Cellular Automata and Cryptography

Tiago Santos

Dissertação de Mestrado apresentada à
Faculdade de Ciências da Universidade do Porto em
Ciência de Computadores

2014

Cellular Automata and Cryptography

Tiago Santos
Mestrado em Ciência de Computadores
Departamento de Ciências de Computadores
2014

Orientador
Prof. Rogério Reis, DCC

Coorientador
Prof. António Machiavelo, DMAT

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________

“The advancement and diffusion of knowledge is the only guardian of true liberty.”

James Madison

Abstract

This thesis gives an introduction to cellular automata and their applications in cryptography. We can find references to cryptographic techniques as far as the 1900 B.C.. Since then, cryptography has evolved and is now one of the cornerstones of our information society. From electronic payments to database security, passing through access control, messages authentication and corporative security, cryptography has become ubiquitous in our world. Historically speaking, the study of cellular automata is a very recent field of research. However its popularity has been steadily increasing, not only because of their ability to model complex dynamic systems, but also, due their inherent simplistic and high parallelizable structure. In this thesis, we start by introducing the basic concepts of cryptography and cellular automata, presenting some of the best examples of cryptographic systems with cellular automata as their main component. Then we focus our attention to one of the few public key systems based on cellular automata in existence, proposed by Kari. We analyze Kari's proposal by giving a more detailed description. We also discuss an algorithm of this cryptosystem and provide some critics on its implementation and on its security.

Resumo

Esta tese apresenta uma introdução aos autómatos celulares e a sua aplicação à criptografia. Criptografia é uma área historicamente velha, existindo registos históricos datados de 1900 B.C que a mencionam. Desde então, a criptografia tornou-se num dos pilares da nossa sociedade da informação. Devido à sua importância, a criptografia é neste momento ubíqua em vários domínios, desde os pagamentos eletrónicos à segurança corporativa e de bases de dados, passando pelo controlo de acessos e assinaturas digitais. Historicamente falando, os automatos cellulares são ainda um campo de investigação muito recente. No entando, a sua popularidade tem estado a crescer, não só devido às suas capacidades em modelar sistemas dinâmicos complexos, mas também, devido à sua natureza simplista e estrutura altamente paralelizável. Nesta tese, começamos por introduzir os conceitos básicos de criptografia e autómatos celulares, apresentando depois os melhores exemplos de aplicações de sistemas criptográficos que utilizam os autómatos como um dos seus principais componentes. Depois focamos a nossa atenção para um sistema de chave pública baseado em autómatos celulares, proposto por Kari. Analisamos este sistema e apresentamos-lo de uma forma detalhada. Finalizamos com a apresentação e discussão de uma implementação deste sistema, fornecendo críticas sobre o seu algoritmo e a sua segurança.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisors Prof. Rogério Reis and Prof. António Machiavelo for the continuous support of my MSc study and research, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

To all my professors that contributed to my academic formation.

To everybody, thank you.

Contents

List of Figures

List of Tables

Abbreviations

AES	**A**dvanced **E**ncryption **S**tandard
CA	**C**ellular **A**utomata
DES	**D**ata **E**ncryption **S**tandard
ECA	**E**lementary **C**ellular **A**utomata
FDM CA	**F**ixed **D**omain **M**arker **C**ellular **A**utomaton
ICA	**I**nvertible **C**ellular **A**utomata
IDEA	**I**nternational **D**ata **E**ncryption **A**lgorithm
KDC	**K**ey **D**istribution **C**enter
LCA	**L**inear **C**ellular **A**utomata
LFSR	**L**inear **F**eedback **S**hift **R**egister
LMCA	**L**inear **M**emory **C**ellular **A**utomata
MCA	**M**emory **C**ellular **A**utomata
PCA	**P**rogrammable **C**ellular **A**utomata
PGP	**P**retty **G**ood **P**rivacy
PRNG	**P**seudo **R**andom **N**umber **G**enerator
RCA	**R**eversible **C**ellular **A**utomata

Chapter 1

Introduction

1.1 Motivation

Due the ubiquity of digital communications and digital data in today's world, the development of techniques and tools to protect them have become increasingly more important. This required protection encompasses a set of requirements and solutions where cryptography is a primