local rule f of the CA. The global rule is a function from S_k to S_{k+1} that gives the new state of each cell as a function of the old states of its neighbors. A cellular automaton is invertible if its global map is invertible, i.e., if updating function is one-to-one and onto.

The construction of seed value of CA (the initial state vector S_0) from the given sequence of state vector must be difficult for cryptographic applications. It was stated by the Wolfram that this problem is in the class NP. No systematic algorithm for its solution is currently known that takes a time less than exponential in n . The cellular automaton chosen, known as rule 30, seems according to the numerical evidence to generate temporal sequences that have a high degree of randomness is shown by Wolfram.

Brief Description of the Proposed Zigzag Algorithm:

The proposed algorithm is named as zigzag algorithm because of iterations. The zigzag algorithm is based on the best affine approximation of the q -variable updating function $f(x)$. Using tools from the spectral analysis of Boolean functions a q -variable linear function $g(x)$ that is the best affine approximation to $f(x)$ is constructed. In other words, $g(x)$ has the minimum Hamming distance to $f(x)$ among all linear functions. The Hamming distance between two functions is defined as the

Hamming distance between the binary vectors of length 2^q produced by the application of all possible input values $x \in Z_2^q$ to these functions.

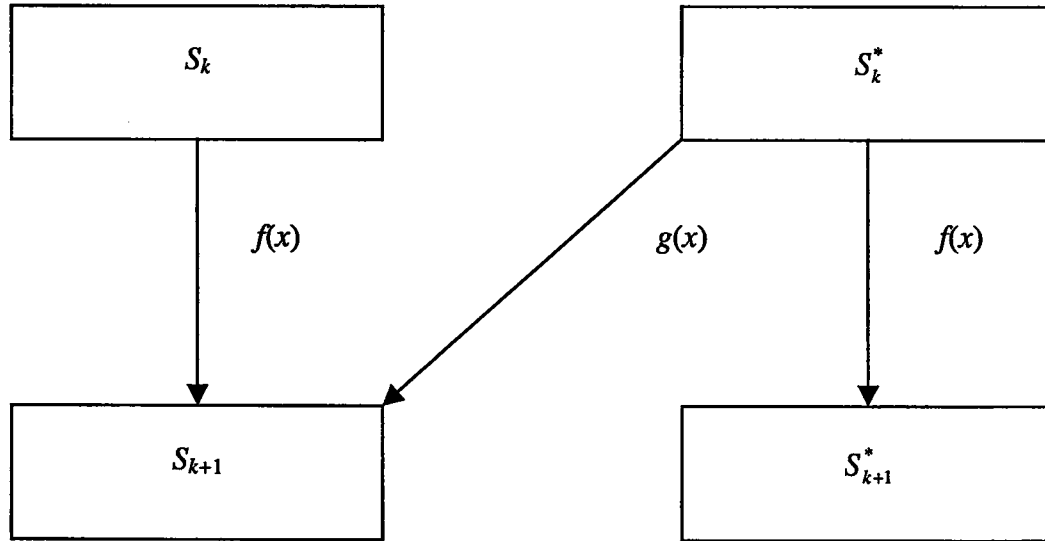


Figure 2. Schematic representation of the first four steps of zigzag algorithm.

The main idea of the zigzag algorithm is illustrated in Figure 2. The state vector S_{k+1} is computed by applying the function $f(x)$ to the state vector S_k . An approximation to S_k is calculated by finding a vector S_k^* that maps to S_{k+1} under the linear function $g(x)$. The vector S_k^* is computed by solving a set of linear equations whose matrix is determined by the parameters of the linear function $g(x)$. Then $f(x)$ is applied to S_k^* and calculated S_{k+1}^* which is an approximation to the state vector S_{k+1} . Since S_{k+1} is known, S_{k+1} and S_{k+1}^* are compared, and corrections to the state vector S_k^* is made until the state vector S_k^* which results in the equality $S_{k+1}^* = S_{k+1}$ is obtained. The zigzag algorithm calculates one of the predecessors of the state vector S_{k+1} .

Analysis of the Zigzag Algorithm:

The time complexity is the basic criteria for success of the zigzag algorithm. The time complexity of zigzag algorithm must be smaller than exhaustive search for usefulness. The order of exhaustive search can be given as $O(2^n)$ for size n CA.

Since each updating function defines a new system, analytic derivation cannot be given, hence only a loose upper bound can be given up to step 11 of zigzag algorithm. Analytical upper bound is derived as $O((1+\alpha)^n)$ for up to step 11 of the zigzag algorithm. Consequently, whole algorithm (up to step 12) execution time can be derived empirically by the help of computer simulation and curve fitting techniques. The time complexity of the zigzag algorithm is exponential. Hence, $\alpha 2^{\beta n}$ is chosen nonlinear model. Since chosen model has a property that is known as intrinsically linear, a linear regression method can be used by the help of some

transformation, hence empirical average complexity is found as $O(2^{0.656n-8.9658})$. This complexity is approximately square root of exhaustive search.

Zigzag algorithm has three different halting steps, which are step 4, step 11, and step 12. The zigzag algorithm finds solution and stops at one of these steps. The halting step depends on r , n , updating rule and S_k . The step 4 is essential, since the zigzag algorithm finds solution in polynomial time complexity, $O(nq^2)$, instead of exponential time complexity in this step. It is also fact that if the updating function is affine, the solution can be found in polynomial time complexity by just solving linear equations, but it is not true for nonlinear updating functions. Computer experiments show that 36.19 % of all configurations can be found in $O(nq^2)$ time for size 13 although expected success rate is 3.181 %, which shows that experimental success rate is approximately 10 times larger than the expected success rate.

Applications of Proposed Zigzag Algorithm:

The applicability of proposed zigzag algorithm is depending how the CA is used on the construction of encryption algorithm.

The concept of the zigzag algorithm can be applied to a variety of cryptographic systems, for instance, the proposed zigzag algorithm was applied to two different cryptographic algorithms: The first one was Wolfram's pseudorandom number generator. It was shown that, the security of Wolfram generator was reduced further. The second one was about Damgard cryptographic hash function. It was shown that collision values could be calculated by the help of the zigzag algorithm.

The given zigzag algorithm is useful for some cryptanalytic attacks; moreover the concept of the zigzag algorithm can be generalized for cryptographic systems, which are based on different algebraic systems.

HÜCRESEL OTOMATALI ŞİFRELEME SİSTEMLERİ İÇİN YENİ BİR ŞİFREANALİZ ÖNERİSİ

ÖZET

Şifreleme haberleşme ağlarıyla gönderilen veya elektronik ortamlarda depolanan bilgileri korumayla ilgilenen bilimsel bir disiplindir. Aynı zamanda mesajın orijinallliğini kontrol etmek, dijital imza üretmek, para transferi için elektronik kimlik oluşturmak, kredi kartı bilgileri göndermek ve programları virüslerden korumak için kullanılır. Şifrelemedeki temel problem açık mesajları şifreli mesajlara dönüştürecek ve şifre analize (şifrelenmiş mesaja ulaşarak açık mesajı elde etme tekniği) dayanabilecek bir yöntem geliştirmektir.

Şifreleme dünyası şifreleme algoritmaları için bir güvenlik ölçütü araya gelmiştir. Claude Shannon ilk kez şifreleme sistemleri için bu amaçla bir matematiksel model vermiştir. Şifre analiz, anahtara sahip olmadan açık mesaja ulaşmak için yapılan çalışmaların tümüdür. Genel kabul şifre analizcinin kullanılan şifreleme sisteminin tüm detaylarını anahtar hariç bildiği şeklindedir. Bu Kerckhoff kuralı olarak bilinmektedir. Eğer şifre analizci kullanılan şifre sistemini bilmiyor ise işi güçleşecektir. Ancak şifreleme sisteminin güvenliği buna dayandırılmaz. Böylece şifreleme sistemleri Kerckhoff kuralı esas alınarak tasarlanır. Yapılabilecek saldırılar şu şekilde sınıflanabilir:

1. Şifreli metin saldırısı (Ciphertext Only-Attack): Şifre analizci sadece şifreli metine sahiptir.
2. Bilinen açık metin saldırısı (Known-Plaintext Attack): Şifre analizci aynı anahtar için bazı açık ve şifreli metin çiftlerini bilir.
3. Seçilmiş açık metin saldırısı (Chosen-Plaintext Attack): Şifre analizci geçici olarak şifreleme cihazına erişebilir ve kendi seçmiş olduğu bazı açık metinler için şifrelenmiş metinler elde edebilir.
4. Seçilmiş şifreli metin saldırısı (Chosen-Ciphertext Attack): Şifre analizci geçici olarak şifre çözme cihazına erişebilir ve seçtiği bazı şifrelenmiş metinler için açık metinler elde edebilir.

İş yükü (work factor), bir algoritmayı kırmak için gereken bütün analizlerin bir ölçütüdür. İş yükünü tanımlamak için belirlenmiş bir kurallar kümesi yoktur. Genel olarak ölçüt şifre analiz için geçen süre ve gerekli matematiksel işlem adedi veya gerekli donanım ve yazılımların miktarı ile verilebilir. Gerekli olan para miktarı da genel bir karşılaştırma ölçütü olarak alınabilir.

Eksiksiz şifre analiz (exhaustive search) metodunda, anahtar veya açık mesaj mümkün bütün çözümler tek tek denenerek bulunur. Elde şifrelenmiş ve buna karşı düşen açık mesaj çifti varsa, mümkün bütün anahtarlar şifrelenmiş mesaj eldeki doğru mesajı verinceye kadar denenerek doğru anahtar bulunur. Elde sadece şifrelenmiş mesaj varsa, anlamlı bir mesaja ulaşıncaya kadar bütün anahtarlar sırasıyla denenerek aday anahtar bulunur.

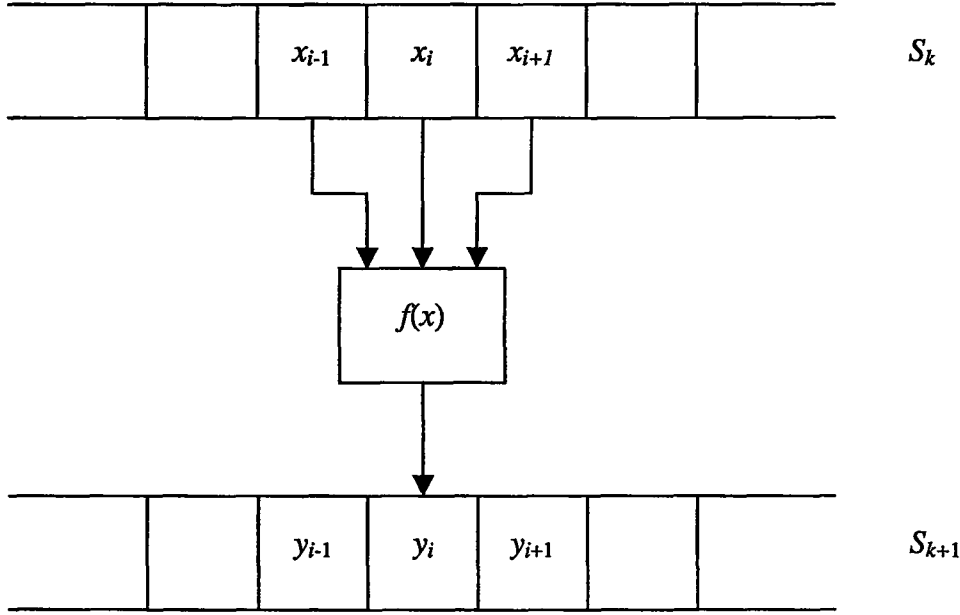
Eksiksiz şifre analizi için iş yükü deneme sayısı ile doğrudan orantılıdır. Bu tür saldırı yapılacak deneme sayısı çok fazla yapılarak engellenir. Bir diğer tür şifre analizi ise şifre algoritmasından faydalanarak bazı parametrelere göre açık mesajı veya anahtarı (veya anahtarın bir kısmını) veren matematiksel denklemlerin çıkarılmasıdır. Bunu önlemek için bu tür denklemler oluşturulduğunda bunların çok karmaşık olmasını sağlayacak şekilde şifre algoritması tasarlanmakta, böylece algoritmanın analiz edilmesi pratikte mümkün olmamaktadır.

Walsh dönüşümü ve fonksiyonları, işaret işleme, görüntü işleme, haberleşme, lojik tasarım ve analiz konularında geniş bir biçimde kullanılmıştır. Daha sonraları bu uygulama alanlarına şifreleme de katılmış, ilk kez Rueppel tarafından DES isimli şifreleme algoritmasının S-kutularından ikincisine yaklaşık bir fonksiyon bulmak amacıyla kullanılmıştır. Bu tezde bir şifre analiz aracı olarak kullanılan en iyi doğrusal yaklaşımın (best affine approximation - BAA) tanımı Walsh fonksiyonları esas alınarak aşağıda verilen şekilde hesaplanmaktadır. Walsh fonksiyonu, iki vektör $w, z \in GF(2^n)$ (GF Galois field) için $Q(w, z) = (-1)^{w \cdot z}$, $w \cdot z = w_1 z_1 \oplus \dots \oplus w_n z_n$, şeklinde tanımlanır ($\oplus$ işlemi modulo 2 toplamayı göstermektedir). Her hangi bir boole fonksiyonunun Walsh dönüşümü $S_{(f)}(w) = 2^{-n} \sum_{z \in GF(2)^n} (-1)^{f(z)} Q(w, z)$ şeklinde

ifade edilebilir ve bu gösterilim kullanılarak bu boole fonksiyonuna en yakın affine (doğrusal ve doğrusal fonksiyonların komplementlerinin oluşturduğu küme) fonksiyon, $P_f(w, x) = \frac{1}{2} + \frac{1}{2} S_{(f)}(w)$, $w, z \in GF(2)^n$ şeklinde bulunabilir. Bu tezde de affine yaklaşıklık, CA yerel fonksiyonuna (local function) doğrusal yaklaşıklık elde etmek için kullanılmıştır.

Hücreli otomata (Cellular automata-CA), ilk kez Stanislaw Ulam'ın önerileri ardından, karmaşık sistemlerin davranışlarını modellemek için, John Von Neuman tarafından ortaya atılmıştır. Bir boyutlu CA, doğrusal bir dizi şeklinde birbirine bağlanmış n hücreden oluşmuş olup her hücre 0 veya 1 değerini alır. $q=2r+1$ değişkenli bir boole fonksiyonu olan $f(x)$, hücre değerlerini günceller (şekil 1). Her hücre değeri paralel ve senkron olarak, $x_{t+1} = f(x_t)$ fonksiyonu ile de ifade edildiği gibi, $t=1,2,\dots,n$ değerleri için ayrı zamanda güncellenir. Sınır değerleri, indeksin mod n 'e göre hesaplanması ile elde edilir, yani, doğrusal olarak birbirine bağlanmış bu dizi, aslında, dairesel bir kaydedicidir (register). $r, f(x)$ fonksiyonun çapı olarak adlandırılmaktadır. Her hangi bir hücrenin yeni değeri hesaplanırken o hücrenin kendi değeri ve r komşuluğunda olan q hücrenin değeri kullanılır. $q=3$ için CA aşağıdaki şekilde gösterilmiştir.

Mümkün n hücre olması ve her birinin 0 veya 1 değeri alabilmesi nedeni ile mümkün 2^n tane durum vektörü vardır. S_k k . andaki durum vektörünü gösterebilir. Bu durumda CA, $k=0$ 'da S_0 başlangıç vektöründen başlayarak $k = 1, 2, 3$, için S_1, S_2, S_3 ,



Şekil 1: 1 boyutlu, 1 yarıçaplı fonksiyonlu hücresel otomata.

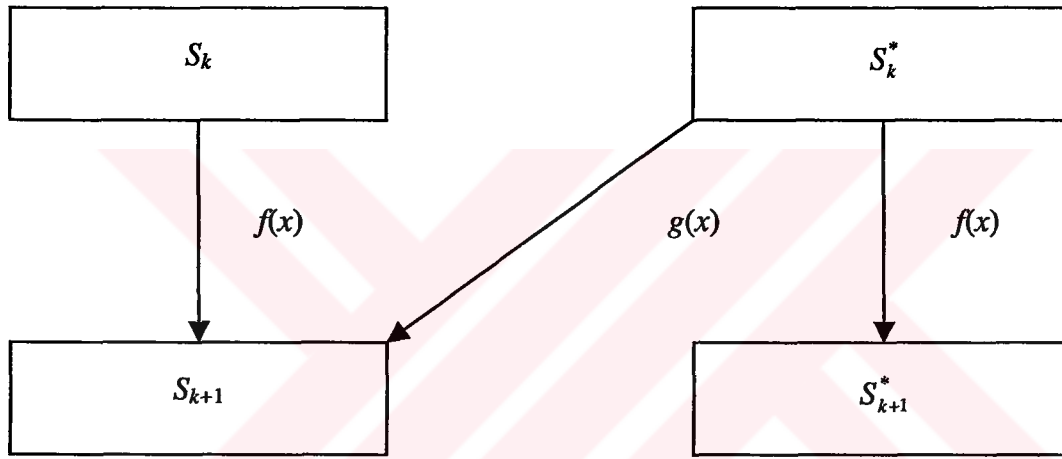
vektörlerini izleyecektir. Bu sistemin zaman içindeki iterasyonu doğal olarak bir çevrime girecektir, $S_k = S_{k+P}$. Burada, P periyodu göstermekte olup, $f(x)$ fonksiyonunun, başlangıç koşulunun ve hücre sayısının bir fonksiyonudur. $f(x)$ fonksiyonuna yerel fonksiyon (local map), S_k durumunu (state) S_{k+1} durumuna götüren fonksiyona genel fonksiyon (global map) adı verilir. Açıktır ki, CA'nın genel fonksiyonu 1-1 ve üzerine (onto) ise, CA'nın tersi vardır, dolayısıyla, ICA'dır.

CA kullanılarak oluşturulmuş üreteçlerin şifreleme amacı ile kullanılabilmesi için başlangıç değerinin (başlangıç durum vektörü S_0) verilen durum vektörlerinden bulunması güç olmalıdır. Wolfram, bu problemin NP-tam (nondeterministic polynomial) sınıfından olduğunu ve şu ana kadar n 'ye üstel (exponansiyel) olarak bağlı olan algoritmalarından daha iyi bir algoritma bulunamadığını göstermiştir. 30 numaralı kurala sahip CA rasgele sayı üretici olarak ilk kez Wolfram tarafından önerilmiştir. Bu sistem yardımı ile üretilen durum vektörlerinin, rasgelelik özelliklerini sağladığı görülmüştür.

Önerilen Zikzak Algoritmasının Tanımlanması:

Önerilen algoritma yaptığı iterasyonlar nedeniyle zikzak algoritması olarak isimlendirilmiştir. Zikzak algoritması, q -değişkenli $f(x)$ fonksiyonunun en iyi doğrusal yaklaşıklığını (Best Affine Approximation-BAA) kullanmaktadır. $f(x)$ fonksiyonunun BAA'sı olan q -değişkenli affine $g(x)$ fonksiyonu, boole fonksiyonlarının spektral analiz araçları kullanılarak elde edilebilir. Bir başka deyişle, $g(x)$, $f(x)$ ile doğrusal fonksiyonlar arasındaki en düşük Hamming uzaklığına sahip olan affine fonksiyondur. İki fonksiyon arasındaki Hamming uzaklığı bu tezde şöyle verilmiştir: Fonksiyonların mümkün bütün girişlerinin, $x \in Z_q$, bu fonksiyonlara uygulanmasıyla elde edilen 2^q uzunluklu vektörlerin Hamming uzaklığına, fonksiyonların Hamming uzaklığı denir.

Önerilen zikzak algoritması şekil 2 yardımıyla şöyle açıklanabilir. Durum vektörü S_{k+1} , $f(x)$ fonksiyonunun S_k vektörüne uygulanması sonucunda oluşturulmuştur. Amaç S_k vektörünü bulmaktır, ancak, $f(x)$ genel halde doğrusal olmayan bir fonksiyon olup, tüm çözümleri tek tek denemek dışında analitik bir çözüm bilinmemektedir. Algoritmada ilk olarak $f(x)$ yerine, $f(x)$ 'in en iyi doğrusal yaklaştığı olan $g(x)$ doğrusal fonksiyonu kullanılarak, S_k vektörüyle yaklaşık olarak aynı olan S_k^* vektörü hesaplanır. Bu işlem $g(x)$ vektörü kullanılarak oluşturulan bant matrisin tanımladığı doğrusal denklem sisteminin, S_{k+1} kullanılarak çözülmesiyle kolayca gerçekleşir. Ancak elde edilen S_k^* vektörünün geçerli bir çözüm olduğunun kontrol edilmesi, yani $f(x)$ uygulandığında S_{k+1} 'i üretmesi, yani, $S_{k+1}^* = S_{k+1}$ olması gerekmektedir. Aksi durumda, geçerli bir çözüm üretilmemiş olup, elde edilen S_k^* 'da hata düzeltilmesi yapılmalıdır. Bu düzeltme, S_{k+1}^* ve S_{k+1} 'in aynı olmayan noktaları dikkate alınarak, S_k^* üzerinde yapılır. Bu işlemin sonucunda geçerli bir S_k^* elde edilir.



Şekil 2: Algoritmanın bir kısmının şematik gösterilimi.

Önerilen Zikzak Algoritmasının Analizi:

Zaman karmaşıklığı bu şifre analiz metodu için temel ölçüttür. Önerilen zikzak algoritmasının anlamlı olması için zaman karmaşıklığının, eksiksiz denemenin zaman karmaşıklığından daha az olmalıdır. Eksiksiz deneme n genişlikli bir CA için $O(2^n)$ şeklinde verilir.

Her güncelleme fonksiyonu yeni bir sistem tanımladığı için analitik bir karmaşıklık fonksiyonu vermek çok güçtür. Bu nedenle ancak 11. adıma kadar olan kısım için bir üst sınır verilebilmiştir. Analitik üst sınır $O((1+\alpha)^n)$ $0.5 \leq \alpha \leq 1$ şeklinde verilebilmiştir. Tüm algoritmanın karmaşıklık fonksiyonu, algoritmanın farklı n değerleri için bilgisayar üzerinde koşturulması ve bu değerlere uygun eğrinin kestirilmesiyle hesaplanmıştır. Bu hesaplamada $\alpha 2^{\beta x}$ doğrusal olmayan model olarak seçilmiştir. Bu modelin doğası gereği doğrusal (intrinsic linear) olarak bilinen özelliğe sahip olması nedeniyle, doğrusal basitleştirme (linear regression) metodu

kullanılabilmiş ve tüm algoritmanın zaman karmaşıklığı $0.002 \times O(2^{0.656n})$ olarak hesaplanmıştır. Bu değer $O(2^n)$ 'nin yaklaşık karekökü kadar olup oldukça başarılı bir sonuçtur.

Önerilen zikzak algoritması 12 adıma sahip olup 4'üncü, 11'inci veya 12'inci adımlardan birinde sonlanıp çözümü vermektedir. Durma noktası r , n , güncelleme fonksiyonuna ve S' 'e bağlıdır 4. adım oldukça önelidir, çünkü algoritmanın dördüncü adımıda çözümü bulması durumunda zaman karmaşıklığı üstel yerine polinomsal yani $O(nq^2)$ olmaktadır. Güncelleme fonksiyonlarının bir kısmının zaten doğrusal olduğu ve bunların çözümünün doğrusal denklem takımı çözülerek bulunacağı bir gerçektir. Bilgisayar deneyleri göstermiştir ki, çözüm kümesinin, % 36.19'u $O(nq^2)$ zaman karmaşıklığında çözülmüştür. Oysaki beklenen değer % 3.181 olup, bilgisayar deneyleriyle elde edilmiş sonuçların 10 da biridir.

Önerilen Zikzak Algoritmanın Uygulamaları:

Önerilen algoritmanın uygulanabilirliği CA'nın şifreleme sisteminde kullanılış şekline bağlıdır.

Bu tezde önerilen şifre analiz ana fikri pek çok tipte şifreleme algoritmasına uygulanabilir, örneğin bu tezde verilmiş olan şifre analiz metodu iki ayrı CA tabanlı şifreleme sistemine uygulanmıştır. Bunlardan ilki Wolframın sanki rasgele (pseudorandom) sayı üreticidir. Bu algoritma daha zayıflatılmıştır. İkincisi de Damgard'ın hash fonksiyonudur. Damgard özet (hash) fonksiyonunda çarpışma (collision) değerlerinin algoritmik olarak bulunabileceği gösterilmiştir.

Önerilen algoritma bazı şifre analiz çalışmaları için oldukça etkindir; ayrıca, önerilen algoritmanın CA dışındaki farklı cebrik yapılara da genişletilebilme olasılığı vardır.

1. INTRODUCTION

Today, vast amounts of digital data are gathered and stored in large computer databases, and transmitted between the computers and terminal devices linked together in complex communications networks. Without appropriate safeguards, these data are susceptible to interception and modification during transmission, or they may be physically removed or copied while in storage. Among the security measures, cryptography is the one that should be considered, and it involves methods for rendering data unintelligible to unauthorized parties.

Cryptographic algorithms consist of extremely large number of transformations. Each transformation depends on the cryptographic key being used and changes sequences of intelligible data (plaintext) into sequences of apparently random data (ciphertext). The transformation from plaintext to ciphertext is known as encipherment or encryption. In Chapter 2, brief description of cryptology, encryption algorithms and the other essential terms are given. Since complexity theory is used as a metric in cryptology, an introduction to complexity theory will be given in the Chapter 2.1. Complexity theory (Garey, Johnson, 1979) introduces $O(\cdot)$ notation that is basically representing number of operations required in terms of the input size measured in bits.

During the recent years large amount of encryption algorithms are published with all details. All these encryption algorithms are designed with respect to some criteria. These criteria can be grouped in the two categories: Structural criteria (Webster, Tavares, 1986) cover all analytic calculations and statistical criteria (Knuth, 1981), (Geçkinli, Apohan, 1998) cover all measures related with randomness. Although there are a lot of cryptographic algorithms and protocols published (Schneier, 1996), there are small numbers of cryptanalysis methods available in public literature. The cryptanalysis methods are frequently kept secret by the governments. Most famous publicly known methods are linear cryptanalysis (Matsui, 1994) and differential cryptanalysis (Biham, Shamir, 1993), which are

generally used as cryptographic strength measures, rather than practical cryptanalysis methods (Massey, 1995). Definition of cryptanalysis with examples of differential and linear cryptanalysis of DES is also given briefly in Chapter 2.2.3 to present concepts of the cryptanalysis.

The applications of spectral techniques to cryptology have been investigated by Xiao and Massey (1985) as well as Siegenthaler (1986) and Rueppel (1986). By employing the best affine approximation (BAA) approach, the realization complexity of circuits can be much reduced without penalty. Its application to cryptography was first noticed by Rueppel (1986). Construction of the BAA employs Walsh based expansion of Boolean functions which are the cellular automata updating functions. BAA is used to find affine approximation to a nonlinear updating function of the cellular automata at the proposed zigzag algorithm. Necessary definitions and properties of Walsh functions and BAA are given in Chapter 2.4.

Cellular automata are discrete dynamical systems whose behavior is completely specified in terms of a local relation, much as in the case of a large class of continuous dynamical systems defined by partial differential equations. Cellular automata seem to have found a permanent and increasing role as conceptual and practical models of spatially distributed dynamical systems. A cellular automaton is defined in terms of local function, dimension, boundary condition and configuration (state or state vector). Formal definition and some properties of cellular automata are given in Chapter 3. During recent years, cellular automata found wide range of applications in the area of cryptography (Daemen, 1991), (Nandi et al, 1994). Cellular automata are used to develop symmetric key, public key encryption algorithms (Kari, 1992), (Guan, 1987) and cryptographic hash functions (Damgard, 1990). Some of the cellular automata based proposed and patented (Gutowitz, 1993) cryptographic applications are given in Chapter 3.1. The natural question arises out the security of these algorithms. In the public literature, analyses of some of these algorithms are given. However, these analyses are limited only to restricted parameter cellular automata structures. The only publicly known Meier-Staffelbach (1991) attack is also presented in Chapter 3.2. However, Meier-Staffelbach approach is applicable solely to uniform cellular automata that have partial linear and radius one updating functions. Indeed, our proposed zigzag algorithm and Meier-

Staffelbach cryptanalysis approach is applied to two different types of cellular automata applications. It is observed that coupling of these two methods create more powerful cryptanalysis method.

Cryptographic hash functions, which map larger domains to smaller ranges, play a fundamental role in modern cryptography. Cryptographic hash functions are used particularly for data integrity and message authentication. It has been known that cellular automata based hash function (Damgard, 1990) has a collision (Daemen et al, 1991), but no algorithm to calculate collision values has been given until now. Damgard hash function and its known cryptanalysis methods are presented in Section 3.1.2. In Section 5.2, proposed zigzag algorithm is also applied to Damgard hash function to calculate collision. Proposed zigzag algorithm is the first known method to calculate collision.

The proposed cryptanalysis algorithm that is called Zigzag Algorithm (Koç and Apohan, 1997) is presented in Section 4. This novel algorithm is new and can be applied to cryptographic applications of hybrid cellular automata, cellular automata with an updating function having radius greater than 1 and also to cellular automata that have dimension greater than one. The proposed algorithm is based on best affine approximation (BAA) (Ding et al, 1991) of a given nonlinear updating function of cellular automaton and use correction algorithms to correct errors, which occur due to approximation. Proposed zigzag algorithm can be summarized as follows: First, BAA of given cellular automata updating function is calculated. Then, predecessor of a given state is found by solving linear equations using calculated BAA, thus found predecessor is applied to given cellular automata, and successor state is calculated. The calculated successor and given state are compared, and if they are equal, the calculated predecessor is correct and the algorithm terminates. If they are not equal, a correction algorithm is executed. The correction algorithm constructs tables for each unmatched bit of the calculated successor and given states. Overlapped tables are reduced to single table, and rows of the reduced tables are examined in the calculated predecessor. Successor is calculated for each predecessor and compared with a given state until equivalence is established. Thus, the cryptanalysis algorithm finds cellular automata configurations (states) backward in time.

It is known that, Backward tracing of states in any cellular automata is NP-complete (Wolfram, 1984). It has been also pointed out that there is not any effective algorithm, better than exhaustive search, which means that it has an $O(2^n)$ time complexity function. Hence, it must be shown that the proposed zigzag algorithm must have smaller time complexity than exhaustive search for effectiveness. The time complexity function of the zigzag algorithm is derived analytically up to step 11 in Section 4. However, wholly analytic derivation has serious difficulties. Thus, an analysis of the algorithm is accomplished empirically by using run time results on computer. The analytic bound is found for up to step 11 as $O((1+\alpha)^n)$, where $0.5 \leq \alpha \leq 1$ and the empirical bound is given as $O(2^{0.656n-8.9658})$. Both of these bounds are smaller than that of exhaustive search, which demonstrates the success of the zigzag algorithm. Several examples of the zigzag algorithm with their halting steps are given in Section 5.

In this thesis, our zigzag algorithm and Meier-Staffelbach cellular automata cryptanalysis approach are applied together to Wolfram's pseudorandom number generator (Wolfram, 1985) and Damgard's hash function, and it is observed that security of these two cryptographic algorithms are reduced further (Section 5). Encryption key of Wolfram's pseudorandom number generator can be found by using the zigzag algorithm. It has been also shown that the zigzag algorithm finds collision values at the Damgard's cellular automata based hash function.

2. CRYPTOGRAPHIC TERMINOLOGY AND CONCEPTS

Cryptography is a scientific discipline that is concerned for the protection of information transmitted through communications networks or stored in electronic media. It can also be used for message authentication, digital signature and personal identification for authorizing electronic funds transfer, credit card transactions, protecting programs against viruses. A basic problem in cryptography is devising procedures to transform messages (plaintext) into cryptograms (ciphertext) that can withstand intense cryptanalysis (the techniques used by opponents to penetrate encrypted communications and recover the original information).

The cryptographic algorithm can be thought of as an extremely large number of transformations, the particular transformation in effect depending on the cryptographic key being used. Each transformation change sequences of intelligible data (plaintext) into sequence of apparently random data (ciphertext). The transformation from plaintext to ciphertext is known as encipherment or encryption. Each transformation must have unique inverse operation, also identified by cryptographic key. The inverse transformation from ciphertext to plaintext is called decipherment or decryption.

An encryption system (figure 2.1) is a five-tuple (P, C, K, E, D) , where the following conditions are satisfied:

1. P is a finite set of all possible plaintext.
2. C is a finite set of all possible ciphertext.
3. K , the key space, is a finite set of all possible keys.
4. For each $k \in K$, there is an encryption rule $e_k \in E$ and a corresponding decryption rule $d_k \in D$. Each $e_k: P \rightarrow C$ and $d_k: C \rightarrow P$ are functions such that $d_k(e_k(p)) = p$ for every plaintext $p \in P$.

The main condition is condition 4. It says that if a plaintext p is encrypted using e_k , and the resulting ciphertext is subsequently decrypted using d_k , then the original plaintext p results.

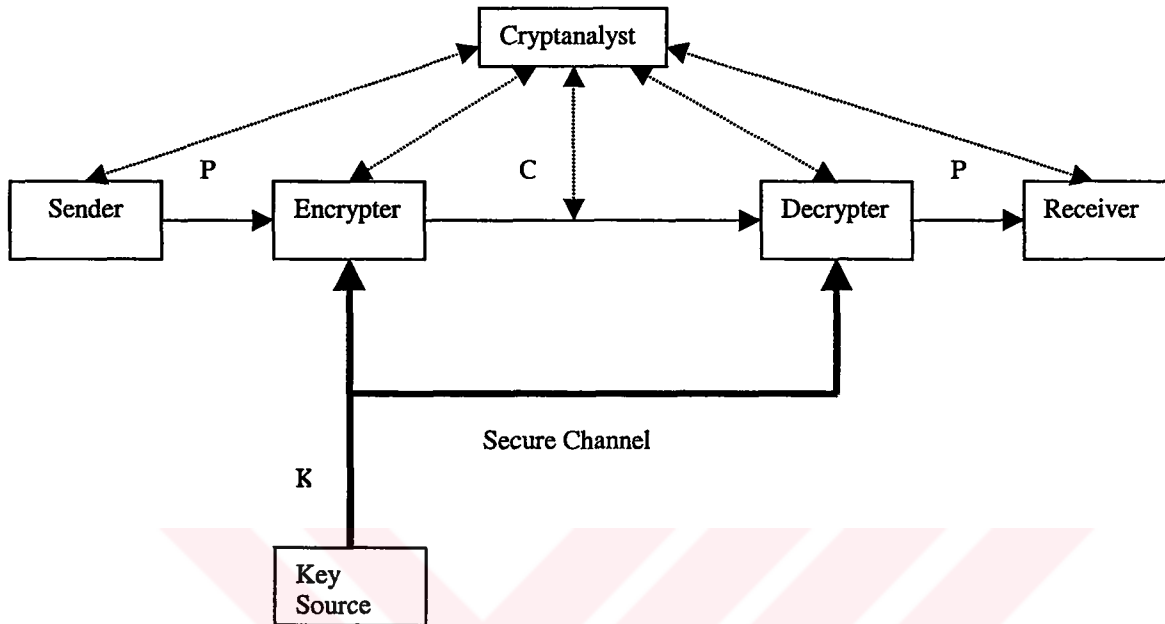


Figure 2.1. The general scheme of symmetric cryptographic system.

Sender and receiver will employ the following protocol to use a specific cryptosystem. First, they choose a random key $k \in K$. This is done when they are in the same place and are not being observed by cryptanalyst, or alternatively, when they do have access to a secure channel, in which case they can be in different places. Later, sender encrypts P by using predetermined key k . Hence sender computes $e_k(P) = C$, the resulting ciphertext. Receiver decrypts it using the decryption function $d_k(C) = P$, obtaining original plaintext. Clearly, it must be the case that each encryption function is an one-to-one (1-1) function.

2.1. Complexity Theory

Every problem has a different running time on the computer for the solution. The running time of the algorithm gives the difficulty of it. Complexity theory

(Garey, Johnson, 1979) classifies problems with respect to difficulty of finding the solution.

The time complexity function for an algorithm expresses its time requirements by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size. Of course, this function is not well defined until one fixes the encoding scheme to be used for determining input length and the computer or computer model to be used for determining execution time. However, as we shall see, the particular choices made for these will have little effect on the broad distinctions made in the theory of NP-completeness.

Different algorithms possess a wide variety of different time complexity functions, and whether the characterizations of which are "too inefficient" will always depend on the situation at hand. However, computer scientists recognize a simple distinction that offers considerable insight into these matters. This is the distinction between polynomial time algorithms and exponential time algorithms. Thus, the required comparison between complexity functions of algorithms must be defined in terms of $O(\cdot)$ notation. Definition of $O(\cdot)$ can be given as follows: A function $f(n)$ is $O(g(n))$ whenever there exist a constant c such that $|f(n)| \leq c \cdot |g(n)|$ for all values of $n \geq 0$. A polynomial time algorithm is defined to be one whose time complexity function is $O(p(n))$ for some polynomial function $p(n)$, where n is used to denote input length. Any algorithm whose time complexity function cannot be so bounded is called an exponential time algorithm (although it should be noted that this definition includes certain non-polynomial time complexity functions, like $n^{\log n}$, which are not normally regarded as exponential functions).

The distinction between these two types of algorithms has particular significance when considering the solution of large problem instances. Table 2.1. illustrates the differences in growth rates among several typical complexity functions of different types, where the functions express execution time in terms of microseconds. The much more explosive growth rates for the two exponential complexity functions are noticeable

Table 2.1. Execution time of different problems, which have different complexity functions. All results are given for the same computer, sec represents second, min represents minute, centr represents century, repeatedly (Garey and Johnson, 1979).

		Size (n)					
		10	20	30	40	50	60
$f(n)$	n	.00001 sec.	.00002 sec.	.00003 sec.	.00004 sec.	.00005 sec.	.00006 sec.
	n^2	.0001 sec.	.0004 sec.	.0009 sec.	.0016 sec.	.0025 sec.	.0036 sec.
	n^3	.001 sec.	.008 sec.	.027 sec.	.064 sec.	.125 sec.	.216 sec.
	n^5	.1 sec.	3.2 sec.	24.3 sec.	1.7 min.	5.2 min.	13 min.
	2^n	.001 sec.	1 sec.	17.9 min.	12.7 day	35.7 years	366 centr.
	3^n	0.59 sec.	58 min.	6.5 years	3855 centr.	$2 \cdot 10^8$ centr.	$1.3 \cdot 10^{13}$ centr.

Even more revealing is an examination of the effects of improved computer technology on algorithms having these time complexity functions. Table 2.2. shows how the largest problem instance solvable in one hour would change if a computer is available 100 or 1000 times faster than our present machine. It can be observed that with the 2^n algorithm a thousand-fold increase in computing speed only adds 10 to the size of the largest problem instance solvable in hour, whereas with the n^5 algorithm this size almost quadruples. These tables indicate some of the reasons why polynomial time algorithms are generally regarded as being much more desirable than exponential time algorithms.

Table 2.2. Effect of improved technology on several polynomial and exponential time algorithms (Garey and Johnson, 1979).

Size of Largest Problem Instance Solvable in 1 Hour			
Time complexity function	With present computer	With computer 100 times faster	With computer 1000 times faster
n	n	$100 n$	$1000 n$
n^2	n	$10 n$	$31.6 n$
n^3	n	$4.64 n$	$10 n$
n^5	n	$2.5 n$	$3.98 n$
2^n	n	$n + 6.64$	$n + 9.97$
3^n	n	$n + 4.19$	$n + 6.29$

Most exponential time algorithms are merely variations on exhaustive search, whereas polynomial time algorithms generally are made possible only through the gain of some deeper insight into the structure of a problem. Furthermore, some exponential time algorithms have been quite useful in practice. Time complexity as defined is a worst-case measure, and the fact that an algorithm has time complexity 2^n means only that at least one problem instance of size n requires that much time. Most problem instances might actually require far less time than that, a situation that appears to hold for several well-known algorithms. The simplex algorithm for linear programming has been shown to have exponential time complexity (Zadeh, 1973) but it has an impressive record of running quickly in practice. Unfortunately, examples like these are quite rare (Garey and Johnson, pp. 9, 1979).

2.2. A Brief Definition and Concept of Cryptanalysis

Obtaining plaintext from the ciphertext without having encryption key and/or algorithm is called cryptanalysis. Thus, cryptanalyst tries to find encryption key or directly plaintext. Any procedure for attacking a cryptographic algorithm requires that certain cryptographic information (such as encryption algorithm, ciphertext, plaintext and corresponding ciphertext) be available to carry out the attack. Therefore, the set of procedures that can be used to attack an algorithm depends on the information available to an adversary. Knowing cryptographic information and how the opponent might reasonably be able to obtain the plaintext is thus the basis for determining the class of attacks that the algorithm must be designed to resist.

The general assumption that is usually made is that the opponent, the cryptanalyst, knows the cryptosystem with all details being used but not the key. This is usually referred to as Kerckhoff's principle (Meyer and Matheas, 1982). Of course, if cryptanalyst does not know the cryptosystem being used, that will make his task more difficult. However, the security of a cryptosystem is not based on the assumption that cryptanalyst does not know what system is being employed. Hence, cryptosystem is designed under Kerckhoff's principle.

Different types of attacks can be enumerated as follows:

1. Ciphertext Only: The opponent possesses a string of ciphertext, c .
2. Known Plaintext: The opponent possesses a string of plaintext p , and corresponding ciphertext c .
3. Chosen Plaintext: The opponent has obtained temporary access to the encryption machinery. Hence, he can choose a plaintext string, p , and construct the corresponding ciphertext string, c .
4. Chosen Ciphertext: The opponent has obtained temporary access to the decryption machinery. Hence, he can choose a ciphertext string, c , and construct the corresponding plaintext string, p .

2.2.1. Perfect Secrecy and Shannon Measure on Security

The world of cryptology has searched a measure for the security of the encryption algorithm. Shannon is the first person that gave the mathematical model for the security of cryptographic system. Shannon considered two very different notions of security for cryptographic systems (Massey, pp. 2.20, 1995). He first considered the question of theoretical security, by which he meant, "How secure is a system against cryptanalysis when the enemy has the unlimited time and labor available for the analysis of intercepted cryptograms?"

Definition (Shannon, 1949): A secret key cryptosystem is perfect if it is unconditionally secure against a cipher-text only attack when the key is used only one time.

$$\text{PERFECT SYSTEM} \Leftrightarrow H(P) = H(P|C) \quad (2.1)$$

where $H(\cdot)$ denotes entropy function. Perfect system practically exists which is known as one-time pad. This system requires the secure distribution of as many digits of secret key as there are digits of plaintext to be enciphered.

Shannon's theory of theoretical security, casts much light into cryptology, but leads to be pessimistic conclusion that the amount of secret key needed to build a theoretically secure cipher will be impracticably large for most applications that is known as one-time pad.

All but few cryptographic applications, such as encrypting the MOSCOW - WASHINGTON hotline, where the need for secrecy is paramount and the amount of plaintext is quite limited, is based on one-time pad. The WASHINGTON - MOSCOW hotline will not be encrypted with a one-time pad, alternatively a conventional secret-key cipher that requires much less key will be used (Meyer and Matyas, 1982).

Shannon also treated the question of practical security by which he meant: Is the system secure against a cryptanalyst who has a certain limited amount of time and computational power available for the analysis of intercepted cryptograms? To implement a cryptographic algorithm conveniently, the key must be fixed in length,

relatively short and capable of being used repeatedly without weakening security. However, an algorithm that uses the finite key can, theoretically always be broken. The only question concerns how much work and resources the opponent must expend. Fortunately, it is not necessary to implement unbreakable algorithms if the work (work factor) required to break the algorithm is sufficiently great to discourage an opponent from attacking it. Work factor measures what is needed to carry out a specific analysis or attack against a cryptographic algorithm. The attack is conducted under a given set of assumptions, which includes the information available to achieve a predetermined goal such as recovery of the plaintext or key. A good cryptographic algorithm design maximizes the amount of work that an opponent must expend to break it. Thus, for a given algorithm and set of assumptions, the work factor is an expression of the minimum amount of work necessary for successful attack. In practice, there is no universally accepted, fixed set of parameters used to express the work factor. However, it is frequently measured in one or more of the following: cryptanalyst hours, number of mathematical or logical operations, computing resources, necessary special hardware and calendar time. To be useful, the work factor should be expressed using parameters, which can be translated for the purpose of comparison into a common base, such as cost in dollars.

One can study possible methods of solution available to the cryptanalyst describing them in the most general terms possible and then define a set of design rules to thwart any one these methods. An algorithm is then constructed which can resist these general methods of solution. Second, one can construct an algorithm in such a way that breaking it requires the solution of some known problem, but one that is difficult to solve. The Data Encryption Algorithm (DES) (Schneier, 1996) was designed using the first approach, whereas RSA (RIVEST and et. al. 1978) public key algorithms have been designed using the second approach. Both of these algorithms are very well known and implemented on different systems (Apohan, Yüzbaşıç, 1993), (Apohan, 1994).

2.2.2. Exhaustive Search

In an exhaustive attack, an attempt is made to recover the plaintext or key by using direct search methods. For example, in key exhaustion, a known plaintext is enciphered with a trial key and result is compared for equality with the known

corresponding ciphertext. If only ciphertext is available, it can be decrypted with the trial key and the resulting plaintext can be inspected to see if it makes any sense. In this way, it can be determined if the trial key is a candidate for the unknown key or not.

Exhaustive attacks can be thwarted by making the number of required trials very large. However, the work factor of an exhaustive attack, which is directly proportional to the number of trials, is easily determined even when the number of trials is so large that the attack is not feasible. This is not the case with some other attacks from a definition of the cryptographic algorithm solved for the variable or variables representing the known message or key. One way to thwart this purely mathematical attack is to construct the algorithm so that each plaintext bit is a sufficiently complex mathematical function of the ciphertext and plaintext. If the mathematical equations describing the algorithm's operation are so complex that, an analytical attack cannot be successful, then a work factor for this method cannot be calculated. In that case, one usually says that the algorithm cannot be broken in the practical sense.

The distinction between strong and unbreakable should be apparent. While in theory a strong algorithm can always be broken, in the practical sense it cannot. Unbreakable is an absolute attribute and means that even with an unlimited amount of computational power, data storage, and calendar time, speaks, strong is a variable, and unbreakable is its maximum value.

Except in rare cases, it is impossible to state absolutely that certain information will never become available to an opponent under all-operating conditions and environments in which the algorithm may be implemented. Therefore, conservative approach must be used in algorithm design. It is assumed that the opponent has a wide range of information that might be useful attacking the algorithm. The algorithm is then designed to resist all known attacks made possible by this information. Therefore, a conservative approach must likewise be used in the design of a system, such as a communication or a file security system, which implements a cryptographic algorithm. It is assumed that the opponent has knowledge of a wide range of attacks that might be capable of breaking the

algorithm. The system is then designed to deny the opponent the information needed to carry out the attacks.

In summary, the design of a strong cryptographic algorithm must satisfy the following conditions:

1. The mathematical equations describing the operation of the algorithm is so complex that, for all practical purposes, it is not possible to solve them using analytical methods.
2. The cost or time required recovering the message or key is too great when using methods that are mathematically less complicated, because too many computational steps are required or too much data storage is required.

Furthermore, it is assumed that the above conditions are satisfied even when the opponent has the following advantages:

1. Relatively large amount of plaintext (specified by the opponent, if he so desires) and corresponding ciphertext are available.
2. Relatively large amounts of ciphertext (specified by the opponent) and the corresponding plaintext are available.
3. All details of the algorithm are available. It is not assumed that cryptographic strength depends on maintaining the secrecy of the algorithm.
4. A number of large high-speed computers (determined by the resources available to the opponent) can be used for cryptanalysis.

2.2.3. Differential and Linear Cryptanalysis of DES

On May 15, 1973, the National Bureau of Standards published a solicitation for cryptosystems in the Federal Register. This led ultimately to the development of the Data Encryption Standard (Meyer, Matyas, 1982), or DES, which has become the most widely used cryptosystem in the world. DES was developed at IBM, as a modification of an earlier system known as LUCIFER. DES has been renewed by the National Bureau of Standards every five years since its adaptation. It was renewed

until 1998. It was implemented on the different microprocessor and system (Apohan, Yüzbaşıç, 1993).

A complete description of DES is given in the Federal Information Processing Standards Publication 46, dated January 15, 1977. DES encrypts a plaintext bit string p of length 64 using a key k which is a bit string of length 56, obtaining a ciphertext bit string which is again a bit string of length 64. High-level description of the system can be given as follows:

Given a plaintext p , a bit string p_0 is constructed by permuting the bits of p according to fixed initial permutation IP . $p_0 = IP(p) = L_0R_0$, where L_0 comprises the first 32 bits of p_0 and R_0 the last 32 bits.

16 iteration of a certain function is then computed. L_iR_i is computed, $1 \leq i \leq 16$, according to the following rule: $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus F(R_{i-1}, k_i)$, where $\oplus$ denotes the exclusive-or of two bit strings, details of function F can be found (MEYER, MATYAS, 1982), and $k_1, k_2, \dots, k_{16}$ are each bit string of length 48 computed as a function of the key k .

Apply the inverse permutation IP^{-1} to bit string $L_{16}R_{16}$, obtain the ciphertext c .

2.2.3.1. Differential Cryptanalysis of DES

In 1990, Biham and Shamir (1993) introduced differential cryptanalysis. This is a new method of cryptanalysis, heretofore unknown to the public. Using this method, Biham and Shamir found a chosen plaintext attack against DES that was more efficient than brute force.

Differential cryptanalysis looks specifically at ciphertext pairs: pairs of ciphertext whose plaintexts have particular differences. It analyses the evolution of these differences as the plaintexts propagate through the rounds of DES when they are encrypted with the same key.

Simply, chose pairs of plaintexts with a fixed difference. The two plaintexts can be chosen at random, as long as they satisfy particular difference conditions; the cryptanalyst does not even have to know their values. Then, using the differences in the resulting ciphertexts, different probabilities are assigned to different keys. As you

analyze more and more ciphertext pairs, one key will emerge as the most probable. This is the correct key.

If the last round of DES is considered and K_{16} is known, 48 bits of key will be found. The other 8 bits can be obtained by brute force. Differential cryptanalysis will get K_{16} .

Certain differences in plaintext pairs have a high probability of causing certain differences in the resulting ciphertext pairs. These are called characteristics. Characteristics extend over a number of rounds and essentially define a path through these rounds. There is an input difference, a difference at each round, and an output difference-with a specific probability.

These characteristics can be found by generating a table where the rows represent the possible input XORs (the XOR of two different sets of input bits), the columns represent the possible output XORs, and the entries represent the number of times a particular output XOR occurs for a given input XOR.

Table 2.3. The necessary amount of workload for differential cryptanalysis.

No. of Rounds	Chosen Plaintexts	Known Plaintexts	Analyzed Plaintexts	Complexity of Analysis
8	2^{14}	2^{38}	4	2^9
9	2^{24}	2^{44}	2	2^{32}
10	2^{24}	2^{43}	2^{14}	2^{15}
11	2^{31}	2^{47}	2	2^{32}
12	2^{31}	2^{47}	2^{21}	2^{21}
13	2^{39}	2^{52}	2	2^{32}
14	2^{39}	2^{51}	2^{29}	2^{29}
15	2^{47}	2^{56}	2^7	2^{37}
16	2^{47}	2^{55}	2^{36}	2^{37}

DES's resistance can be improved by increasing the number of rounds. Chosen plaintext differential cryptanalysis DES with 17 or 18 rounds takes about the same time as a brute force search (Schneier, 1996). At 19 rounds or more, differential cryptanalysis becomes impossible because it requires more than 2^{64} chosen plaintexts: DES has a 64-bit block size, so it only has 2^{64} possible plaintext blocks.

Differential cryptanalysis is largely theoretical attack. The enormous time and data requirements to mount a differential cryptanalytic attack put it beyond the reach of almost everyone (table 2.3.). To get the requisite data for this attack against a full DES, 1.5 megabits-per-second data stream of chosen plaintext for almost three years must be encrypted. This is primarily chosen-plaintext attack. It can be converted to a known-plaintext attack, but all of plaintext-ciphertext pairs must be sifted, looking for the useful ones. For full 16-round DES, this makes the attack slightly less efficient than brute force (the differential cryptanalytic attack requires $2^{55.1}$ operations, and brute force requires 2^{55}). The consensus is that DES, when implemented properly, is still secure against differential cryptanalysis.

Why is DES so resistant to differential cryptanalysis? Why are the S-boxes optimized to make this attack as difficult as possible? Why are there as many rounds as required, but no more? Don Coppersmith (1994) from IBM wrote

The design took advantage of certain cryptanalytic techniques, most prominently the technique of "differential cryptanalysis," which were not known in the published literature. After discussion with NSA, it was decided that disclosure of the design consideration would reveal the technique of differential cryptanalysis, a powerful technique that can be used against many ciphers.

2.2.3.2. Linear Cryptanalysis of DES

Linear cryptanalysis is another type of cryptanalytic attack, invented by Mitsuru Matsui (1994). This attack uses linear approximations to describe the action of block cipher (DES).

This means that if some of the plaintext bits and some of the ciphertext bits are XORed together, XOR of some of the key bits will be found. This is a linear

approximation and is hold with some probability p . If $p \neq \frac{1}{2}$, then this bias can be exploited. Use collected plaintexts and associated ciphertexts to guess the values of the key bits. Increasing data set will give more reliable guess and also increasing bias will give greater success rate with the same amount of data.

The basic attack is to use the best linear approximation for 16-round DES. It requires 2^{47} known plaintext blocks, and will result in 1 key bit but attack can be refined: By using 14-round linear approximation for rounds 2 through 15 and guessing the 6 subkey bits relevant to S-box 5 for the first and last rounds (12 key bits). Effectively 2^{12} linear cryptanalysis are done and the correct one picked based on probabilities. This recovers the 12 bits and b_{26} , and reversing plaintext and ciphertext recovers another 13 bits. To get the remaining 30 bit exhaustive search is used.

2.3. Cryptographic Hash Functions

Cryptographic hash functions (Pieprzyk, Sadeghiyan, 1991) play a fundamental role in modern cryptography. While related to conventional hash functions commonly used in non-cryptographic computer applications - in both cases, larger domains are mapped to smaller ranges - they differ in several important aspects. Our focus is restricted to cryptographic hash functions (hereafter, simply, hash functions) and in particular to their use for data integrity and message authentication.

Hash functions take a message as input and produce an output referred to as a hash-code, hash-result, hash-value, or simply hash. More precisely, a hash function h maps bit-strings of arbitrary finite length to strings of fixed length, say n bits. For a domain D and range R with $h:D \rightarrow R$ and $|D| > |R|$, the function is many-to-one, implying that the existence of collisions (pairs of inputs with identical output) is unavoidable. Indeed, restricting h to a domain of t -bit inputs ($t > n$), if h were "random" in the sense that all outputs were essentially equal probable, then about 2^{t-n} inputs would map to each output, and two randomly chosen inputs would yield the same output with probability 2^{-n} (independent of t). The basic idea of cryptographic hash functions is that a hash-value serves as a compact representative image

(sometimes called digital fingerprint or message digest) of an input string, and can be used as if it were uniquely identifiable with that string.

Hash functions are used for data integrity in conjunction with digital signature schemes (Pieprzyk, Sadeghiyan, 1991), where for several reasons a message is typically hashed first, and then the hash-value, as a representative of the message, is signed in place of the original message. A distinct class of hash functions, called message authentication codes (MACS), allows message authentication by symmetric techniques. MAC algorithms may be viewed as hash functions that take two functionally distinct inputs, a message and a secret key, and produce a fixed-size (say n -bit) output, with the design intent that it be infeasible in practice to produce the same output without knowledge of the key. MACS can be used to provide data integrity and symmetric data origin authentication, as well as identification in symmetric-key schemes.

A typical usage of (unkeyed) hash functions for data integrity is as follows. The hash-value corresponding to a particular message x is computed at time T_1 . The integrity of this hash-value (but not the message itself) is protected in some manner. At a subsequent time T_2 , the following test is carried out to determine whether the message has been altered, i.e., whether a message x' is the same as the original message. The hash-value of x' is computed and compared to the protected hash-value; if they are equal, one accepts that the inputs are also equal, and thus that the message has not been altered. The problem of preserving the integrity of a potentially large message is thus reduced to that of a small fixed-size hash-value. Since the existence of collisions is guaranteed in many-to-one mapping, the unique association between inputs and hash-values can be in the computational sense. A hash-value should be uniquely identifiable with a single input in practice, and collisions computational should be difficult to find (essentially never occurring in practice).

2.3.1. General Classification and Security of Hash Functions

At the highest level, hash functions may be split into two classes: unkeyed hash functions is, whose specification dictates a single input parameter (a message); and keyed hash functions, whose specification dictates two distinct inputs, a message

and a secret key. To facilitate discussion, a hash function is informally defined as follows.

Definition: A hash function (in the unrestricted sense) is a function h that has, as a minimum, the following two properties:

1. **Compression:** h maps input x of arbitrary finite bit length, to an output $h(x)$ of fixed bit length n .
2. **Ease of computation:** given h and input x , $h(x)$ is easy to compute.

The cryptographic restricted sense definition of hash function is given as follows: A collision free hash function h is an easily computable function that maps strings of arbitrary length to strings of some fixed length in such a way that finding two strings x and x' , with $h(x) = h(x')$ is computationally infeasible. The pair x, x' is called a collision for h . A collision free function f from m bits to t bits with $m > t$ is easily computable function that maps strings of length m to strings of length t , in such a way that finding a pair of strings x, x' of length m , with $f(x) = f(x')$ is computationally infeasible.

The presence of the term 'computationally infeasible' makes the previous definition somewhat fuzzy. By interpreting this term in different ways, different definitions of collision free functions or collision free hash functions can be obtained.

An iterated hash function is a hash function $Hash(.)$ determined by an easily computable function $h(.,.)$ from two binary sequences of respective lengths m and l to binary sequences of length m in the manner that the message $M = (M_1, M_2, \dots, M_n)$, where M_i is of length l , is hashed to the hash value $H = H_n$ of length m by computing recursively

$$H_i = h(H_{i-1}, M_i) \quad i = 1, 2, \dots, n \quad (2.2)$$

where H_0 is a specified initial value. Explicit dependence of on H_0 is shown by $H = Hash(H_0, M)$. The function h will be called the hash round function. Such a recursive construction of hash functions has been called the "meta-method" by Merkle (Menezes, 1997).

For iterated hash functions, following five attacks are defined:

1. Target attack: Given H_0 and M , find M' such that $M' \neq M$ but $\text{Hash}(H_0, M') = \text{Hash}(H_0, M)$.
2. Free-start target attack: Given H_0 and M , find H_0' and M' such that $(H_0', M') \neq (H_0, M)$ but $\text{Hash}(H_0', M') = \text{Hash}(H_0, M)$.
3. Collision attack: Given H_0 , find M and M' such that $M' \neq M$ but $\text{Hash}(H_0, M') = \text{Hash}(H_0, M)$.
4. Semi-free-start Collision attack: Find H_0 , M and M' such that $M' \neq M$ but $\text{Hash}(H_0, M') = \text{Hash}(H_0, M)$.
5. Free-start Collision attack: Find H_0 , H_0' , M and M' such that $(H_0', M') \neq (H_0, M)$ but $\text{Hash}(H_0', M') = \text{Hash}(H_0, M)$.

In applications where H_0 is specified and fixed, attack 2, 4 and 5 are not "real attacks". This is because the initial value H_0 is then integral part of the hash function so that a hash value computed from a different initial value will not be accepted. However, if the sender is free to choose and/or to change H_0 , attacks 2, 4 and 5 can be real attacks, depending on the manner in which the hash function is used.

For an m -bit hash function, brute-force target attacks, in which one randomly chooses an M' until one hits the "target" $H = \text{Hash}(H_0, M)$, require about 2^m computations of hash values. It follows from usual "birthday argument" that brute force collision attacks require about $2^{m/2}$ computation of hash values. In particular, for hash round functions with $l \geq m$ so that all 2^m hash values can be reached with one-block messages, brute force target attacks require about 2^m computations of the round function h while brute force collision attacks require about $2^{m/2}$ computations of the round function h . Computational security of hash function is ideal when there is no attack substantially better than brute force.

2.4. Walsh Transforms and Their Properties

Walsh functions and Walsh transforms have a wide range of applications to signal processing, image processing and communication as well as logic design and

analysis (Beauchamp, 1984), (Moraga, 1985). Xiao and Massey (1985), Rueppel (1986) and Ding et al (1991) have investigated the applications of spectral techniques to cryptology. Presentation of Walsh transform can be given as follows:

Let w and z are two vectors in $GF(2)^n$. For all w and z in $GF(2)^n$, the Walsh functions are defined as:

$$Q(w, z) = (-1)^{wz} \quad (2.3)$$

where $wz = w_1z_1 \oplus \dots \oplus w_nz_n$, and $\oplus$ denotes modulo-2 addition (exclusive-or operation). All Walsh functions are characters of the Abelian group $(GF(2)^n)$.

It has been shown that any Boolean function $f: GF(2)^n \rightarrow GF(2)$ has a finite series expansion as follows (Karpovsky, 1976):

$$f(z) = \sum_{w \in GF(2)^n} S_f(w) Q(w, z) \quad (2.4)$$

where $S_f(w)$ is given as

$$S_f(w) = 2^{-n} \sum_{z \in GF(2)^n} f(z) Q(w, z) \quad (2.5)$$

$S_f(w)$ is called the first kind of spectrum of f , and the above transform is referred to as the first kind of Walsh transform. The second kind of Walsh transform is defined as

$$S_{(f)}(w) = 2^{-n} \sum_{z \in GF(2)^n} (-1)^{f(z)} Q(w, z) \quad (2.6)$$

using $S_{(f)}(w)$, $f(z)$ may be computed as

$$f(z) = \frac{1}{2} - \frac{1}{2} \sum_{w \in GF(2)^n} S_{(f)}(w) Q(w, z) \quad (2.7)$$

It is easy to show that above formula hold. However, from the definition of the first kind of spectrum, it appears difficult to see what the Boolean function

spectrum represents. The following theorem gives the relationship between the two kinds of spectrum of Boolean functions. By the following theorem and the definition of the second kind of spectrum, it can be seen that the spectra represent the extent to which a Boolean function correlates with all linear functions.

Theorem: The above two kinds of spectrum are related by

$$S_{f(w)} = \begin{cases} -2S_f(w), & w \neq 0 \\ 1-2S_f(w), & w = 0 \end{cases} \quad (2.8)$$

Basic properties can be listed as follows: $w, z \in \text{GF}(2)^n$.

$$\sum_w S_{(f)}(w) = (-1)^{f(0)} \quad (2.9)$$

$$\sum_w S_f(w) = f(0) \quad (2.10)$$

$$\sum_w S_f(w)^2 = 2^n \sum_z f(z)^2 \quad (\text{Parseval Theorem}) \quad (2.11)$$

$$\sum_w S_{(f)}(w)^2 = 1 \quad (\text{Energy Conservation Law}) \quad (2.12)$$

2.4.1. The Best Affine Approximation of Boolean Functions

The best affine approximation (BAA) of Boolean functions has wide applications in logic design (Moraga, 1985). By employing the BAA approach, the realization complexity of circuits can be much reduced without penalty. Its application to cryptography was first noticed by Rueppel (1986). He used to BAA to

approximate the S-box S_2 in the Data Encryption Standard (DES), but did not exploit its application to cryptography further. Later BAA approach is widely used by Ding (1991) to analyze the stability of the stream cipher.

Definition: If the affine function $wx \oplus l$ makes the following formula achieve its minimal value,

$$\sum_x (f(x) \oplus wx \oplus l), \quad w, z \in GF(2)^n \quad l \in GF(2) \quad (2.13)$$

then $wx \oplus l$ is called the best affine approximation of $f(x)$.

Theorem: Let $P_f(wx)$ denote the probability of agreement between $f(x)$ and wx , then

$$P_f(wx) = \frac{1}{2} + \frac{1}{2} S_f(w), \quad w, x \in GF(2)^n \quad (2.14)$$

$$P_f(wx) = \begin{cases} \frac{1}{2} - 2S_f(w), & w \neq 0 \\ \frac{1}{2} + 2S_f(w), & w = 0 \end{cases} \quad (2.15)$$

Proof: By the definition of $S_f(w)$,

$$\begin{aligned} S_f(w) &= 2^{-n} [\# \{x: f(x) = wx\} - \# \{x: f(x) \neq wx\}] \\ &= -1 + 2 \# \{x: f(x) = wx\} / 2^n \\ &= 2P_f(wx) \end{aligned} \quad (2.16)$$

where $\#\{\bullet\}$ denotes the number of elements on the set $\{\bullet\}$.

The following theorem about how to determine the best affine approximation of Boolean functions was developed by Rueppel with an information approach.

Theorem: (Rueppel 1986) Assume that

$$a = \max \{ |S_{(f)}(w)| : w \in \text{GF}(2)^n \} \quad (2.17)$$

and $|S_{(f)}(w)| = a$. Then

1. If $S_{(f)}(w) \geq 0$, wx is a best affine approximation of $f(x)$ and the probability of agreement is

$$P_f(wx) = \frac{1}{2} + \frac{1}{2}a \quad (2.18)$$

2. If $S_{(f)}(w) < 0$, $1 \oplus wx$ is a best affine approximation of $f(x)$ and the probability of agreement is

$$P_f(1 \oplus wx) = \frac{1}{2} + \frac{1}{2}a \quad (2.19)$$

The best affine approximation of a Boolean function is not unique. What we would like to find in logic design and cryptology is the one $wx \oplus 1$ with minimum Hamming weight w ; the purpose of desiring such a linear function in logic design is for reducing the realization complexity of logic circuits. The spectra of $f(x)$ measure the correlation between $f(x)$ and all linear functions.

3. A SURVEY ON CELLULAR AUTOMATA AND ITS CRYPTOGRAPHIC APPLICATIONS

Conventional models of computation, such as Turing machine (Garey, Johnson, 1979), make a distinction between the structural part of computer, which is fixed, and the data on which the computer operates which are variable. The computer cannot operate on its own; it cannot extend or modify itself, or build other computer.

Cellular Automata were introduced by John von Neuman, following a suggestion of Ulam, to provide a more realistic model for the behavior of complex systems (Toffoli and Margolus, 1987). In a cellular automaton, objects that may be interpreted as passive data and objects that may be interpreted as computing devices are both assembled out of the same kind of the structural elements and subject to the same fine grained laws. Hence, computation and construction is just two possible modes of activity.

Physical systems containing many discrete elements with local interactions are often conveniently modeled as cellular automata. Any physical system satisfying differential equations may be approximated as a cellular automaton by introducing finite differences and discrete variables. A very wide variety of examples may be considered; only a few are presented here. In the most direct cases, the cellular automaton lattices are positioned in space. At a microscopic level, the sites may represent points in crystal lattice, with values given by some quantised observable (such as spin component) or corresponding to the types of atoms or units. At a more macroscopic model, each site in a cellular automaton may represent a region containing many molecules. In this way, cellular automata may be used as discrete models for nonlinear chemical systems involving a network of reactions coupled with a spatial diffusion (Greenberg, 1978). They have also been used in a model for the evolution of spiral galaxies (Schewe, 1981) and turbulent fluids.

Many biological systems have been modeled by cellular automata (Linden Mayer, 1968 and Rosen, 1981). Examples include leaf and branch arrangements, heart fibrillation, and population of nonmobile organisms (such as plants).

Cellular automata may be considered as parallel processing computers (Peterson, 1979). They have been used, for example, as highly parallel multipliers (Nishio, 1981) and prime number sieves (Fischer, 1965). They have been also used extensively for image processing and visual pattern recognition (Rosenfeld, 1979).

Cellular automata (CA) are massively parallel computing devices that have gained popularity as discrete dynamical systems that model natural systems consisting of large collections of simple objects locally interacting with each other. It consists of a d -dimensional (ideally infinite) array of identical cells. The positions of cells are indexed by Z^d , the set of d -tuples of integers. Each cell is always in one state from a finite state S ; that is, it carries a fixed amount of information. The cells can read states of some nearby cells, their so-called neighbors. The neighborhood vector

$$N = (x_1, x_2, \dots, x_n) \quad (3.1)$$

of the CA, consisting of n different elements of Z^d , tells where the neighbors of each cell are situated. The neighbors of a cell in position $x \in Z^d$ are the cells in the positions $x + x_i$, for $i = 1, 2, \dots, n$. when the addition of vectors is done compenentwise.

The cells alter their states synchronously on discrete time steps according to the local rule f of the CA. The global rule is a function from S^n to S that gives the new state of each cell as a function of the old states of its neighbors. The array is homogenous in the sense that all cells operate under the same local rule.

Above it is assumed that the array containing the cells is infinite. In practice, the space has to be made finite. This can be easily achieved by connecting the cells in a shape of a finite d -dimensional rectangle and wrapping it around, that is, by attaching the cells on the opposite edges of rectangle to each other, thus obtaining the d -dimensional torus. This is also named as periodic boundary conditions. More formally, let $M_1, M_2, \dots, M_d$ be positive integers and let the positions of all cells are indexed by the elements of the set.

$$X(M_1, M_2, \dots, M_d) = \{0, 1, \dots, M_1-1\} \times \{0, 1, \dots, M_2-1\} \times \dots \times \{0, 1, \dots, M_d-1\} \quad (3.2)$$

There are altogether $M_1 M_2 \dots M_d$ cells in the array. When determining the neighbors of cells the additions of the components of indices are done cyclically, modulo M_i in the i 'th component for each $i, 1 \leq i \leq d$, that is,

$$(x_1, x_2, \dots, x_d) + (y_1, y_2, \dots, y_d) = ((x_1 + y_1) \bmod M_1, (x_2 + y_2) \bmod M_2, \dots, (x_d + y_d) \bmod M_d). \quad (3.3)$$

The states of all the cells in the array (finite or infinite) are described by a configuration of the CA. A configuration c is a function $c : X \rightarrow S$, where $X = \mathbb{Z}^d$ (if there are infinitely many cells) or $X = X(M_1, M_2, \dots, M_d)$ (if the number of cells is finite). In either case, the configuration assigns states to all cells. Let $C(X, S)$ (or briefly C when X and S are clear from context) denote the set of all configurations. The local rule f determines the global function $Gf : C \rightarrow C$ that describes the dynamics of the CA. At each time step a configuration c is transformed into a new configuration $Gf(c)$ where

$$Gf(c)(x) = f(c(x + x_1), c(x + x_2), \dots, c(x + x_n)) \quad (3.4)$$

for all x in X .

Every configuration in a cellular automaton has a unique successor in time. A configuration may however have several distinct predecessors, as illustrated in the state transition diagram (figure 3.1). The presence of multiple predecessors implies that time evolution mapping is not invertible. The cellular automaton thus exhibits irreversible behavior in which information on initial states is lost through time evolution. The existence of multiple configurations implies that some configuration have no predecessors. These configurations occur only as initial states, and cannot be generated time evolution of the cellular automaton. These states are named as "Garden of Eden" configurations. Thus a cellular automaton is invertible if its global map is invertible, i.e., if every configuration - which, by definition, has exactly one successor - also has exactly one predecessor.

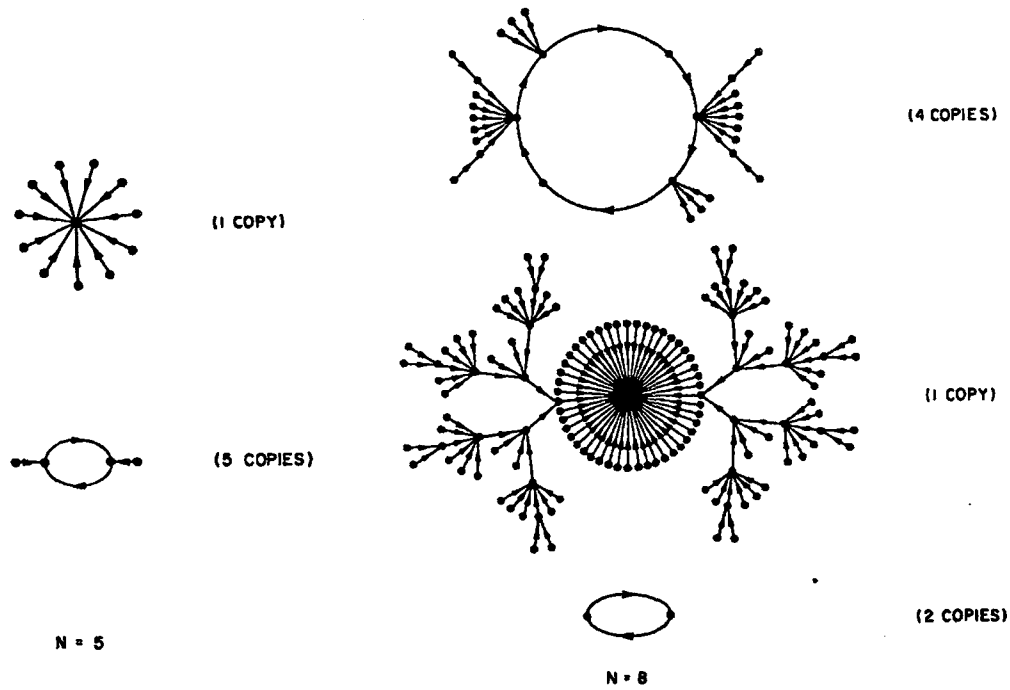


Figure 3.1: Global State transition diagram of rule 18 and periodic boundary cellular automata (Wolfram et. al, 1984).

Nonlinear cellular automata found wide application in the cryptology. They evolve according to

$$x_i' = f(x_{i-1}, x_i, x_{i+1}) \quad (3.5)$$

where $f(x_{i-1}, x_i, x_{i+1})$ is an arbitrary function over R_k , not reducible to linear form. The absence of additives in general prevents the use of algebraic techniques developed additive cellular automata (Wolfram, pp.76-101, 1994). The difficulties in the analysis of nonlinear cellular automata are analogous to those encountered in the analysis of nonlinear feedback shift registers. In fact, the possibility of universal computation with sufficiently complex nonlinear cellular automata demonstrates that a complete analysis of these systems is fundamentally impossible. Thus, in this case the analysis of nonlinear cellular automata is limited to extensive computer simulations and partial analysis of some radius 1 rules for 1-dimensional cellular automata.

3.1. Cryptographic Applications of Cellular Automata.

Dynamical systems are often described as "unpredictable" or "complex" as aspects of their behavior may bear cryptic relationships with the simple evolution laws that define them. Some theorists work to quantify this complexity in various ways. Others try to turn the cryptic nature of dynamical systems to a practical end: encryption of messages to preserve their secrecy.

Cellular automata based encryption algorithms have some advantages with respect to other encryption structures. Some of them can be given as follows: Cryptosystem constructed from cellular automaton primitives can be implemented in simply massively parallel hardware (Daemen, 1992a). They can be counted on to deliver high encryption decryption rates at low cost (Nandi et. al., 1994). In addition to these practical features, cellular automaton cryptosystem may help illuminate some foundational issues in both dynamical systems theory and cryptology, since each of these disciplines rests heavily on the meanings given to the intuitive notion of complexity (Gutowitz, 1993).

Unpredictability, and related notion, complexity is of major concern in the theory of dynamical system. The most interesting, unpredictability and complexity arise from simply built deterministic dynamical systems whose large time behavior bears little evident connection with the equations, which define them. One way to clarify the meaning of unpredictable in the context of dynamical systems is comparing it with similar notions in a neighboring scientific discipline. Here the neighboring discipline is cryptology.

The future state of a (chaotic) dynamical system depends sensitively on its initial state. After enough time has elapsed, initial condition is forgotten. Yet, since the system is deterministic, the same trajectory will always be traced out from the same initial condition. Thus, if the key of the cryptosystem is the initial state of a publicly known dynamical system, a collection of users who share a secret key can send a secret messages to each other by combining these messages in some way with some part of the trajectory traced out by the secret initial state under the action of the dynamical system. Anyone who does not know secret initial state would not be able to recreate the trajectory, and thus would not be able to disentangle it from the

encrypted message. By using this idea, some chaotic encryption systems have been proposed (Matthews, 1989). It has been shown that some of them have serious weaknesses (Wheeler, 1991), (Wheeler, Matthews, 1991).

The discrete dynamical system version of this idea (Wolfram, pp.429-432, 1985) uses iteration of a cellular automaton to generate the bit string. The cellular automaton chosen, known as rule 30, seems according to the numerical evidence (Wolfram, 1984) to generate temporal sequences that have a high degree of randomness. The secret key is the initial state of the system, and a message can be encrypted and decrypted by combining it with the temporal sequences generated by the dynamical system using an XOR operation. The cellular automata based on Wolfram's cryptosystem has been cryptanalysed in detail (Meier, Staffelbach, 1991). The essential observation that is using a known plaintext attack, the effective key space can be considerably reduced in size. The attack constructs internal state of cellular automaton by using partial linearity of rule 30. This system is still accepted as secure with the different parameters. Therefore, novel optical implementation is given of it by Madjorova et al (1997).

Linear feedback shift register has been used for cryptographic and noncryptographic (such as test pattern generator) applications. Serra et al (1988) have shown that linear cellular automata are isomorphic conventional linear feedback shift registers. The consequences of similarity transform between the transition matrix of a linear cellular automaton and that of linear feedback shift registers are explored. It is shown that the bit sequence from a stage of cellular automata is identical to that from the LFSR found by the similarity transformation. When two matrices have the same characteristic equation, they are said to be similar. In fact there exists a non-singular matrix P such that $B = P^{-1}A P$. This operation is called a similarity transform where A represents transition matrix of linear cellular automata and B represents characteristic matrix of the linear feedback shift register. If two autonomous linear finite state machines have the same characteristic polynomial, they have the same cycle structure (Bardell, 1990). Thus if the characteristic polynomial of a size n cellular automata is primitive polynomial, its cycle length is $2^n - 1$, the same as that of the linear feedback shift register obtained by a similarity transform of its transition matrix (Bardell, 1990).

Consideration of inverse as well as forward iteration of dynamical systems opens some new ways to use dynamical systems for encryption. One possibility, which again has occurred independently to a number of investigators, is to concentrate on reversible dynamical system. Using a reversible dynamical system, a message can be encrypted by encoding it as a state of system and then running the system in time some distance. The resulting state is ciphertext. To decrypt the ciphertext, the system is inverse iterated the same number of time steps as were used in encryption, recovering the plaintext as a state of system. When forward and inverse iteration is used, the key is the dynamical system itself.

Briefly, let us consider two systems that pursue this idea of using reversible dynamical systems, one by Guan (1987), and one by Kari (1992). Each of these authors aims toward the production of public key cryptosystems. A public cryptosystem uses two keys; one key is used for encryption, the other key for decryption. One key is held in private, the other rendered public. In the schemes of both Guan and Kari, the public key is a dynamical system inverse to the private key. Guan used a homogeneous variant of cellular automata in which the rule that updates a site's value depends on the site.

Nandi et. al. (1994) deal with the theory and application of cellular automata for a class of block ciphers and stream ciphers. Based on cellular automata state transitions, certain fundamental transformations are defined which are block-ciphering functions of the proposed enciphering scheme. These fundamental transformations are found to generate a simple (alternating) group of even permutations, which in turn is a subgroup of the permutation group. These functions are implemented by using class of programmable cellular automata built around rules 51, 153 and 195. Further, high quality pseudorandom pattern generators built around rule 90 and 150 programmable cellular automata with a rule selector (i.e., combining function) has been proposed as running key generators in stream ciphers. Prototype system is designed, fabricated and tested with 3-micron gate array library at Indian Telephone Industry. It was later shown that it is affine and insecure (Blackburn, et.al. 1997).

Daemen et al (1992 a) have proposed high-speed cryptographic coprocessor called Subterranean that is based on cellular automata. It can be used as a

pseudorandom sequence generator and cryptographic hashing. Estimated encryption and hashing speed are respectively 0.4Gbit/s and 0.6Gbit/s for 20MHz chip.

Cellular automata are also used for cryptographic hash function design (Daemen et al, 1991). At Crypto'89, Ivan Damgard (1990) presented a method that allows one to construct a computationally collision free hash function that has provably the same level of security as the computationally collision free function with input of constant length that is based upon. He also gave three examples of collision free functions to use in this construction. For two of these examples collisions have been found, and the third one that is designed by using cellular automata, is attacked in (Daemen, 1991) and is not collision-free. Later it was improved for concrete proposal based on cellular automata and it is named Cellhash.

3-way is a block cipher whose block and key length is a 96 bits (Daemen et al, 1992b). Key components in the construction of 3-way are a 3-bit nonlinear S-box and linear mapping that can be described by modular polynomial multiplication Z_2 . The arrangement of components allows software implementations in the range of 10Mbit/s on a modern PC. The dedicated hardware implementation allows above 1Gbit/s using standard technology (1.2 μ CMOS). The cipher structure of 3-way is shown strong to both linear and differential cryptanalysis. In this algorithm, S-box's is actually realized by using cellular automaton.

3.2. Known Cryptanalysis Methods of Cellular Automata

The only known cryptanalysis method of CA based systems is given by Meier and Staffelbach (1991) for stream cipher of Wolfram (1985). It will be given by details as follow:

3.2.1. Cryptanalysis of Wolfram Stream Cipher

The security of the cellular automata based stream cipher of Wolfram (1985) is investigated by Meier and Staffelbach (1991). A cryptanalytic algorithm is developed by them for a known plaintext attack where the plaintext is assumed to be known up to the unicity distance that is least amount of plaintext and corresponding ciphertext pairs for cryptanalysis. It was first defined by Shannon (Massey, 1995).

The cryptanalytic algorithm takes advantage of an equivalent description of the cryptosystem in which the keys are not equally probable. It is shown that key search can be reduced considerably. This is established by an information theoretic analysis of arbitrary key sources with non-uniform probability distribution.

The two rules those seem best as random sequence generators are nonlinear, and are given by

$$s_2(t+1) = f(s_1(t), s_2(t), s_3(t)) = s_1(t) \oplus (s_2(t) \vee s_3(t)) \quad (3.6)$$

$$s_2(t+1) = f(s_1(t), s_2(t), s_3(t)) = s_1(t) \oplus (s_2(t) \vee \overline{s_3(t)}) \quad (3.7)$$

It is indicated in (Wolfram, 1986) that, with respect to certain measures, rule (3.6) appears to be more favorable than rule (3.7). For stream cipher applications, the value $s_i(t)$ attained by a particular site i through time serve as pseudo random bits in a key stream sequence. The initial state of the register is used as the seed, or the secret key.

In Meier and Staffelbach (1991) cryptanalysis method, a known plaintext attack is assumed where the plaintext is known up to the unicity distance. It is shown that there is an equivalent cryptosystem, such that the problem of deducing the original key can be reduced to finding the key in the equivalent system. It turns out that the number of keys in the equivalent system is much smaller. Thus, a search for the equivalent key is much more efficient than a search for the original key.

Moreover, the equivalent system has another property, which is favorable for cryptanalysis; certain keys have much higher probability than others do. It turns out that in the situation the cost of a cryptanalytic attack is considerably reduced if one is contented to succeed only with a certain success probability δ (e.g., $\delta = 0.5$). In this respect, the efficiency of Meier and Staffelbach (1991) cryptanalytic algorithm has been determined by numerous experiments for different key sizes.

The results show that the security of the cellular automaton for rule (3.7) is in fact quite low for key sizes $N < 300$. Meier and Staffelbach (1991) mention that, for key size $N = 200$ and success probability $\delta = 0.5$, the average number of trials μ

necessary to find the key was obtained to be about 23,000. Thus the effective key size can be estimated as $\log_2 \mu = 14.5$ bits. For larger N , Meier and Staffelbach (1991) can extrapolate from the experiments that, for the cellular automaton (with rule (3.7)) to be securing, one needs the key size of about 1000 bits, as mentioned above.

If Meier and Staffelbach (1991) cryptanalytic algorithm is applied to cellular automata satisfying rule (3.7), it turns out that cryptanalysis is much easier than for rule (3.6). In fact, for the average number of trials μ , the effective key size $\log_2 \mu$ is roughly half of the corresponding value for rule (3.6), e.g., for $N = 400$ obtained value is $\log_2 \mu = 13.8$ bits. Therefore, this cryptanalysis method confirms to prefer rule (3.6) to rule (3.7).

The equivalent cryptosystem used in this attack implements a key source with a strongly nonuniform, but not explicitly known probability distribution. In general, for a given cryptosystem, knowledge of additional information (e.g., part of the plaintext) may result in an equivalent description of the cryptosystem where the keys in the equivalent system are no longer equiprobable. Therefore, an analysis of an "exhaustive search" for key sources with nonuniform probability distribution is of independent interest. Since by assumption one does not know the most probable keys, which would be tried first in an optimum search, the best one can do is to generate the trial keys according to the original probability distribution of the key source until the correct key has been found.

Suppose that the cryptanalyst performs m different attacks on m chosen keys, and the required numbers of trials are listed in ascending order $T_1 \leq T_2 \leq \dots \leq T_m$.

Then the mean value

$$\mu(m, \delta) = \frac{1}{\lceil \delta m \rceil} \sum_{i=1}^{\lceil \delta m \rceil} T_i \quad (3.8)$$

is an empirical value for the average number of trials to a given success probability δ . Authors also gave lower and upper bounds for the expected value of $\mu(m, \delta)$ which is derived for any key source. The relation of $\mu(m, \delta)$, with the entropy of the key source for stationary ergodic key sources with finite memory is shown as:

$$\lim_{N \rightarrow \infty} \liminf_{m \rightarrow \infty} \frac{\log_2 E[\mu(m, \delta)]}{N} = \lim_{N \rightarrow \infty} \limsup_{m \rightarrow \infty} \frac{\log_2 E[\mu(m, \delta)]}{N} = H_S \quad (3.9)$$

where H_S denotes the per bit entropy of the key source S (as introduced by Shannon (1949)), by (3.6) the values of the sites determined by a site vector $S(t) = (s_{i-n}(t), \dots, s_i(t), \dots, s_{i+n}(t))$ form a triangle as shown in Figure 3.2. The sequence $s_i(t)$ in the middle column is called the temporal sequence (at site i). The security of a stream cipher based on cellular automata relies on the difficulty of determining the seed from the given temporal sequence. In an exhaustive search, one would try all possible seeds until the correct temporal sequence is produced.

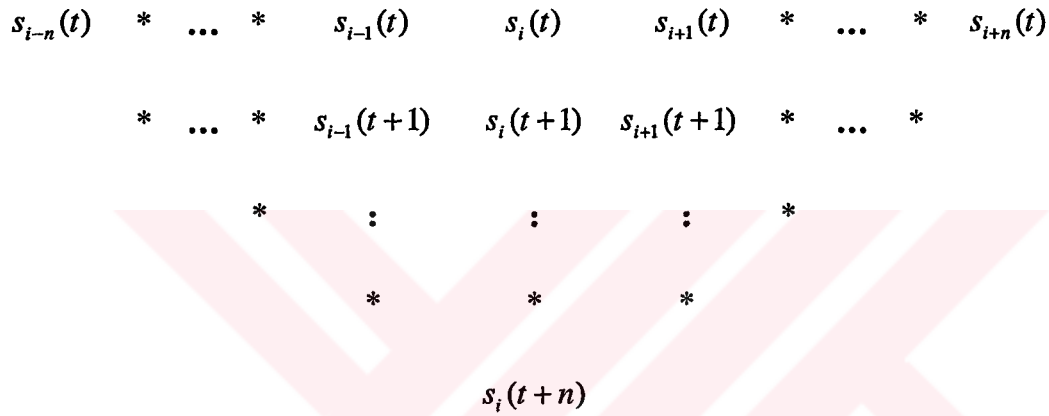


Figure 3.2: Triangle determined by $s_{i-n}(t), \dots, s_{i+n}(t)$.

A different approach is introduced, which finds the seed using the partial linearity of (3.6) by writing it in the form $s_{i-1}(t) = s_i(t+1) \oplus (s_i(t) \vee s_{i+1}(t))$. Given the values of the cells in two adjacent columns, this allows the values of all cells in a triangle to the left of the temporal sequence to be reconstructed. This process is called completion backwards. By completion backward, $N-1$ digits of the temporal sequence $s_i(t)$ and $N-2$ digits of its right adjacent sequence $s_{i+1}(t)$ determine the seed. Similarly, the seed may also be reconstructed from $N-2$ digits of the temporal sequence and $N-1$ digits of its left adjacent sequence $s_{i-1}(t)$.

Thus if $N-1$ digits of the temporal sequence are given, knowledge of the seed is equivalent to knowledge of one of its adjacent sequences. In view of this equivalence, an adjacent sequence can be considered as a key that determines the remainder of the temporal sequence. In a known plaintext attack, only (a portion of)

the temporal sequence is known, but neither of its adjacent sequences. The aim is to search for a correct (right or left) adjacent sequence and to determine the seed by completion backwards. In fact, it turns out that this search can be done much faster than a direct exhaustive search over the original seed.

In constructing adjacent sequences, which are consistent with a given temporal sequence, there is an essential difference between left and right adjacent sequences. Meier and Staffelbach (1991) explain this difference for a general cellular automaton of width $2n+1$ where the sites $s_{i-n}(t)$, ..., $s_{i+n}(t)$ are not restricted by periodic boundary conditions.

Suppose a temporal sequence $s_i(t)$, ..., $s_i(t+n)$ are given. Referring to figure 3.2, the problem of finding the left adjacent sequence is equivalent to completing the triangle consistently to the left of the temporal sequence. By (3.6), knowledge of $s_{i-n}(t)$, ..., $s_{i-1}(t)$ together with the temporal sequence is sufficient to determine the triangle to the left. However, the sites $s_{i-n}(t)$, ..., $s_{i-1}(t)$ cannot be chosen arbitrarily. For example, if $s_i(t)=1$, then $s_{i-1}(t)=s_i(t+1) \oplus 1$, and there are similar restrictions at any site in the process of extending to the left.

On the other hand, for triangles to the right any choice of the sites $s_{i+1}(t)$, ..., $s_{i+n}(t)$ has a completion consistent with the given temporal sequence. This follows from the fact that, by (3.6), for any right adjacent $s_{i+1}(t)$ there is a left adjacent $s_{i-1}(t)$ consistent with the next digit $s_i(t+1)$ of the temporal sequence. Furthermore, according to (3.6), any choice of $s_{i+2}(t)$, ..., $s_{i+n}(t)$ has a consistent extension to $s_{i+1}(t+1)$, ..., $s_{i+n-1}(t+1)$. Iteration of this construction leads to the desired completion of the triangle to the right. This process will be called completion forwards which, constructs a right adjacent sequence consistent with the given temporal sequence for any choice of the sites $s_{i+1}(t)$, ..., $s_{i+n}(t)$.

Now suppose that the cellular automaton satisfies periodic boundary conditions with period N . This means that the indices are calculated modulo N . For illustration consider Figure 3.2 where $n=N-1$ and $s_{i-N} s_{i-1}(t)=s_{i+N-1}(t)$, ..., $s_{i-N+1}(t)=s_{i+1}(t)$. Thus to determine the seed, $(s_i(t), \dots, s_{i+N-1}(t))$ or $(s_{i-N+1}(t), \dots, s_i(t))$ are computed. For $t' \geq t$, the right adjacent $s_{i+1}(t')$ may not be chosen arbitrarily to be

consistent with the next digit $s_i(t' + 1)$ of the temporal sequence, as the left adjacent $s_{i-1}(t')$ may be restricted by the boundary conditions. Nevertheless, by neglecting the boundary conditions, for any choice of the value $s_{i+1}(t), \dots, s_{i+N-1}(t)$, completion forwards yields a consistent triangle to the right, and one obtains a possible candidate for the right adjacent sequence. In fact, if $s_{i+1}(t), \dots, s_{i+N-1}(t)$ coincide with the original seed, the correct right adjacent sequence will be produced. It turns out that there are many (incorrect) seeds leading to the correct right adjacent sequence that, by completion backwards, determines the correct seed. This phenomenon is the basis of Meier and Staffelbach (1991) cryptanalytic algorithm which, for given digits $s_i(t), \dots, s_{i+N-1}(t)$ of the temporal sequence, determines the seed. The algorithm is given as follows:

1. Generate a random seed $s_{i+1}(t), \dots, s_{i+N-1}(t)$.
2. Complete forwards, i.e., determine the right adjacent sequence by executing

FOR $k = 1$ TO $N-2$

FOR $j = 1$ TO $N-k-1$

$$s_{i+j}(t+k) = s_{i+j-1}(t+k-1) \oplus (s_{i+j}(t+k-1) \vee s_{i+j+1}(t+k-1))$$

3. Complete backwards, i.e., determine the seed $s_{i-N+1}(t), \dots, s_{i-1}(t)$ by executing

FOR $j = 1$ TO $N-1$

FOR $k = N-1-j$ DO TO 0

$$s_{i-j}(t+k) = s_{i-j+1}(t+k+1) \oplus (s_{i-j+1}(t+k) \vee s_{i-j+2}(t+k))$$

4. Load the cellular automaton with the computed seed and produce the output sequence up to the length according to the unicity distance. Terminate if the produced sequence coincides with the given temporal sequence else go to step 1.

Although this analysis effort, Wolfram generator is still accepted as secure with the different parameters, hence novel optical implementation is given by Madjorova et al (1997).

3.2.2. Cryptanalysis of Damgard's One-Way Function Based on Cellular Automata

At Crypto'89 Ivan Damgard (1990) presented a method that allows constructing a computationally collision free hash function that has provably the same level of security as the computationally collision free function with the input of constant length that it is based upon. A collision free hash function will be denoted by "hash function" and a collision free function with an output shorter than its input by "CF function". A function that takes an input of fixed length to an output of fixed length that is shorter than the input will be denoted by "FI function". A CF function is a special case of a FI function.

For the hash function, based on cellular automata Daemen et al. (1991) show that, the probability of the hash value being unaffected by the complementation of certain message bits, is large. Hence, collisions for the hash function can be created ad libitum. However, this does not necessarily mean that collisions have been found for the corresponding hash functions that is built on these CF functions. The underlying FI functions are shown computationally collision free, the proof of Damgard does not apply and Damgard's hash functions are not provable collision free.

Damgard's CF function $g_c(.)$ computes a 128-bit string h starting from a 256-bit string x and $h = g_c(x)$. In the following, Daemen et al. (1991) method will be explained in an informal way that how it operates and how to create collisions:

A constant 256-bit string z is appended to x resulting in a 512-bit string. This is taken to be the initial state $a^0 = a_0^0 a_1^0 \dots a_{511}^0$ of a 512-bit binary cellular automaton with periodic boundary conditions (Toffoli, Margolus, 1990). When the cellular automaton is iterated the bits of the state are modified according to the local updating rule

$$a_i^{t+1} = a_{i-1}^{t+1} \oplus (\overline{a_i^t} \vee \overline{a_{i+1}^t}) \quad (3.10)$$

where a_i^t denotes the i -th bit of the state of the CA after t iterations. The cellular automaton performs 384 iterations and the output h consists of the concatenation of the first bits of the strings a^{257} to a^{384} , hence $h = h_0 h_1 \cdots h_{127}$ with $h_i = a_0^{257+i}$. h can be seen as the concatenation of subsequent values in time of one bit of the state, i.e. a temporal sequence. The state of the cellular automata at a certain moment can be seen as an array of values in space, i.e. a spatial sequence.

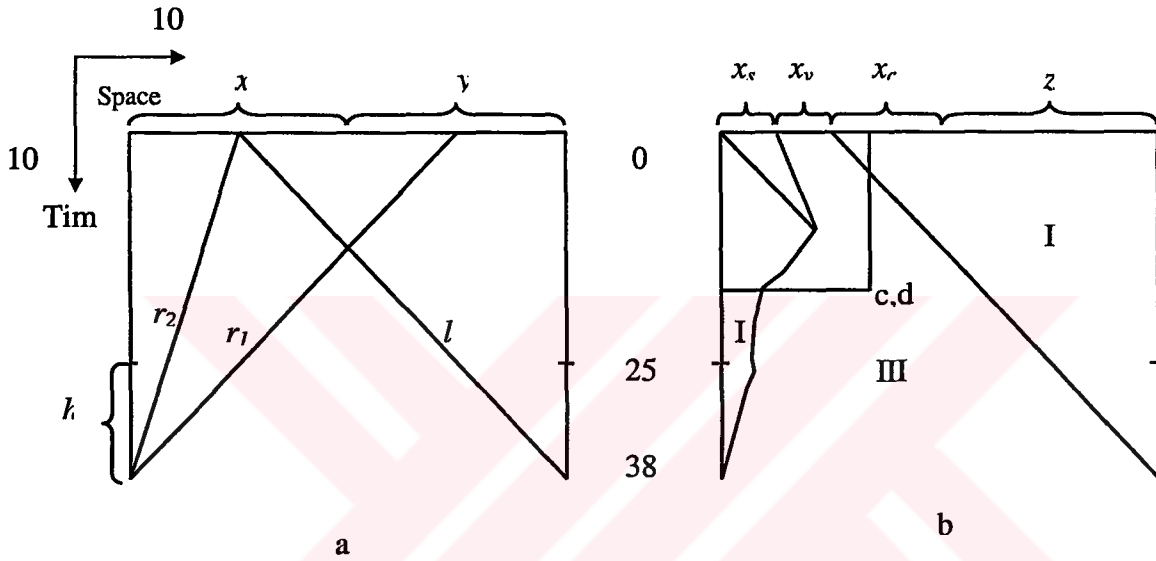


Figure 3.3. The space-time diagram of the computation (Wolfram, 1986) .

Bit a_i^t is at the position i units to the right and t units downwards from the upper left corner of the diagram. The top line of the diagram is the initial state a^0 containing x and z . Because there are periodic boundary conditions, the right and left edges are in fact adjacent and the diagram can be seen as an unfolded cylinder. Mathematically this implies that the lower indices should be taken modulo 512.

Because of the local nature of the cellular automata, updating each bit a_i^t can only affect bits a_{i-1}^{t+1} to a_{i+1}^{t+1} of the next state. Hence, each bit a_i^t can only depend on bits a_{i-1}^{t-1} to a_{i+1}^{t-1} of the previous state. Applying this recursively implies that a_i^t depends only on a_{i-k}^{t-k} to a_{i+k}^{t-k} of the state k iterations ago. The union of these sets for

all k is called the dependence area of a'_i denoted by A'_i . In words: if all bits in A'_i are left unchanged, a'_i cannot change, whatever happens outside of A'_i . In figure 3.3 the borders of the dependence area of the last bit of h are indicated. The left border is line l with slope $+1$ and the right border is r_1 with slope -1 . This comprises the dependence areas of all other bits of h that are mere translations upwards of the indicated area. Hence, the indicated area can be seen as the dependence area of all h and will be denoted by H .

However, the updating rule is asymmetric, and so is the propagation of information (Wolfram, 1986). It turns out that for almost all-initial states a^0 , H is bounded at the right by a border that is much steeper than r_1 . In Figure 3.3 the actual border is approximated by a diagonal r_2 with slope -3 such that all of a_0 lies in H . In reality, it is fixed by the initial state a_0 . For most a_0 however, r_2 can be well approximated by a straight line with slope -4 . This implies that for these states some bits of a_0 lie outside of H , providing a first type of collisions. For most inputs the output is independent of the bits a_{126-k}^k to a_{126}^0 for some i . Hence for the majority of inputs x there exists an integer $k > 0$ such that all x' that only differ from x in bit positions $126-k$ to 126 have the same output h , leading to 2^k simultaneous collisions.

The description in (Damgard, 1990) does not specify the constant z . However Daemen's attack works independently of the particular value of z . x is the concatenation of x_s (64 bits), x_v (62 bits) and x_c (130 bits). x_s contains the alternating pattern. Hence, in the division of figure 3.3 area II contains the alternating pattern that "buffers" changes in area III from area I. The dependence area for this configuration is formed by area I. In conclusion: all x that have the same x_c and for which x_s contains the alternating pattern have the same output. This yields 2^{130} different sets; each containing 2^{62} inputs with the same output h . Finding these sets requires no computations whatsoever.

The input to the hash function is a message m that has been padded in a one-to-one way to a multiple of 128 bits. Suppose $m = m_0 || m_1 || \dots || m_n$, with every m_i a 128 bit string. The hash, value h is the result of applying the hash

function $h_c(\cdot)$ to m . We have $h = h_c(m)$. This function is defined in terms of the CF function g_c .

$$h^i = g_c(h^{i-1} \parallel m_{i-1}) \quad \text{for } i = 1, \dots, n \quad (3.11)$$

h^0 is equal to initial value, a fixed initial value that is part of the specification of $h_c(\cdot)$. The hash value h is given by h^n . Suppose we have an intermediate hash value h^i . h^{i+1} is then computed by $h^{i+1} = g_c(h^i, m_i)$. In figure 3.3 it can be seen that changes in the first k bits of m_i (denoted by $\leftrightarrow$) influence only the $k + 1$ last bits of h^{i+1} (denoted by $\leftrightarrow$) if the border β is steep enough. This is the case for almost all-initial configurations. Now suppose the last bit of h^{i+1} is not affected. This happens with probability $1/2$. In the next step, h^{i+2} is computed from h^{i+1} and m_{i+1} . In figure 3.3 it can be seen that, for small k there is a large probability that h^{i+2} will be independent of the k bits of h^{i+1} that were affected. From this, it can be concluded that changes in bits that are situated at the beginning of the block m_i have a fair probability not to influence the hash value h .

4. THE ZIGZAG ALGORITHM

Cellular automata (CA) are massively parallel computing devices that have gained popularity as discrete dynamical systems that model natural systems consisting of large collections of simple objects locally interacting with each other. It has wide applications on the physics, complexity theory, cryptography, etc. The zigzag algorithm is directly related to cryptanalysis of CA based encryption algorithm. It can be generalized to multidimensional CA.

Although the zigzag algorithm can be generalized to different family of cellular automata, the zigzag algorithm is given for one-dimensional and periodic boundary cellular automata in this thesis because of wide cryptographic applications of this class. Consequentially necessary descriptions of considered system will be given as follows: A one-dimensional cellular automaton consists of a linearly connected array of n cells, each of which takes the value of 0 or 1, and a Boolean function $f(x)$ with q variables. The value of the cell x_i is updated in parallel (synchronously) using this function in discrete time steps as $x_i' = f(x)$ for $i = 1, 2, \dots, n$. The boundary conditions are usually handled by taking the index values modulo n , i.e., the linearly connected array is actually a circular register. The parameter q is usually an odd integer, i.e., $q = 2r + 1$, where r is often named the radius of the function $f(x)$; the new value of the i th cell is calculated using the value of the i th cell itself and the values of r neighboring cells to the right and left of the i th cell. The cellular automaton for $q=3$ is illustrated in Figure 4.1.

Since there are n cells, each of which takes the values of 0 or 1, there are 2^n possible state vectors. Let S_k denote the state vector at the time step k . Starting from an initial state vector S_0 , the cellular automaton moves to the states S_1, S_2, S_3 , etc., at time steps $k = 1, 2, 3$, etc. The state vector S_k takes values from the set of n -bit binary vectors as k advances, and the state machine will eventually cycle, i.e., it will reach a state S_{k+P} which was visited earlier $S_k = S_{k+P}$. The period P is a function of the initial state, the updating function, and the number of cells.

Cellular automata are generally considered as discrete dynamical systems, or discrete approximations to partial differential equations modelling a variety of natural systems. Further information concerning the properties, dynamics, and applications of cellular automata can be found in (Wolfram, 1994).

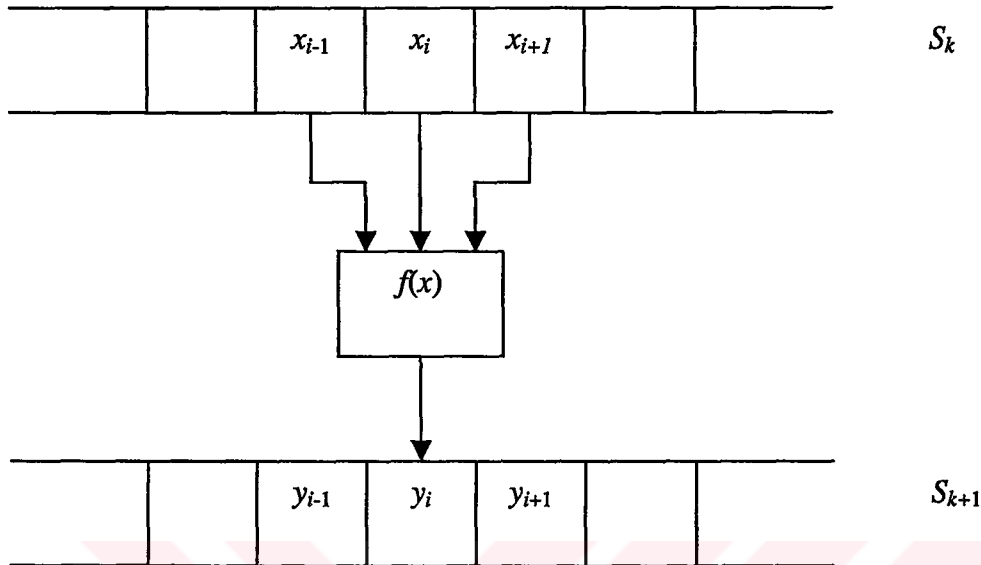


Figure 4.1: One-dimensional cellular automata with the radius one updating function.

The construction of seed value (the initial state vector S_0) from the given sequence of state vector is difficult. It was stated in (Wolfram, 1985) that

This problem is in the class NP. No systematic algorithm for its solution is currently known that takes a time less than exponential in n .

While it is true that an exponential number of trials is required to compute S_0 given one or more S_k for $k \geq 1$ for general updating functions and arbitrary values of S_0 , it is shown that the number of trials may be much fewer for particular classes of updating functions and seed values. An algorithm is given to compute S_k for a given S_{k+1} and $f(x)$, whose running time depends on the updating function $f(x)$ and the state S_{k+1} . If $f(x)$ is an affine function, then the running time of the algorithm is only $O(nq^2)$. Furthermore, the running time of the zigzag algorithm may still be $O(nq^2)$ for nonlinear updating functions and certain seed values. The number of trials is exponential in n , but fewer than 2^n for nonlinear updating functions and arbitrary seed values. The inversion of the cellular automata for general updating functions

and seed values is an NP-complete problem since it is equivalent to the general problem of solving Boolean equations.

4.1. Brief Description of The Zigzag Algorithm

The proposed zigzag algorithm is based on the best affine approximation of the q -variable updating function $f(x)$. Using tools from the spectral analysis of Boolean functions (Beauchamp, 1975), (Ding and et al, 1991), (Rueppel, 1986) a q -variable linear function $g(x)$ which is the best affine approximation to $f(x)$ is constructed. In other words, $g(x)$ has the minimum Hamming distance to $f(x)$ among all linear functions. The Hamming distance between two functions is defined as the Hamming distance between the binary vectors of length 2^q produced by the application of all possible input values $x \in \mathbb{Z}_2^q$ to these functions.

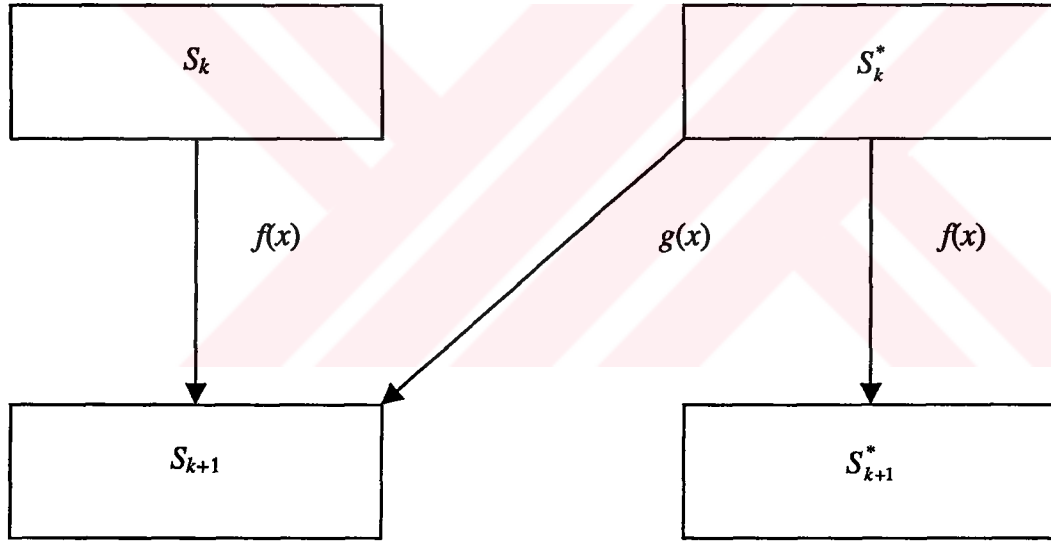


Figure 4.2. Schematic representation of the first four steps of zigzag algorithm.

The main idea of the zigzag algorithm is illustrated in Figure 4.2. The state vector S_{k+1} is computed by applying the function $f(x)$ to the state vector S_k . An approximation to S_k is calculated by finding a vector S_k^* that maps to S_{k+1} under the linear function $g(x)$. The vector S_k^* is computed by solving a set of linear equations whose matrix is determined by the parameters of the linear function $g(x)$. Then $f(x)$ is applied to S_k^* and calculated S_{k+1}^* which is an approximation to the state vector S_{k+1} .

Since S_{k+1} is known, S_{k+1} and S_{k+1}^* are compared, and corrections to the state vector S_k^* is made until the state vector S_k^* which results in the equality $S_{k+1}^* = S_{k+1}$ is obtained. The zigzag algorithm calculates one of the predecessors of the state vector S_{k+1} . Repeated application of the zigzag with different $g(x)$, whenever applicable, may yield other predecessors.

It is assumed that the n -dimensional binary vector S_{k+1} and the q -variable updating function $f(x)$ are given. Objective is to devise an algorithm for computing S_k as shown in Figure 4.2. The details of the zigzag algorithm are described below.

4.1.1 The Definition of Zigzag Algorithm with the Algorithmic Steps

The zigzag algorithm has 12 algorithmic steps. These steps will be given as follows with all details.

1. The best affine approximation $g(x)$ to the updating function $f(x)$ is obtained, and α , probability of agreement between $f(x)$ and $g(x)$, is determined.

The q -variable function $f(x)$ may be given in algebraic form, however, only its values at 2^q points in order to construct its best affine approximation are necessary. The best affine approximation of $f(x)$ is the function:

$$g(x) = w_1 x_1 \oplus w_2 x_2 \oplus \dots \oplus w_q x_q \oplus c \quad (4.1)$$

where $w_i \in Z_2$ for $i = 1, 2, \dots, q$, and $c \in Z_2$, such that the summation:

$$\sum_{x \in Z_2^q} f(x) \oplus g(x) \quad (4.2)$$

achieves its minimal value (Ding and et al, 1991). The function $g(x)$ is determined by calculating the vector $w = (w_1, w_2, \dots, w_q)$ and the constant c , which can be computed by exhaustive search. However, there is an efficient method for constructing $g(x)$ based on the Walsh transform (Beauchamp, 1975), (Ding and et al, 1991), (Rueppel, 1986), (Rueppel, 1992). The function $f(x)$ and $g(x)$ will agree at a portion of 2^q points. Let A be the number of points at which $f(x) = g(x)$. Note that $A \geq 2^q/2$, since,

otherwise, $f(x)$ will agree with $g(x) \oplus 1$ (the complement of $g(x)$) in more than $2^q/2$ points, and thus, $g(x) \oplus 1$ is taken as the best affine approximation. The probability of agreement between $f(x)$ and $g(x)$ is defined as $\alpha = A/2^q$. In general, $\alpha \geq 1/2$. If $f(x)$ is linear, then $\alpha = 1$.

2. The state vector S_k^* is obtained using the affine function $g(x)$, and by solving a set of linear equations of dimension n in $GF(2)$. Let $S_k^* = (x_1, x_2, \dots, x_n)$ and $S_{k+1} = (y_1, y_2, \dots, y_n)$. The state vector S_{k+1} is obtained by applying the function $g(x)$ to the state vector S_k^* . Instead of computing the inverse of the function $g(x)$, a set of linear equations of dimension n in the field $GF(2)$ involving the known quantities y_i is solved, in order to compute the unknown quantities x_i for $i=1, 2, \dots, n$. The matrix of the linear system of equations is a nearly band matrix with a bandwidth of q . For example, the linear system of equations of dimension $n = 8$ for the 3-variable ($q=3$) approximating affine function:

$$g(x) = w_1 x_1 \oplus w_2 x_2 \oplus w_3 x_3 \oplus c \quad (4.3)$$

is given as

$$\begin{bmatrix} w_2 & w_3 & 0 & 0 & 0 & 0 & 0 & w_1 \\ w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 \\ w_3 & 0 & 0 & 0 & 0 & 0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} + \begin{bmatrix} c \\ c \\ c \\ c \\ c \\ c \\ c \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} \quad (4.4)$$

If calculated band matrix has no unique solution, then $g(x) = x_1$ is chosen. This function has a unique solution for all size of cellular automata.

Lemma: $g(x) = x_1$ defines invertible band matrix for all size n cellular automata.

Proof: A matrix of the following form

$$\begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_{n-1} & a_0 & a_1 & \cdots & a_{n-2} \\ a_{n-2} & a_{n-1} & a_0 & \cdots & a_{n-3} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{bmatrix} \quad (4.5)$$

is called a circulant matrix, denoted by $c[a_0, a_1, \dots, a_{n-1}]$. Let $S(n, q)$ denote the set of all $n \times n$ circulants over F_q . Denote the $n \times n$ permutation matrix $S = (S_{ij})$ where

$$S_{ij} = \begin{cases} 1, & \text{if } j \equiv i+1 \pmod{n}, \quad 0 \leq i, j \leq n-1 \\ 0, & \text{otherwise} \end{cases} \quad (4.6)$$

Then $S^n = I_n$ and if $A = c[a_0, a_1, \dots, a_{n-1}]$ then

$$A = \sum_{i=0}^{n-1} a_i S^i \quad (4.7)$$

It is clear that $S(n, q)$ is isomorphic to R_n , the ring of polynomials in $F_q[x]$ modulo $x^n - 1$, under the mapping defined by

$$\rho \left(\sum_{i=0}^{n-1} a_i S^i \right) = \sum_{i=0}^{n-1} a_i x^i \quad (4.8)$$

It is readily established that the nonsingular matrices of $S(n, q)$ are in one-to-one correspondence with the polynomial coprime to $x^n - 1$, i.e., and the invertible elements in R_n . Thus if $A \in S(n, q)$ and $\rho(A) b(x) \equiv 1 \pmod{x^n - 1}$ i.e. $\gcd(\rho_A(x), x^n - 1) = 1$ then $A^{-1} = \rho^{-1}(b(x))$ (Blake, 1993). Inversion matrix for function $g(x) = x_1$ can be given as follows:

$$A = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \quad (4.9)$$

by using isomorphism

$$\rho_A(x) = x^{n-1} \quad (4.10)$$

it can easily be seen that $\gcd(x^{n-1}, x^n + 1) = 1$, thus above matrix invertible for all size n cellular automata

3. The function $f(x)$ is applied to the state vector S_k^* to obtain the state vector S_{k+1}^* , as shown in Figure 4.2.

4. If $S_{k+1} = S_{k+1}^*$, then algorithm stops. The algorithm found a predecessor to the state vector S_{k+1} . If $S_{k+1} \neq S_{k+1}^*$, the first three steps are again executed for different BAAs which are found at step 1. If still $S_{k+1} \neq S_{k+1}^*$, then 'corrections' are applied to the state vector S_k^* which gives the equality $S_{k+1} = S_{k+1}^*$. The correction heuristics are in Steps 5 through 12.

5. S_{k+1} and S_{k+1}^* are compared, and all single bits which are not equal marked.

6. Let x_j and x_j^* be two bit values in S_{k+1} and S_{k+1}^* , respectively. Either $x_j^* = x_j$ or $x_j^* = \overline{x_j}$ is true. If the latter is true, the correct bit value can be obtained by complementing, which requires that the inputs to the function $f(x)$ giving the value of x_j^* need to be changed. The truth table of the function $f(x)$ is examined, and a template consisting of all possible input values that produce the complemented output is prepared. There are two kinds of templates for every incorrect bit: bit value is changed from 0 either to 1, or from 1 to 0.

7. Templates that are equal to the number of incorrect bits in the state vector S_{k+1}^* , are constructed. In other words, the number of templates is equal to the Hamming distance between S_{k+1} and S_{k+1}^* , i.e., $H(S_{k+1}, S_{k+1}^*)$. The length of a template is equal to the number of 1s in the function $f(x)$ if the change is to be made from 0 to 1. Similarly, if the change is to be made from 1 to 0, the length of the template will be equal to the number of 0s in $f(x)$.

8. The templates are grouped in such a way that each group contains neighbouring templates. Templates are considered neighbours if the difference of their indices is strictly less than q . For example, if $q=3$ and the indices of incorrect bits are 6, 7, 8, 11, 13, then the templates corresponding to 6, 7, 8 are placed in one group, while the templates corresponding to 11, 13 are placed in another group.

9. Each group is then represented by a single reduced template, which enumerates all possible input values producing the correct bits within the group. This operation is performed by examining the templates, and removing the conflicts. The objective is to find input configurations, which correct all bits in the group at the same time. An input configuration, which corrects one bit while destroying one or more bits is considered a conflict, and is removed from the set. This removal reduces the number of rows in each template.

10. Any rows within the template are removed, destroying the bit values outside the group. The input values correcting the bit values within a group should not be allowed to corrupt any correct bits, which have not been placed in a group.

11. The remaining templates give all possible bit configurations giving the correct state vector S_k^* . These bit configurations are placed in S_k^* one by one, apply the function $f(x)$. If there is a bit configuration which produces $S_{k+1}^* = S_{k+1}$, then a predecessor to the state S_{k+1} has been found.

12. If none of these bit configurations produces the correct state vector, then it can be concluded that bits not placed in the templates may need to be changed. All bit values outside the templates together with the bit values in the reduced templates are enumerated exhaustively in order to calculate the correct state vector S_k^* .

4.2. Analysis of The Zigzag Algorithm

Execution time is the basic criteria for success of the zigzag algorithm. The time complexity of zigzag algorithm must be smaller than exhaustive search for success. The order of exhaustive search can be given as $O(2^n)$ for size n CA.

Since each updating function defines a new system, analytic derivation cannot be given and only a loose upper bound can be given up to step 11 of zigzag algorithm. Consequently, whole algorithm (up to step 12) execution time can be derived empirically by the help of computer simulation and curve fitting techniques. Analytical upper bound is derived as $O((1+\alpha)^n)$ for up to step 11 of zigzag algorithm, however empirical average complexity is found as $O(2^{0.656n-8.9658})$.

The derivations of analytical and empirical results are given in the following section.

4.2.1 Analytic Results on Time Complexity Function of Zigzag Algorithm.

The construction of the best affine approximation of the updating function $f(x)$ is performed using the Walsh transform. The running time of the Walsh transform is $O(q2^q)$ (Beauchamp, 1984) if a fast (FFT-type) algorithm is used, where q is the number of variables in $f(x)$. q is small and fix number (3 or 5) and generally $q \ll n$ for implementation. Since this is very small compared to the number of trials required, the time required to calculate all $g(x)$ functions which are the best affine approximations to $f(x)$ is ignored.

In Step 2, one of these affine functions is selected, and a linear system of equations of dimension n in the field F_2 is solved. This requires implementation of F_2 arithmetic and the Gaussian elimination algorithm. As was mentioned, the matrix of these linear equations is band matrix with a bandwidth of q . It is well known that the Gaussian elimination algorithm requires $O(nq^2)$ arithmetic operations to solve a linear system of equations with size n and bandwidth q . If $f(x)$ happens to be a linear function, the zigzag algorithm will terminate in Step 4 since in this case $f(x)$ will be equal to $g(x)$ at all points, and the application of $f(x)$ to the state vector S_k^* will produce S_{k+1}^* which will be equal to S_{k+1} .

Even if $f(x)$ is not linear, the algorithm may still terminate in Step 4 for certain seed values. This is because the nonlinear function $f(x)$ and the linear function $g(x)$ agree at $\alpha 2^q$ points; these two functions are indistinguishable from one another if these q -tuples are used as inputs. If S_k consists of those q -tuples at which

$f(x) = g(x)$, then the application of $f(x)$ to S_k , and $g(x)$ to S_k^* will result in the equality $S_{k+1} = S_{k+1}^*$. Thus, the solution will be obtained immediately after solving the set of linear equations.

If S_{k+1} is different from S_{k+1}^* , the correction steps (5 through 11) are applied. The basic template is constructed for each bad cell, i.e. a cell which differs in S_{k+1} and S_{k+1}^* . All of the intersecting basic templates are merged together, executing Steps 6 through 8. Finally, all combinations from the remaining reduced templates tried to find the correct value of S_k . The size of the final search space is the product of the number of rows the remaining reduced templates.

In the rest of this analysis, our goal is to find an upper bound for the size of the final search space. As for the notation, $P(.)$ will be used for probability, $E[.]$ for expected value, $P(.|.)$ for conditional probability, $E[.|.]$ for conditional expected value. The number of rows in the template is considered as size of template.

This part of analysis covers computing the expected value of the size of a reduced template that is obtained by merging intersecting templates. Let T be the reduced template obtained by merging the intersecting templates T' and T'' , respective sizes r , r' , and r'' . Define δ_{ij} as

$$\delta_{ij} \triangleq \begin{cases} 1 & \text{If } i\text{th row of } T' \text{ and } j\text{th row of } T'' \text{ agree at the intersecting cells.} \\ 0 & \text{otherwise.} \end{cases} \quad (4.11)$$

If the size of the intersection between T' and T'' (i.e., number of cells common to both) is denoted by v , the probability of δ_{ij} being 1 is 2^{-v} , all $i = 1, 2, \dots, r', j = 1, 2, \dots, r''$. Then the expected size of T , i.e. $E[r]$, is

$$E[\delta_{ij}] = 1 \cdot P(\delta_{ij} = 1) + 0 \cdot P(\delta_{ij} = 0) \quad (4.12)$$

$$E[r] = E\left[\sum_{i,j} \delta_{ij}\right] = \sum_{i,j} E[\delta_{ij}] = \sum_{i,j} 2^{-v} = r' r'' 2^{-v}. \quad (4.13)$$

By induction, the expected size of a reduced template which is obtained by merging m intersecting templates $T^{(1)}, T^{(2)}, \dots, T^{(m)}$ with respective sizes $r^{(1)}, r^{(2)}, \dots, r^{(m)}$ is equal to $\prod_{i=1}^m r^{(i)} \prod_{i=1}^{m-1} 2^{-v_i}$ where v_i denotes sizes of the intersection between $T^{(i)}$ and $T^{(i+1)}$. $T^{(1)}, T^{(2)}, \dots, T^{(m)}$ are the basic templates which compose T (Every reduced template is composed by the merger of a number of basic templates.), the expected size of T is equal to

$$E[r] = 2^{m(q-1) - \sum_{i=1}^{m-1} v_i}. \quad (4.14)$$

since the average size of a basic template is approximately 2^{q-1} . Since f (the updating function of the CA) has to have good randomness properties, especially for applications such as cryptography, its output has to be 1 for half of the 2^q inputs, and 0 for the other half.

Now, an approximate expression for the size of the final search space for a given number of bad cells is obtained; i.e., disagreement points between S_{k+1} and S_{k+1}^* . Let b denote the number of bad cells at a particular instance of the algorithm, and V denotes total size of intersections between consecutive basic templates; i.e., if v_i is the size of the intersection between consecutive basic templates $T^{(i)}$ and $T^{(i+1)}$ $i = 1, 2, \dots, b$,

$$v = \sum_{i=1}^b v_i. \quad (4.15)$$

After merging all intersecting templates, a set of reduced templates will be obtained, $\{T_j\}$, with respective sizes $\{r_j\}$. The size of the search space, which will be denoted by U , equal to the product of the size of the final reduced templates, $\prod_j r_j$. Assuming r_j 's are independent of each other, next, we obtain the following expression for the expected size of the search space, $E[U]$, where B is the random variable for the number of bad cells and V is the random variable for the total size of intersections:

$$\begin{aligned}
E[U \mid B = b, V = v] &= E\left[\prod_j r_j \mid B = b, V = v\right] = \prod_j E[r_j \mid B = b, V = v] \\
&= 2^{b(q-1)} \times 2^{-\sum_{i=1}^b v_i} = 2^{b(q-1)-v}
\end{aligned} \tag{4.16}$$

and

$$\begin{aligned}
E[U \mid B = b] &= \sum_v E[U \mid B = b, V = v] P(V = v \mid B = b) \\
&= \sum_v 2^{b(q-1)-v} P(V = v \mid B = b) = 2^{b(q-1)} \sum_v 2^{-v} P(V = v \mid B = b) \\
&\approx 2^{b(q-1)-E[V \mid B=b]}
\end{aligned} \tag{4.17}$$

Now, the problem is to find a lower bound for $E[V \mid B=b]$ that is expected total size of intersections between consecutive basic templates for a given number of bad cells. The total size of intersections can be found by counting the number of basic templates, which include a given cell. Let m_i denote the number of basic templates that include the cell x_i . m_i' is defined as $\max(0, m_i - 1)$, which implies $V = \sum_{i=1}^n m_i'$. Note that m_i is equal to the number of bad cells among q cells affecting x_i , where it is known that b cells among a total of n cells are bad. Thus, m_i is distributed with hypergeometric distribution (Davero, 1995) with parameters n, b, q , and therefore, the expected value of m_i is equal to bq/n . The expected value of V can be calculated as

$$E[V \mid B = b] = E\left[\sum_{i=1}^n m_i'\right] = \sum_{i=1}^n E[m_i'] = \sum_{i=1}^n (E[m_i] - 1 \cdot P(m_i \geq 1)) \geq \sum_{i=1}^n (E[m_i] - 1), \tag{4.18}$$

which gives

$$E[V \mid B = b] \geq n\left(\frac{bq}{n} - 1\right) = bq - n. \tag{4.19}$$

by substituting expression (4.19) in (4.17),

$$E[U | B = b] \leq 2^{b(q-1)-(bq-n)} = 2^{n-b} \quad (4.20)$$

Now, our goal is to get $E[U]$ from $E[U | B=b]$. Recall that S_{k+1} and S_{k+1}^* are equal to $f(S_k^*)$ and $g(S_k^*)$, respectively. Assuming that S_k^* is uniformly distributed (it takes all values with the same probability), the probability of $S_{k+1}[i]$ being not equal to $S_{k+1}^*[i]$ is $(1-\alpha)$, for $1 \leq i \leq n$. Hence, the probability

$$P(B = b) = \binom{n}{b} (1-\alpha)^b \alpha^{n-b}; \quad (4.21)$$

and

$$\begin{aligned} E[U] &= \sum_{b=0}^n E[U | B = b] P(B = b) = \sum_{b=0}^n E[U | B = b] \binom{n}{b} (1-\alpha)^b \alpha^{n-b} \\ &\leq \sum_{b=0}^n 2^{n-b} \binom{n}{b} (1-\alpha)^b \alpha^{n-b} = (2\alpha + (1-\alpha))^n = (1+\alpha)^n \end{aligned} \quad (4.22)$$

is obtained. So, an upper bound for the average size of the search space is found as

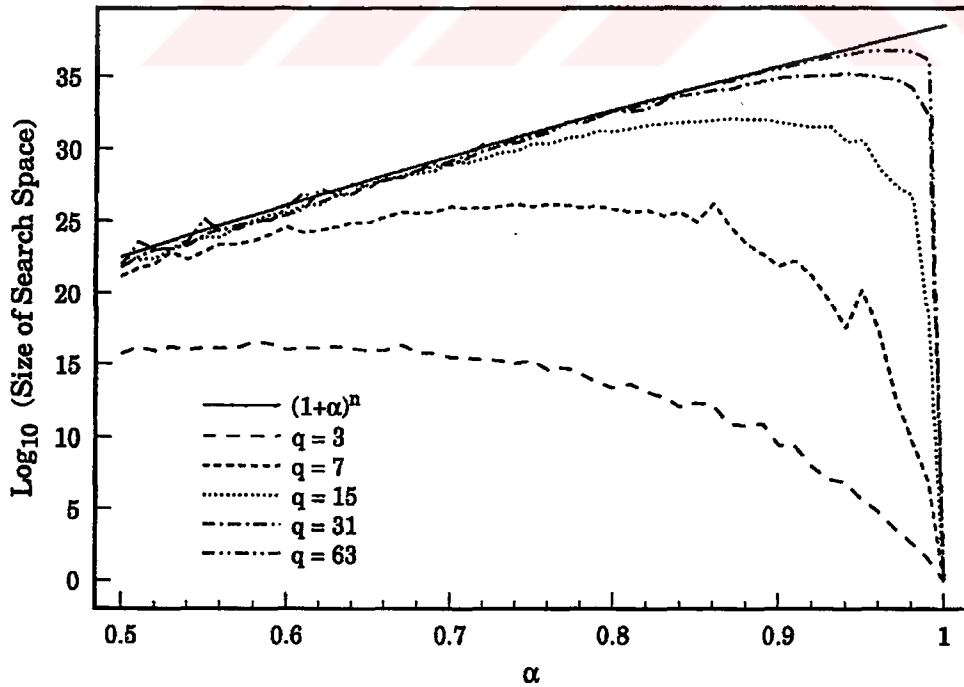


Figure 4.3. The experimental values for the size of the search space.

this bound, a large number of experiments are performed. The size of the space for several values of q and α are calculated for $n = 128$. For each value of q and α , the experiment is repeated on 1000 randomly generated sets of bad cells, assuming a basic template has 2^{q-1} rows and that the number of rows in a reduced template obtained by merging two templates with r_1 and r_2 rows and v common cells is equal to $r_1 r_2 2^{-v}$. The results of experiments are given in figure 4.3.

4.2.2. Computer Aided Analysis of Zigzag Algorithm

The given zigzag algorithm is applicable for all updating functions. Since each updating function defines a new system, entirely analytical derivation of execution time of zigzag algorithm appears impossible. Hence, some results about the zigzag algorithm are given by the help of the computer experiment.

Computer experiment has two important outcomes. The first outcome of the experiment is the estimation of whole algorithm's execution time. Some analytic results can only be given up to step 11 of the zigzag algorithm with probabilistic analysis in the previous section as an upper bound. The second outcome of the experiment is realized to give some results on halting steps of algorithm. It has shown that algorithm may halt on step 4 or step 11 or step 12.

During the experiment, cellular automata is forward iterated one step for all possible S_t , and their corresponding successors (S_{t+1}) are calculated: $S_t \xrightarrow{f} S_{t+1}$. Then, calculated successors (S_{t+1}) are taken as an input to the zigzag algorithm and termination steps of the zigzag algorithm are recorded with the execution time. Termination steps are used to find percentage of halting steps of zigzag algorithm. Correctness of solutions is also checked during the experiment.

Computer experiments are realized by using five 170 MHz Sun SPARC workstations. The programs are written by using Mathematica programming language. Source code of program is given in the appendix A. Graphical representation of data is also given in this section.

4.2.2.1. Empirical Derivation of Zigzag Algorithm Complexity.

The time complexity of exhaustive search will be given to compare time complexity that is found by computer experiment. The exhaustive time complexity can be given as follows for this experiment: First, the total number of operations can be formulated for given n and r : It is known that total number of possible configurations (S_i) is 2^n for size n automata and total number of radius r updating functions is $2^{2^{r+1}}$. Hence total number of possible different case is

$$2^{2^{r+1}} \times 2^n = 2^{n+2^{r+1}}. \quad (4.23)$$

Exhaustive search is $O(2^n)$ for a given S_i , n , r , in general case (4.23). Then, the expected complexity of this experiment for exhaustive search is

$$2^{n+2^{r+1}} \times 2^n = O(2^{n+2^{r+1}}). \quad (4.24)$$

The above formula shows that an experiment can only be realised for only small values of n and r , thus experiment is done for size (n) 7, 8, 9, 10, 11, 12, 13 cellular automata for all radius 1 updating functions. Thus expected complexity for $r=1$ is

$$O(2^{2n+8}) = 256 \times O(2^{2n}) \quad (4.25)$$

Analytically given formula that is $256 \times O(2^{2n})$, includes complexity of exhausted configurations and rules. It is known that number of configurations is 2^n and number of rules is 256. If analytic formula is normalised with respect to this complexity, $\frac{256 \times 2^{2n}}{256 \times 2^n} = 2^n$ is found as expected normalised complexity.

The experiment results which are given for all possible updating functions and configurations, are tabulated (table 4.1):

The given analytical analysis and plot of the data (figure 4.4) showed that time complexity of the zigzag algorithm is exponential. Hence, $\alpha 2^{Bn}$ is chosen

Table 4.1. Experimental results of execution time versus automata size.

n	Time (sec)
7	2074.88414
8	5527.07109
9	14758.28187
10	51225.46349
11	175493.96252
12	441458.59034
13	2204174.74720

nonlinear model to fit above data, where α and β are parameters, which will be found. Since chosen model has a property that is known as intrinsically linear, a linear regression method can be used by the help of some transformation (Davero, 1995).

Intrinsically Linear (Davero, 1995): A function relating y to n is intrinsically linear if by means of a transformation on x and/or y , the function can be expressed as $y' = \beta_0 + \beta_1 x'$, where x' is the transformed independent variable and y' is the transformed dependent variable.

Intrinsically linear functions lead directly to probabilistic models that, though not linear in x as a function, have parameters whose values are easily estimated using

Intrinsically linear functions lead directly to probabilistic models that, though not linear in x as a function, have parameters whose values are easily estimated using ordinary least squares. A probabilistic model relating y to x is intrinsically linear by means of a transformation on y and/or x , if it can be reduced to a linear probabilistic model $y' = \beta_0 + \beta_1 x'$.

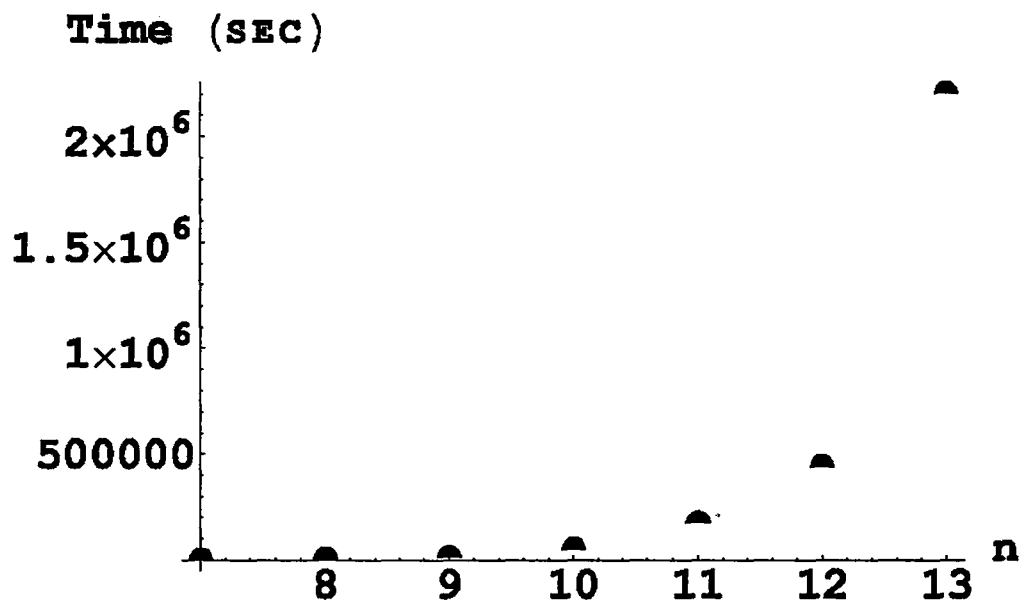


Figure 4.4 Plot of the experimental results.

$y = \alpha 2^{\beta x}$, a multiplicative exponential model, so that $\log_2(y) = y' = \beta_0 + \beta_1 x'$ with $x' = x$, $\beta_0 = \log_2(\alpha)$, $\beta_1 = \beta$. Denoting the size of the automata as the independent variable x , and execution time as the dependent variable y , the converted x' and y' values of above table are given in the table 4.2.

Table 4.2. Transformed experimental results for linear probabilistic model.

x'	y'
7	11.01882
8	12.43230
9	13.84924
10	15.64457
11	17.42106
12	18.75192
13	21.07181

The major advantage of an intrinsically linear model is that the parameters β_0 and β_1 of the transformed model can be immediately estimated using the principle of least squares simply by substituting x' and y' into the estimating formulas:

$$\hat{\beta}_1 = \frac{n \sum x'_i y'_i - \sum x'_i \sum y'_i}{n \sum (x'_i)^2 - (\sum x'_i)^2} = \frac{7 \times 1148.27 - 70 \times 110.19}{7 \times 728 - 4900} = 1.65607 \quad (4.26)$$

$$\hat{\beta}_0 = \frac{\sum y'_i - \hat{\beta}_1 \sum x'_i}{n} = \frac{110.19 - 1.6561 \times 70}{7} = -0.81934 \quad (4.27)$$

Parameters of the original nonlinear model can then be estimated by transforming back:

$$\alpha = 2^{\hat{\beta}_0} = 2^{-0.81934} = 0.56670 \approx 0.567 \quad \beta = \hat{\beta}_1 = 1.65607 \approx 1.656 \quad (4.28)$$

and estimated complexity is given as

$$O(0.567 \times 2^{1.656 n}). \quad (4.29)$$

It can be easily seen that proposed zigzag algorithm have better performance than the exhaustive search by comparing formula 4.2.2.1.3 and formula 4.2.2.1.7. If the estimated complexity is normalised with the 256×2^n , it is found as $0.002 \times O(2^{0.656 n})$ that is approximately square root of the exhaustive search space. The success of estimation can also be observed from the figure 4.5, which represents both the data points and the curve of estimated complexity function.

The time complexity function was derived analytically as $(1+\alpha)^n$ ($0.5 \leq \alpha \leq 1$) and empirical time complexity is calculated as $0.002 \times O(2^{0.656 n})$. The first function covers up to step 11 of the zigzag algorithm and the second function covers whole zigzag algorithm. Both functions can be compared by taking logarithm of them.

$$\text{Log}_2 (1+\alpha)^n = n \text{Log}_2 (1+\alpha) = \begin{cases} n & \alpha=1 \\ 0.585 n & \alpha=0.5 \end{cases} \quad (4.30)$$

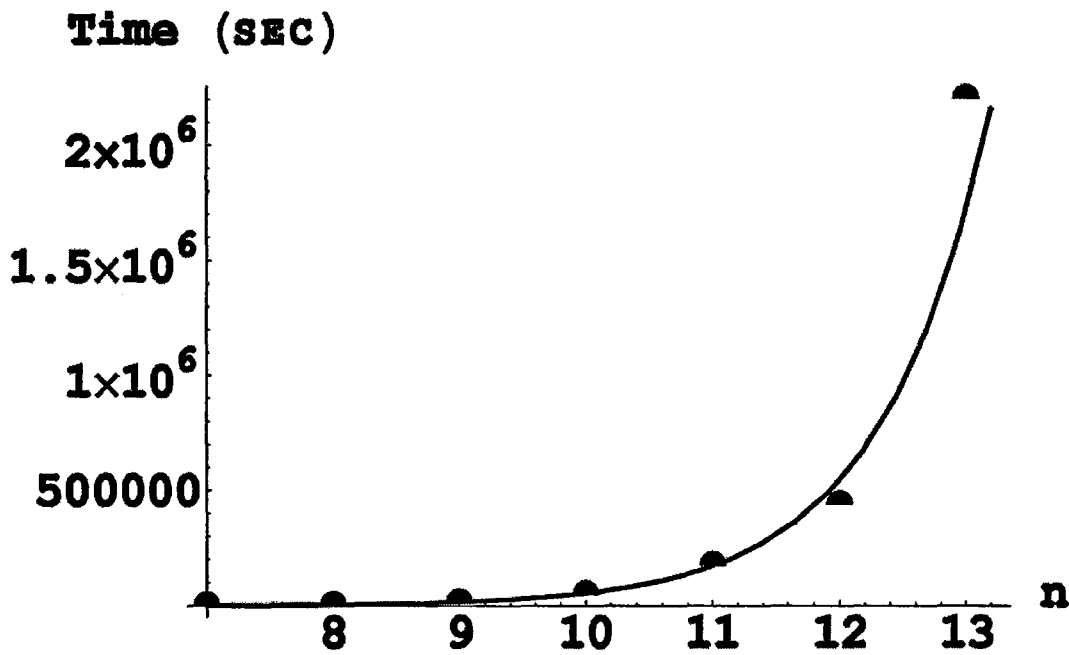


Figure 4.5. Plot of experimental results and curve of the calculated complexity function.

$$\log_2(0.002 \times 2^{0.656n}) = 0.656n - 8.966 \quad (4.31)$$

Since $(1+\alpha)^n$ was given for some part of the zigzag algorithm and it is greater than empirical time complexity for $\alpha \geq 0.656$, $(1+\alpha)^n$ is loose upper bound.

$0.002 \times O(2^{0.656n})$ can also be given as $O(2^{0.656n-8.966})$.

4.2.2.2. Analysis of Halting Steps of Zigzag Algorithm.

The zigzag algorithm has three different halting steps, which are step 4, step 11, and step 12. The zigzag algorithm finds solution and stops at one of these steps. The halting step depends on r , n , updating rule and S_k . The step 4 is essential, since the zigzag algorithm finds solution in polynomial time complexity, $O(nq^2)$, instead of exponential time complexity in this step. It is also fact that if the updating function is affine, the solution can be found in polynomial time complexity by just solving linear equations, but it is not true for nonlinear updating functions.

The zigzag algorithm can find solution for nonlinear updating function in polynomial time complexity for some instance of problem. The experiment is realised for all updating functions. The solutions that are enumerated for all affine functions must be removed from experimentally obtained results, to observe the success rate of zigzag algorithm in step 4 for nonlinear updating functions. The solutions in the polynomial time complexity can be counted as follows:

The updating functions are Boolean functions. The number of Boolean functions is given as 2^{2^m} , and number of affine functions is 2^{m+1} , where m is the number of arguments (Karpovsky, 1976). It was defined that m is $2r+1$, consequently expected percentage of solutions which are found at step 4, is

$$\frac{\text{Number of Affine Functions}}{\text{Number of Radius 1 Functions}} \times 100 = \frac{2^{m+1}}{2^{2^m}} \times 100 = \frac{16}{256} \times 100 = 6.25 \% \quad (4.32)$$

for $r = 1$.

These affine functions are 0, 15, 51, 60, 85, 90, 102, 105, 150, 153, 165, 170, 195, 204, 240 and 255. However, some of these affine functions construct noninvertible system of linear equations for cellular automata size n . The invertible functions give solutions in the $O(nq^2)$. The invertible functions that construct nonsingular matrices at step 2 of zigzag algorithm are tabulated below with respect to cellular automata size.

Table 4.3. The invertible functions for different size of cellular automata and theirs percentage with respect to whole set of functions.

n	Invertible functions	$\frac{\# \text{invertible}}{\# \text{functions}} \times 100$
7, 8, 9, 10, 11, 12, 13	15, 51, 85, 170, 204, 240	2.344 %
7, 8, 10, 11, 13	105, 150	0.781 %

The table 4.3 shows all invertible functions for different sizes of cellular automata. The number of invertible functions is 6 for size of 7, 8, 9, 10, 11, 12, 13 and 2 for size 7, 8, 10, 11, 13. Total number of functions is 256. It can be seen from the table 4.3 that expected success rate is 2.344 % for size 7, 8, 10, 11, 12, 13 and 0.781% for size 7, 8, 10, 11, 13 cellular automata.

The success ratio of zigzag algorithm, obtained by the experiment, are tabulated (table 4.4.) with respect to cellular automata size n:

Table 4.4. The success ratio of zigzag algorithm.

	n = 7	n = 8	n = 9	n = 10	n = 11	n = 12	n = 13
step 4	50,87 %	48,27 %	38,07 %	41,96 %	40,01%	32,86%	36,19%
step 11	37,77 %	39,50 %	53,03 %	42,34 %	42.47%	55,38%	43.62%
step 12	11,36 %	12,23 %	8,90 %	15,70 %	17.52%	11.76%	20.19%

The calculated (expected) and experimental results for step 4 are tabulated (table 4.5) below with respect to n.

Table 4.5. The success ratio of zigzag algorithm.

Success rate	n = 7	n = 8	n = 9	n = 10	n = 11	n = 12	n = 13
Expected	3.181	3.181	2.344	3.181	3.181	2.344	3.181
Experimental	50,87	48,27	38,07	41,96	40,01	32,86	36,19

Computer experiments show that 36.19 % of all configurations can be found in $O(nq^2)$ time for size 13 although expected success rate is 3.181 %, which shows that experimental success rate is approximately 10 times larger than the expected success rate. It was shown that complexity of zigzag algorithm is smaller than the exhaustive search. The essential point is that the zigzag algorithm finds solution for

some nonlinear updating functions as if they are linear functions. Hence it finds solution in polynomial time complexity $O(nq^2)$ instead of exponential time complexity $O(2^n)$ for some instances of the problem and success rate is unexpectedly high.

4.3. Presentation of Halting Steps of Zigzag Algorithms with The Examples

These examples, illustrating the application of the zigzag algorithm to the cellular automaton based on the updating function CA30, are constructed to describe certain properties of the zigzag algorithm.

The first step is to construct the best affine approximation of the function CA30. As was mentioned earlier, the best affine approximation of a function may not be unique. For instance, four affine functions are the best affine approximations of the CA30 updating function. These affine functions are:

$$g_1(x_1, x_2, x_3) = x_1 \oplus 1 \quad (4.33)$$

$$g_2(x_1, x_2, x_3) = x_1 \oplus x_2 \quad (4.34)$$

$$g_3(x_1, x_2, x_3) = x_1 \oplus x_3 \quad (4.35)$$

$$g_4(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3 \quad (4.36)$$

any of these affine functions $g_i(x)$ matches the function CA30 in $A = 6$ points out of 8, which gives $\alpha = 6/8 = 0.75$. This value of α is the maximum attainable among all affine functions. The function CA30 and its best affine approximations are tabulated (table 4.6) below:

Table 4.6. Affine functions of CA30.

x_{i-1}	x_i	x_{i+1}	CA30	$g_1(x)$	$g_2(x)$	$g_3(x)$	$g_4(x)$
0	0	0	0	1	0	0	0
0	0	1	1	1	0	1	1
0	1	0	1	1	1	0	1
0	1	1	1	1	1	1	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	0	0
1	1	0	0	0	0	1	0
1	1	1	0	0	0	0	1

Example 1:

S_k is an internal state of cellular automata where $k \in \mathbb{Z}$. Let $n=11$ and $S_k=01001000010$, which gives $S_{k+1}=11111100111$ under the updating function CA30. Now, assuming that S_k ($k \in \mathbb{N}$) is unknown and S_{k+1} is given, S_k can be calculated by using the proposed zigzag algorithm. First, the affine function $g_4(x)$ is selected, and a set of linear equations of dimension 11 is solved in order to compute S_k^* as explained in Step 2. The state vector obtained from this step is found as $S_k^*=01001000010$. The updating function CA30 is then applied to S_k^* to obtain S_{k+1}^* , which is found as $S_{k+1}^*=11111100111$.

	1	2	3	4	5	6	7	8	9	A	B		1	2	3	4	5	6	7	8	9	A	B
S_k	0	1	0	0	1	0	0	0	0	1	0	S_k^*	0	1	0	0	1	0	0	0	0	1	0
S_{k+1}	1	1	1	1	1	1	0	0	1	1	1	S_{k+1}^*	1	1	1	1	1	1	0	0	1	1	1

Since $S_{k+1}=S_{k+1}^*$, it is concluded that the algorithm successfully found the solution vector S_k at the end of Step 4. Although the updating function $f(x)$ is not linear, solution has been found immediately after solving a set of linear equations. As was explained in Section 3, this will happen when the seed vector (S_k) entirely consists of those q -tuples at which $f(x)$ and its best affine approximation $g(x)$ agree. The above table is inspected and noticed that $f(x)$ and $g_4(x)$ are equal at the following set of 3-tuples: $R_1 = \{ 000, 001, 010, 100, 101, 110 \}$. On the other hand, the set of all 3-tuples, which make up S_k is found as $R_2 = \{ 000, 001, 010, 100 \}$. Since $R_2 \subset R_1$, the application of $f(x)$ to S_k will produce the same S_{k+1} as the application of $g_4(x)$ to S_k .

Example 2:

If there are several predecessors of S_{k+1} , the proposed zigzag algorithm will find one of them, which may not be the original S_k used to obtain S_{k+1} . For example, let $n=8$ and $S_k=11100100$ which gives $S_{k+1}=10011111$ under CA30. After solving the linear equations for the affine function $g_4(x)$, we obtain $S_k^*=00001001$. We then apply CA30 to S_k^* , and calculate $S_{k+1}^*=10011111$.

	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
S_k	1	1	1	0	0	1	0	0	S_k^*	0	0	0	0	1	0	0	1
S_{k+1}	1	0	0	1	1	1	1	1	S_{k+1}^*	1	0	0	1	1	1	1	1

Since $S_{k+1}=S_{k+1}^*$, the algorithm has indeed found a predecessor of S_{k+1} . However, this predecessor is not the original S_k from which we computed S_{k+1} . The application of the other best affine approximations of $f(x)$ may yield other predecessors of S_{k+1} .

Example 3:

In this example, $n=11$ and $S_k=11001101001$ which gives $S_{k+1}=00111001111$ under the updating function CA30, are considered. The affine function $g_1(x)$ is selected, a set of linear equations is solved, and $S_k^*=10001100001$ is calculated, and finally CA30 is applied to S_k^* to calculate $S_{k+1}^*=01011010011$.

	1	2	3	4	5	6	7	8	9	A	B		1	2	3	4	5	6	7	8	9	A	B
S_k	1	1	0	0	1	1	0	1	0	0	1	S_k^*	1	0	0	0	1	1	0	0	0	0	1
S_{k+1}	0	0	1	1	1	0	0	1	1	1	1	S_{k+1}^*	0	1	0	1	1	0	1	0	0	1	1

As can be seen from above the state vector S_{k+1}^* is not equal to S_{k+1} , and thus Steps 5 through 12 of the zigzag algorithm need to be executed. The state vectors S_{k+1} and S_{k+1}^* differ in bit positions 2, 3, 7, 8, and 9. Thus, five templates are construct as shown below:

	1	2	3		2	3	4		6	7	8
a	0	0	0	e	0	0	1	i	0	0	0
b	1	0	1	f	0	1	0	j	1	0	1
c	1	1	0	g	0	1	1	k	1	1	0
d	1	1	1	h	1	0	0	l	1	1	1

	7	8	9		8	9	A
m	0	0	1	q	0	0	1
n	0	1	0	r	0	1	0
o	0	1	1	s	0	1	1
p	1	0	0	t	1	0	0

There are two groups of templates, based on the neighbourhood principle. The first group contains the templates 2 and 3, while the second group consists of the templates 7, 8, and 9. The templates in the first group are combined by retaining those rows of the templates that have the same bit values. For example, the row 'a' in the template 2 and the row 'e' in the template 3 are combined making a row corresponding to the bit positions 1, 2, 3, and 4. The reduced templates are given below:

	1	2	3	4		6	7	8	9	A
ae	0	0	0	1	Imr	0	0	0	0	0
bf	1	0	1	0	Ims	1	0	1	1	0
bg	1	1	0	1	Jnt	1	1	0	1	1
ch	1	1	1	0	Kpq	1	1	1	1	1

All rows in the first reduced template are exhaustively checked and those rows that complement bit values in positions 2 and 3 are kept, while the other bits are not being altered. The only row in the first reduced template is the row 'ch'. Similarly, in the second reduced template, the row 'jnt' complements the bit values in positions 7, 8, and 9, while not altering the other bit values. Therefore, the predecessor of the state S_{k+1} , is given as

1	2	3	4	5	6	7	8	9	A	B
1	1	0	0	1	1	0	1	0	0	1

where the bit values in positions 1, 2, 3, 4 come from the row 'ch', the bit values in positions 6, 7, 8, 9, A come from the row 'jnt', while the bit values 5 and B are the original values of S_k^* .

Example 4:

Let $n=11$ and $S_k=01101000010$ which gives $S_{k+1}=11001100111$ using the updating function CA30. In this example, the affine function $g_1(x)$ is selected and $S_k^*=01100110000$ is calculated by solving a set of linear equations. Then CA30 is applied to S_k^* and $S_{k+1}^*=11011101000$ is computed.

	1	2	3	4	5	6	7	8	9	A	B		1	2	3	4	5	6	7	8	9	A	B
S_k	0	1	1	0	1	0	0	0	0	1	0	S_k^*	0	1	1	0	0	1	1	0	0	0	0
S_{k+1}	1	1	0	0	1	1	0	0	1	1	1	S_{k+1}^*	1	1	0	1	1	1	0	1	0	0	0

By inspection it is determined that S_{k+1} and S_{k+1}^* differ in bit positions 4, 8, 9, A, and B. Therefore, the templates are found as

	3	4	5		7	8	9		8	9	A
a	0	0	0	e	0	0	0	i	0	0	1
b	1	0	1	f	1	0	1	j	0	1	0
c	1	1	0	g	1	1	0	k	0	1	1
d	1	1	1	h	1	1	1	l	1	0	1

	9	A	B		A	B	1
m	0	0	1	q	0	0	1
n	0	1	0	r	0	1	0
o	0	1	1	s	0	1	1
p	1	0	0	t	1	0	0

The template 4 is put into the first group by itself, while the second group consists of the templates 8, 9, A, and B. The row 'b' is the only row in the first reduced template, which complements the 4th bit while not altering the other bit values. The reduced templates of these two groups are given below:

	3	4	5		7	8	9	A	B	1
b	1	0	1	eint	0	0	0	1	0	0
				fjq	1	0	1	0	0	1
				glmr	1	1	0	0	1	0
				glms	1	1	0	0	1	1

However, when each row is tried in the above-reduced templates in S_k^* , the correct S_{k+1}^* is not produced. The bit values outside the templates may also need to be changed. Thus, step 12 of the zigzag algorithm is executed; in the worst case, all untouched bits (bits 2 and 6) may need to be altered. To obtain the correct solution vector, bit 6 should be complemented, and bit 2 should be kept in the same order.

1	2	3	4	5	6	7	8	9	A	B
0	1	1	0	1	0	0	0	0	1	0

Here, bits 3, 4, 5 come from the first reduced template (row 'b'), bits 7, 8, 9, A, B, 1 come from the second reduced template (row 'eint'), and bit 6 is the complement of the original bit 6.



5. APPLICATIONS OF PROPOSED ZIGZAG ALGORITHM

Two applications of the proposed algorithm will be given in this section. The first one is finding the key of the Wolfram's (1986) cryptographic pseudorandom number generator. The second one is calculation of collision value at the Damgard's (1990) hash function.

5.1. Applications of Zigzag Algorithm to Wolfram's Pseudorandom Number Generator.

A random sequence generator based on the one-dimensional cellular automaton with $q=3$ and the so-called CA30 updating function

$$f(x_{i-1}, x_i, x_{i+1}) = x_{i-1} \oplus (x_i \vee x_{i+1}) \quad (5.1)$$

was proposed by Wolfram in (Wolfram, 1985). The state vectors produced by this cellular automaton seem to have randomness properties, e.g., the time sequence values of the central cell shows no statistical regularities under the usual randomness tests (Knuth, 1981). In order to use such a generator for cryptographic purposes, the seed value (the initial state vector S_0) must be difficult to be constructed given a sequence of state vectors. It was stated in (Wolfram, 1985) that

This problem is in the class NP. No systematic algorithm for its solution is currently known that takes a time less than exponential in n .

While it is true that an exponential number of trials required to compute S_0 given one or more S_k for $k \geq 1$ for general updating functions and arbitrary values of S_0 , it is shown that the number of trials may be much fewer for particular classes of updating functions and seed values. An algorithm is given to compute S_k for a given S_{k+1} and $f(x)$. The running time of the algorithm depends on the updating function $f(x)$ and state S_{k+1} . If $f(x)$ is an affine function, then the running time of the algorithm

is only $O(nq^2)$. Furthermore, the running time of the zigzag algorithm may still be $O(nq^2)$ for nonlinear updating functions and certain seed values. The number of trials is exponential in n , but fewer than 2^n for nonlinear updating functions and arbitrary seed values.

The inversion of the cellular automata for general updating functions and seed values is an NP-complete problem since it is equivalent to the general problem of solving Boolean equations. It has also been shown by Meier and Staffelbach (1991) that the sequence of the central bits of the states, which is named as the temporal sequence, can be used to determine the seed using the partial linearity of the CA30 updating function. Given the temporal sequence, their method successfully finds the seed value when its size is up to 500 bits for radius 1 functions that are partially linear. The zigzag algorithm differs from their approach in the sense that it computes the predecessor of a given state vector. However, the method in (Meier, Staffelbach, 1991) can be coupled with the zigzag algorithm. First, their method is used to compute a state vector, given the central bit values. The zigzag algorithm can then be used to compute the predecessors of this state. Thus, the entire evolution of the state machine that generates the random numbers can be mapped.

Although the Meier-Staffelbach algorithm is applicable to only functions, which have a partial linearity and radius 1, zigzag algorithm does not have any restriction on the function. The proposed zigzag algorithm is based on the best affine approximation of the q -variable updating function $f(x)$.

Wolfram's pseudorandom number generator and its only known cryptanalysis method that is presented by Meier-Staffelbach (1991). However, Wolfram's generator is still accepted as a cryptographically secure with the different parameters (Madjarova, 1997).

Wolfram's generator is based on rule 30 cellular automata with periodic boundary condition. Probability of agreement (α) was given as 0.5 for rule 30 with the best affine approximation in the section 3.1.1. Thus complexity of attack up to step 11 of algorithm is

$$k \times O((1+\alpha)^n) = k \times O(1.5^n) \quad (5.2)$$

where k represents number of states from key to state which is found by the Meier-Staffelbach algorithm. The expected complexity of attack for the whole algorithm is for any updating function

$$O(k \times 2^{0.656n - 8.9658}) = O(2^{0.656n + \log_2(k) - 8.9658}) \quad (5.3)$$

However exhaustive search runtime complexity is $O(2^n)$.

The proposed zigzag algorithm is applied to Wolfram's pseudorandom number generator by the help of the Meier-Staffelbach. It is apparent that coupling of these two methods constructs powerful cryptanalysis tool for Wolfram pseudorandom number generator and it has been first of all shown that the key of the Wolfram's pseudorandom number generator can be found in $O(2^{0.656n + \log_2(k) - 8.9658})$ time complexity for any updating function.

5.2. Applications of Zigzag Algorithm to Damgard Hash Function.

At the Crypto'89 Damgard presented a method that allows one to construct a computationally collision free hash function that is based on cellular automata. Later Daemen et al (1991) showed that Damgard hash function has a collision. However, they did not give a method to find collision values. Damgard's hash function and attack of Daemen are presented in this thesis and collisions have been found. Although Daemen et.al. proved that Damgard's hash function has collision, Daeman was not able to give an algorithm to calculate collision values.

The zigzag algorithm is also applied to Damgard cellular automata based hash function and it has been shown that possible input message values can be obtained by the help of the Meier-Staffelbach attack. Hence, zigzag algorithm can be used to find collision values. The zigzag algorithm also inverts the hash function and it is no more one way function that is essential criteria for hash functions. However, messages that are given as an input to hash functions, should not be calculated from the hash values for secure hash functions (Pieprzyk, Sadeghiyan, 1991).

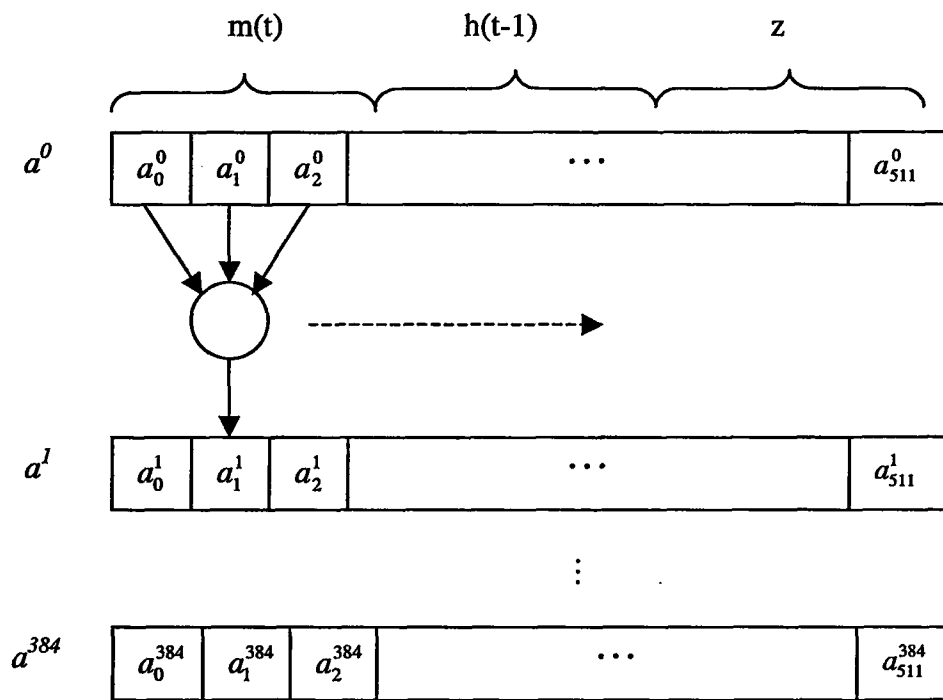


Figure 5.1. Schematic representation of Damgård hash function.

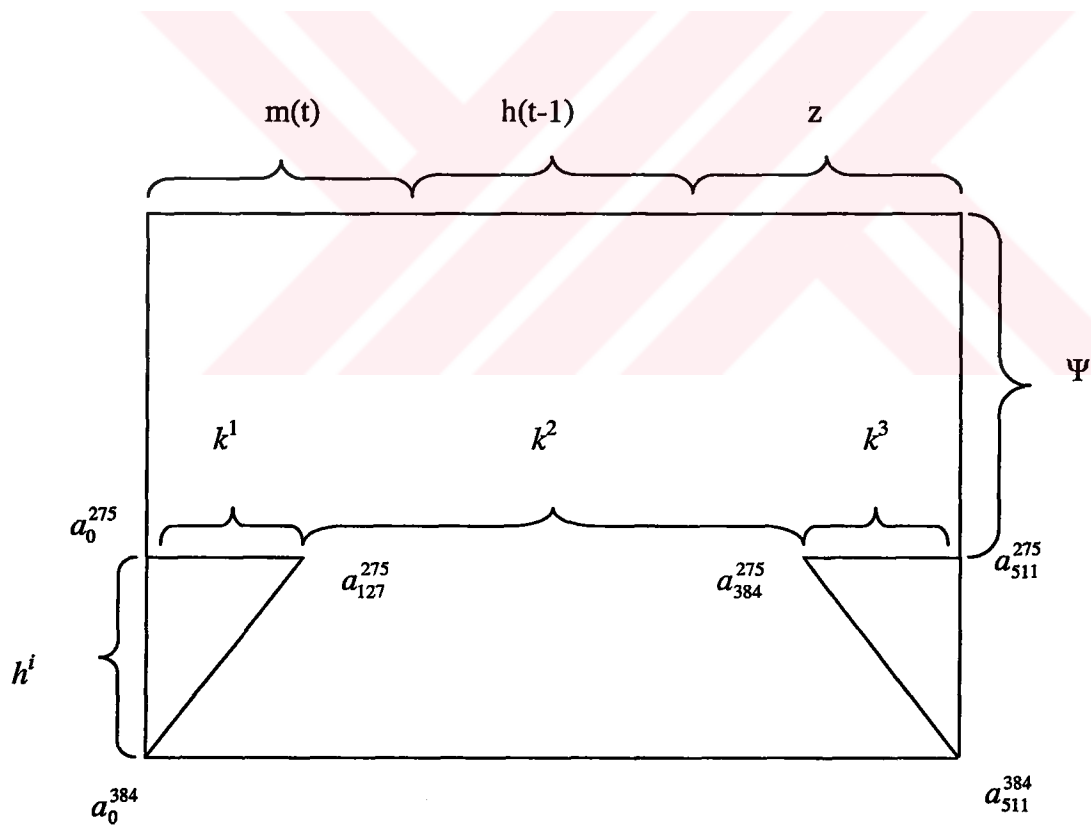


Figure 5.2. Schematic representation of zigzag algorithm.

Application of zigzag algorithm is given as follows: Definition of hash operation is given as

$$h^i = g_c(h^{i-1} \parallel m_{i-1}) \quad \text{for} \quad i = 1, \dots, n. \quad (5.4)$$

h^n is known value of hash result and $g_c(.)$ is realised by cellular automata with periodic boundary condition (Chapter 3.2.2.). Application of Meier-Staffelbach algorithm to h^n gives us vectors k^1 and k^2 which are parts of the a^{257} (figure 5.1 and figure 5.2). But still some bits of the a^{257} are unknown and can not be calculated with any algorithm, since these bits (k^3) does not have any effect on h^n . Thus, unknown bits are randomly chosen, therefore k^3 is constructed by random values. It is clear that 257 number of bits are randomly chosen and naturally total possible of selections are 2^{257} . This calculation based on assumption of slope is -4 that is stated in Section 3.2.2. Thus a^{257} that is concatenation of vectors k^1 , k^2 and k^3 , is prepared for zigzag algorithm. a^{256} is calculated by using a^{257} and zigzag algorithm. The same calculation is repeated up to a^0 . Hence, a^0 that is hashed message m_n , is found. Of course, each randomly chosen k^2 may produce different a^0 . All of these are collision values for these hash functions. Nevertheless, real message can be found as follows: a^0 is composed of x and z by definition of the hash function and z is constant and known value. Since z is a part of the a^0 that is found by the zigzag algorithm, the found a^0 is the correct a^0 . Although the other a^0 are not the real messages, they are still useful as collision values. Hence, zigzag algorithm gives useful results in both cases.

Expected complexity of this operation can be given as follows: $257 \times O(2^{0.656n-8.9658})$ where 257 is the number of states from the state a^{257} to a^0 . Hence, complexity of attack is $O(2^{0.656n-0.9602})$ instead of $O(2^n)$.

CONCLUSIONS

The concept of backward iteration and then forward iteration with the error correction algorithm is a novel idea for cryptanalysis of GF(2) based cryptographic algorithms and this idea is successfully realised and mounted on some CA based cryptographic algorithms. A new cryptanalysis algorithm based on the idea that is named zigzag algorithm was given in the detailed 12 algorithmic steps.

The algorithm is general in terms of updating function and the size of CA. Since each updating function defines a new system, it is extremely difficult to derive general analytic formula for execution time. Consequently, analytically derived time complexity of zigzag algorithm could be given partially. Thus, an upper bound on execution time of zigzag algorithm was given as $O((1+\alpha)^n)$ up to step 11, where $0.5 \leq \alpha \leq 1$. However, this upper bound is extremely loose due to undertaken assumptions to overcome the difficulty of analytic analysis.

A computer experiment is performed because of the difficulty of analytic derivation of time complexity function. The empirical time complexity function was obtained by curve fitting to experimental data. Experimental data was obtained by running the zigzag algorithm on SUN Sparc workstations. Therefore, an empirical time complexity function is obtained as $2^{0.656n-8.9658}$ for whole algorithm (up to step 12). This bound is slightly larger than the analytically found bound, since empirical bound covers the whole algorithm steps. The empirical time complexity is approximately square root of the exhaustive search $O(2^n)$. Therefore, it may be concluded that the given zigzag algorithm is successful. Furthermore, for some instance of the problem, time complexity of zigzag algorithm decreases from $O(2^n)$ to $O(nq^2)$, i.e., the time complexity function reduces from exponential to polynomial function.

The concept of the zigzag algorithm can be applied to a variety of cryptographic systems, for instance, the proposed zigzag algorithm was applied to

two different cryptographic algorithms: The first one was Wolfram's (1985) pseudorandom number generator. It was shown that, the security of Wolfram generator was reduced further. The second one was about Damgard cryptographic hash function. It was shown that collision values could be calculated by the help of the zigzag algorithm.

In this thesis, it was shown by help of the given zigzag algorithm that some CA based cryptographic systems do not have presumed expected security. Thus, the given zigzag algorithm is useful for some cryptanalytic attacks, moreover the concept of the zigzag algorithm can be generalised for cryptographic systems, which are based on different algebraic systems. The zigzag algorithm also requires further analysis and works for different types of CA based cryptographic algorithms, for example (Kari, 1992). The zigzag algorithm can be used for different area of science because of several applications of CA (Toffoli and Margolus, 1987). To conclude, the purpose of this thesis is the presentation of a novel zigzag algorithm and its fundamental concepts.

REFERENCES

- APOHAN, A. M. (1994)**, Interface Design for RSA Processor PQR6, *Marmara Research Center Technical Report*, T1-1, March.
- APOHAN, A. M., YÜZBAŞIÇ, E. (1993)**, A Review of DES and RSA Encryption Algorithms with Software Implementation of DES on TMS320C20, *Marmara Research Center Technical Report*, T1-1, December.
- BARDELL, P. H. (1990)**, *Analysis of Cellular Automata Used as Pseudorandom Pattern Generators Proceedings*, International Test Conference, 1990, 762-767.
- BIHAM, E., SHAMIR, A. (1993)**, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, Berlin.
- BEAUCHAMP, K. G. (1975)**, *Walsh Functions and Their Applications*, New York, NY: Academic Press.
- BEAUCHAMP, K. G. (1984)**, *Applications Walsh and Related Functions*, Academic Press.
- BENNETT, C. H., BRASSARD, G., EKERT, A. (1992)**, Quantum Cryptography, *Scientific American*, pp.50-57, October.
- BLAKE I. F., GAO X., MULLIN R. C., VANSTONE S. A., YAGJOOBAN T., (1993)**, Applications of Finite Fields, In A. J. Menezes editor, pages 7-8. Massachusetts, Kluwer Academic.
- BRASSARD, G. (1995)**, A Quantum Jump in Computer Science, J. V. Leeuwen, *Current Trends in Computer Science*, LNCS 1000, Springer-Verlag.
- BLACKBURN, S., R., MURPHY S., PATERSON K., G., (1997)**, Comments on *Theory and Applications of Cellular Automata in Cryptography*, *IEEE Transactions on Computers*, Vol. 46, No 5, May, 1997.
- COPPERSMITH, D. (1994)**, The Data Encryption Standard (DES) and Its Strength Against Attacks, *IBM Journal of Research and Development*, Vol. 38, no 3, pp 243-250, May.
- CRYPTTECH (1988)**, RSA processor PQR6, Functional Manual Version No. 1.0, CRYPTTECH, February.

- DAEMEN, J., GOVAERTS R., VANDEWALLE, J. (1991)**, *A framework for the design of one-way hash functions including cryptanalysis of Damgard's one-way function based on a cellular automaton*, Advances in Cryptology-Asiacrypt'91, Lecture Notes in Computer Science, No. 739, pages 82-96, New York, Springer-Verlag, Japan.
- DAEMEN, J., GOVAERTS R., VANDEWALLE, J. (1992a)**, *A hardware design model for cryptographic algorithms*, Computer Security-ESORICS 92, Lecture Notes in Computer Science, No. 648, pages 419-434, New York, Springer-Verlag, France.
- DAEMEN, J., GOVAERTS R., VANDEWALLE, J. (1992b)**, *A New Approach To Block Cipher Design*, Fast Software Encryption 92, Lecture Notes in Computer Science, No. 648, pages 419-434, New York, Springer-Verlag, France.
- DAMGARD, I. B. (1990)**, *A design principle for hash functions*, Advances in Cryptology -CRYPTO 89, Lecture Notes in Computer Science, No. 435, pages 416-427, New York, NY: Springer-Verlag.
- DAVERO, J. L. (1995)**, Probability and statistics for engineering and the sciences, CA, Wadsworth Inc.
- DAS, A. K., GANGULY, A., DASGUPTA, A., BHAWMIK, S., CHAUDHURI, P. P. (1990)**, *Efficient Characterization of Cellular Automata*, IEE Proc., Vol. 137, pt. E, No 1.
- DIFFIE, W., HELLMAN, M. E. (1977)**, Exhaustive Cryptanalysis of the NBS Data Encryption Standard, Computer, vol. 10, pp. 74-84, June.
- DIFFIE, W., HELLMAN, M. E. (1976)**, *New Directions in Cryptography*, IEEE Transactions on Information Theory, Vol. IT-22, pp. 644-654, November, 1976.
- DING, C., XIAO, G., SHAN, W. (1991)**, The Stability Theory of Stream Ciphers, New York, NY: Springer-Verlag.
- GAREY, M. R., JOHNSON (1979)**, Computers and Intractability A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York, 1979.
- GECKINLI, N. C., APOHAN, A. M. (2000)**, *Power Spectrum Tests of Random Numbers*, to appear on Signal Processing.
- GOLOMB, S. W. (1982)**, Shift Register Sequences, Laguna Hills, CA: Aegean Park Press.
- GUAN, P. (1987)**, *Cellular Automaton Public Key Cryptosystems*, Complex Systems, 1.

- GUTOWITZ, H. (1993)**, Cryptography with Dynamical Systems, In N. Boccara et al editor, Cellular Automata and Cooperative Systems, pg 237-274, Kluwer Academic, Netherlands.
- HELLMAN, M. E. (1980)**, *A Cryptanalytic Time-Memory Trade-Off*, IEEE Trans. Information. Theory, vol. IT-26, pp. 401-406, July.
- KAHN, D. (1976)**, The Codebreakers: The Story of Secret Writing, MacMillan Co., NewYork.
- KARI, J., (1992)**, Cryptosystem Based on Reversible Cellular Automata,
- KARPOVSKY, M. G. (1976)**, Finite Orthogonal Series in the Design of Digital Devices, New York, NY: John Wiley & Sons.
- KNUTH, D. E. (1981)**, The Art of Computer Programming: Seminumerical Algorithms, volume 2 Reading, MA: Addison-Wesley, Second edition.
- KOÇ, Ç., K., APOHAN, A. M. (1997)**, *Inversion of Cellular Automata Iterations*, IEE Proceedings-Computing Digital Technique, Vol. 144, No. 5, September.
- MADJAROVA M., KAKUTA M., YAMAGUCHI M., OHYAMA N. (1997)**, *Optical Implementation of The Stream Cipher Based on The Irreversible Cellular Automata Algorithm*, Optics Letters, Vol. 22, No 21, November.
- MASSEY, J. L. (1969)**, *Shift Register Synthesis and BCH Decoding*, IEEE Trans. Information Theory, vol. IT-16, pp. 122-127, January.
- MASSEY, J. L. (1995)**, Cryptography: Fundamentals and Applications, Advanced Technology Seminars Notes, Zurich.
- MATSUI, M. (1994)**, *Linear Cryptanalysis Method for DES Cipher*, Advances in Cryptology, Eurocrypt'93 Proceedings, Springer-Verlag, Berlin.
- MATTHEWS, R. (1989)**, *On The Derivation Of A Chaotic Encryption Algorithm*, Cryptlogia, 13:1.
- MEIER, W., STAFFELBACH, O. (1991)**, *Analysis of pseudo random sequences generated by cellular automata*, In D.W. Davies, editor, Advances in Cryptology - EUROCRYPT 91, Lecture Notes in Computer Science, No. 547, pages 186-199. New York, NY: Springer-Verlag.
- MENEZES A. J., OORSCHOT P. C. V., VANSTONE S. A. (1997)**, Handbook of Applied Cryptography, CRC Press, NewYork.
- MEYER, C, H, MATYAS, S. M. (1982)**, Cryptography: A New Dimension In Computer Security, John Wiley & Sons.

- MICALI, E., PONCINO, M. (1996),** *Using Symbolic Rademacher-Walsh Spectral Transforms to Evaluate the Agreement Between Boolean Functions*, IEE Proc.-Computing Digital Technique, Vol. 143, No. 1.
- MITCHELL, D. W. (1990),** *Nonlinear Key Generators*, Cryptologia, Volume:14, Number:4.
- MORAGA, C. (1985),** On Some Applications of The Cherestenson Functions in Logic Design and Data Processing. Mathematics in Simulation, 27, pp. 431-439, North-Holland.
- NANDI, S., CHAUDHURI, P. P. (1996),** *Analysis of Periodic and Intermediate Boundary 90/150 Cellular Automata*, IEEE Transactions on Computers, Vol. 45, No 1.
- NANDI, S., KAR, B. K., CHAUDHURI, P. P. (1994),** *Theory and Applications of Cellular Automata in Cryptography*, IEEE Transactions on Computers, Vol. 43, No 12, December, 1994.
- NEUMAN, J. V. (1966),** Theory of Self-Reproducing Automata Arthur Burks, editor, Univ. of Illinois Press.
- PIEPRZYK, J., SADEGHIYAN, B. (1991),** Design of Hashing Algorithms, Springer-Verlag, Berlin.
- RIVEST, R. L., SHAMIR, A., ADLEMAN L. (1978),** *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communication of ACM, Vol. 21, pp. 120-126, February.
- ROTHAUS, O. S. (1976),** On Bent Functions, J. Comb. Theory, Series A20, pp. 305-330.
- ROSS, S. M. (1989),** Introduction to Probability Models, Academic Press, New York.
- RUEPPEL, R. A. (1986),** Analysis and Design of Stream Ciphers, New York, NY: Springer-Verlag.
- RUEPPEL, R. A. (1992),** Stream ciphers, In G.J. Simmons, editor, Contemporary Cryptology: The Science of Information Integrity pages 65-134. New York, NY: IEEE Press.
- SCHNEIER, B. (1996),** Applied Cryptography Second Edition: Protocols, Algorithm, and Source Code in C, John Wiley & Sons, Inc.
- SHANNON, C. E. (1949),** Communication Theory of Secrecy System Bell System Technical Journal, Vol. 28, pp. 656-715, October.

- SIEGENTHALER, T. (1984)**, *Correlation Immunity of Nonlinear Combining Functions for Cryptographic Applications*, IEEE Transactions on Information Theory, vol. IT-30, pp. 776-780.
- SERRA, M., MILLER, D. M., MUZIO J. C. (1988)**, *Linear Cellular Automata and LFSRs are Isomorphic*, Proceedings of Third Technical Workshop on New Direction for IC Testing, Halifax, pp. 213-223.
- TOFFOLI, T., MARGOLUS. N. H. (1987)**, Cellular Automata Machines, MIT Press Series in Scientific Computation.
- TOFFOLI, T., MARGOLUS. N. H. (1990)**, *Invertible cellular automata: A review*, Physica D, 45:229-253.
- WEBSTER, A. F., TAVARES, S. E.(1986)**, *On The Design of S-boxes*, Advances in Cryptology, Proceedings of Crypto'85, pages 523-534, Springer-Verlag.
- WHEELER, D., D., (1991)**, *Problems With Chaotic Cryptosystems*, Cryptologia, Volume:13, Number:3.
- WHEELER, D., D., MATTHEWS, A. J. (1991)**, *Supercomputer Investigations Of Chaotic Encryption Algorithm*, Cryptologia, Volume: 15, Number: 2.
- WOLFRAM S., MARTIN O., ODLYZKO A. M. (1984)**, Communications in Mathematical Physics, volume 93, March.
- WOLFRAM, S. (1984)**, *Computation Theory of Cellular Automata*, Communications in Mathematical Physics, Volume: 96, November.
- WOLFRAM, S. (1985)**, *Cryptography with cellular automata*, In H.C. Williams, editor, Advances in Cryptology - CRYPTO 85, Proceedings, Lecture Notes in Computer Science, No. 218, pages 429-432. New York, NY: Springer-Verlag.
- WOLFRAM, S. (1986)**, *Random sequence generation by cellular automata*, Advances in Applied Mathematics, Vol 7, pp 123-169.
- WOLFRAM, S. (1994)**, Cellular Automata and Complexity-Collected Papers, Addison-Wesley,
- XIAO, G., MASSEY, J., L. (1985)**, *A spectral Characterization of Correlation-Immune Functions*, IEEE Transactions on Information Theory, Vol. IT-34, pp. 569-571, May, 1988.
- ZADEH, N. (1973)**, *A bad network problem for the simplex method and other minimum cost flow algorithms*, Mathematical Programming 5, 255-266

APPENDIX A.

(* Zigzag Algorithm *)

```
(* zigzag[ size_of_cellular_automata, start_rule, stop_rule];*)
(* Necessary packages. *)
(* ca.m : Cellular automata packages, Telos publication. *)
<<ca.m;
(* comb.m : Combinatorial packages. Combinatorica book, Steven Scania *)
<<comb.m;
(* Statistic packages, Standart Mathematica package.*)
<<Statistics`DataManipulation`;
Needs["Statistics`DataManipulation`"];
getvect[tblono_,satirno_] := (indirgentablo[[prttnt[[tblono,Length[prttnt[[tblono]]]]]] [[satirno]] );

birles[tb1_, tb2_, dist_] := Module[{i,j},
tb12 = Table[{ }, {0}]; dimtb11 = Dimensions[tb1][[1]]; dimtb21 =
Dimensions[tb2][[1]]; dimtb12 = Dimensions[tb1][[2]];
If[dist == 2, For[i = 1, i <= dimtb11, i++, For[j = 1, j <= dimtb21, j++,
    If[tb1[[i]][[ dimtb12 ]] == tb2[[j]][[1]], tb12 =
Append[tb12,Flatten[Append[Drop[tb1[[i]], -1], tb2[[j]]]]];];];
,    If[dist == 1,
    For[i = 1, i <= dimtb11, i++,
    For[j = 1, j <= dimtb21, j++,
        If[tb1[[i]][[ dimtb12 - 1]] == tb2[[j]][[1]] && tb1[[i]][[ dimtb12
]]== tb2[[j]][[2]],
            tb12 = Append[tb12, Flatten[Append[Drop[tb1[[i]], -2],
tb2[[j]]]]];
                ];];];
];
Return[tb12];
];

H[np_] := (-1)^Map[Dot[Strings[{0,1},np],#]&, Strings[{0,1},np]];
skart[ ] := ( sk[[n]] = sk[[n]]+1;For[i = n, i>1, i--, If[sk[[i]] > 1, sk[[i]] = 0;sk[[i-1]] =
sk[[i-1]] + 1; ]]);
Dorduncuadim[]:=
sk1 = UpdateCA[sk,carule];
If[flag != 1,
    If[S[[MaxPoz[[affinesyc]]]] < 0,
        sky = LinearSolve[mtrx,Mod[sk1+1,2],Modulus->2];
    , sky = LinearSolve[mtrx,sk1,Modulus->2];
    ];
```

```

, If[S[[5]] < 0,
    sky = LinearSolve[mtrx, Mod[sk1+1, 2], Modulus->2];
    , sky = LinearSolve[mtrx, sk1, Modulus->2];
];
];
sky1 = UpdateCA[sky, carule];
hatarlar1 = Flatten[Position[Abs[sky1-sk1], 1]];
);

Initforrule[] :=(
sk = Table[0, {n}];
carule = NumberedRule[kuralno];
For[i = 1, i <= 8, i++, f[[i]] = NumberedRule[kuralno][[9-i]][[2]]];
s = b = Table[{ }, {0}];
For[i = 1, i <= 8, i++,
    If[carule[[i, 2]] == 0,
        s = Append[s, carule[[i, 1]]];
        , b = Append[b, carule[[i, 1]]];
    ];
];
S = H[3] . ((-1) ^ f);
MaxPoz = Flatten[Position[Abs[S], Max[Abs[S]]]];
);

Onikinciadim[] := (
digerbitler = Table[i, {i, 1, n}];
fltyaz = Flatten[yaz];
For[i = 1, i <= n, i++,
    If[MemberQ[fltyaz, i] == True,
        digerbitler = Drop[digerbitler, Flatten[Position[digerbitler, i]]];
    ];
lengthdigerbit = Length[digerbitler]; vect = Table[0, { lengthdigerbit}];
denemebound = 2^lengthdigerbit;
For[deneme = 0, deneme < denemebound, deneme++,
    For[i = 1, i <= lengthdigerbit, i++, skytest[[digerbitler[[ i ] ]]] = vect[[i]]; ];
    tabloaridene[];
    If[skytest1 == sk1, Break[]];
    vect[[lengthdigerbit]] = vect[[lengthdigerbit]]+1;
    For[i = lengthdigerbit, i > 1, i--, If[vect[[i]] > 1, vect[[i]] = 0; vect[[i-1]] = vect[[i-1]]
+ 1;];];
];);

Onbirinciadim[] :=(
circfark = indirgentablo = Table[0, {n}];
prtn = Table[{ }, {n}];
lengthhata = Length[hatarlar1];
For[i = 1, i <= n, i++, If[sk1[[i]] == 0, tablist[[i]] = s; , tablist[[i]] = b; ]; ];
For[i = 1, i < lengthhata, i++, circfark[[i]] = hatarlar1[[i + 1]] - hatarlar1[[i]]; ];
circfark[[i]] = n - hatarlar1[[lengthhata]] + hatarlar1[[1]];

```

```

If[MemberQ[circfark,0]==True, circfark = Drop[circfark, -(Length[circfark] -
Flatten[Position[circfark, 0]][[1]] +1)]; ];
lmt = Flatten[Position[circfark, x_ /; x > 2]];
group = 1;
If[lmt != {},
lengthlmt = Length[lmt];
For[i = lengthlmt, i > 1, i--,
For[j = lmt[[i]], j > lmt[[i - 1]], j--, prtn[[group]] = Prepend[prtn[[group]],
hatalar1[[j]]];]; group++; ];
tmp = lmt[[1]];
For[j = tmp, j >= 1, j--, prtn[[group]] = Prepend[prtn[[group]], hatalar1[[j]]];];
tmp = lmt[[lengthlmt]];
For[j = lengthhata, j > tmp, j--, prtn[[group]] = Prepend[prtn[[group]],
hatalar1[[j]]];]; , prtn[[group]] = hatalar1; ];
For[i = 1, i <= group, i++,
qq = tablist[[prtn[[i,1]]]]; tmp = Length[prtn[[i]]];
For[j = 2, j <= tmp, j++, qq =
birles[qq,tablist[[prtn[[i,j]]]],circfark[[Flatten[Position[hatalar1,prtn[[i,j-
1]]]]][[1]]];
];
indirgentablo[[prtn[[i, j-1]]]] = qq;
];
yaz=prtn=Append[prtn,{ }];
minflatposprt = Min[Flatten[Position[prtn,{ }]]];
For[i = 1, i < minflatposprt, i++,
yaz[[i]] = Union[yaz[[i]]+1,yaz[[i]]-1,yaz[[i]]]; tmp = Length[yaz[[i]]];
For[j=1, j <= tmp, j++, If[ yaz[[i, j]] == n+1, yaz[[i,j]] = 1;, If[yaz[[i,j]] == 0,
yaz[[i,j]] = n;];]; ];
yaz[[i]] = Sort[ yaz[[i]] ];
If[ yaz[[i,1]] == yaz[[i,2]], yaz[[i]] = Rest[yaz[[i]] ]; ];
lengthyaz = Length[yaz[[i]]];
If[ yaz[[i, lengthyaz-1]] == yaz[[i, lengthyaz]], yaz[[i]] = Drop[yaz[[i]],-1];];
];
For[i = 1, i < minflatposprt, i++, check = prtn[[i,1]]-1;
If[check == 0, check = n;];
rot = 0; lengthyaz = Length[yaz[[i]]];
For[ j = lengthyaz, j > 1, j--, rot++;
If[yaz[[i,j]] == check, yaz[[i]] = RotateRight[yaz[[i]],rot]; Break[]; ]; ];
];
skytest = sky;
tablolaridene[];
);
tablolaridene[]:=
sayac = Table[1,{group}]; sayac [[group]] = 0; size = 1;
For[i = 1, i <= group, i++, size =size*
Dimensions[indirgentablo[[prtn[[i,Length[prtn[[i]]]]]]][[1]]];];
For[denemesayisi = 1, denemesayisi <= size, denemesayisi++,
i = group; flag=1;
While[flag == 1,
If[sayac[[i]] !=Dimensions[indirgentablo[[prtn[[i,Length[prtn[[i]]]]]]][[1]],

```

```

    flag = 0; sayac[[i]]++;, sayac[[i]] = 1; i-- ; ];
];
For[groupsay = 1, groupsay <= group, groupsay++,
    tmp = Length[yaz[[groupsay]]];
    For[i = 1, i <= tmp , i++,
        skytest[[yaz[[groupsay, i]] ]] = getvect[groupsay,sayac[[groupsay]]] [[i]]; ];
    skytest1 = UpdateCA[skytest,carule];
    If[skytest1 == sk1, denemesayisi = size; ];];
);

```

```

(* starting rule : baslangic
halting rule: bitis
size of automata : n*)
baslangic = 172; bitis = 182; n = 13;
vctsayisi = 2^n;
f = Table[0,{8} ];
tablist = Table[0,{n}];
timelist = Table[0,{bitis-baslangic+1}];
adimdort = adimonbir = adimoniki = 0;
verisayaci = 1;
veriler=Table[0,{bitis-baslangic+1},{3}];
baslangiczamani = Date[];
zaman =Timing[
For[kuralno = baslangic, kuralno <= bitis, kuralno++,
    adimzaman = Timing[
    Initforrule[];
    For[vctcnt = 1, vctcnt <= vctsayisi, vctcnt++, birtaneyapti = 0; mtrxsak = bos;
        For[affinesyc = 1, affinesyc <= Length[MaxPoz], affinesyc++,
            Switch[MaxPoz[[affinesyc]],
                1, el = {0,0,0}, 2, el = {0,0,1}, 3, el = {0,1,0}, 4, el = {0,1,1},
                5, el = {1,0,0}, 6, el = {1,0,1}, 7, el = {1,1,0}, 8, el = {1,1,1}
            ];
            mtrx = RotateRight[Table[Table[RotateRight[Join[el,Table[0,{n-3}]],i],{i,0,n-1}]]];
            If[ Det[mtrx, Modulus->2] != 0, mtrxsak = mtrx; affinesak = affinesyc; flag = 0;
                Dorduncuadim[];
                If[sk1 == sky1, birtaneyapti = 1;adimdort++;Break[]];
            ];
        ];
    ];
    If[birtaneyapti == 0,
        If[mtrxsak == bos, el = {1,0,0};
            mtrx=RotateRight[Table[Table[RotateRight[Join[el,Table[0,{n-3}]],i],{i,0,n-1}]]]; flag = 1; Dorduncuadim[];
            , mtrx = mtrxsak; affinesyc = affinesak; flag = 0; Dorduncuadim[];
        ];
        If[sk1 == sky1, birtaneyapti = 1;adimdort++;
        , Onbirinciadim[ ];
        If[skytest1 == sk1, adimonbir++;, Onikinciadim[];
            If[skytest1 == sk1, adimoniki++;, Print["Basarisiz "];Exit[]];];
    ];
];

```

```

];
];
];
birtaneyapti =0;
skart[];
];
];
Print[kuralno," ",adimdort," ",adimonbir," ",adimoniki,"
",adimdort+adimonbir+adimoniki];
veriler[[verisayaci,1]] = adimdort;
veriler[[verisayaci,2]] = adimonbir;
veriler[[verisayaci,3]] = adimoniki;
adimdort = adimonbir = adimoniki = 0;
timelist[[verisayaci]] = adimzaman[[1]];
verisayaci++;
];
]
bitiszamani = Date[];
sumdort=0;
For[i=1 , i < verisayaci ,i++, sumdort = sumdort + veriler[[i,1]]];
sumonbir =0;
For[i=1, i < verisayaci , i++, sumonbir = sumonbir + veriler[[i,2]]];
sumoniki =0;
For[i=1, i < verisayaci , i++, sumoniki = sumoniki + veriler[[i,3]]];
Print[sumdort," ",sumonbir," ",sumoniki," ",sumdort+sumonbir+sumoniki];
fname = ToString[n]<"-">ToString[baslangic]<"-"
">ToString[bitis]<". ">ToString[dat];
Save[fname,veriler ];
time = zaman[[1]];
Save[fname,time ];
Save[fname,baslangiczamani ];
Save[fname,bitiszamani ];
Save[fname,timelist];

```

BIOGRAPHY

A. Murat APOHAN was born in Istanbul. He graduated from Technical University of Istanbul, Department of Electronic and Communication Engineering and obtained M.Sc. degree from Technical University of Istanbul, Department of Electronic and Communication Engineering.

He has been working as a chief researcher in TUBITAK National Research Institute of Electronics & Cryptology since 1991. He was awarded from several projects.

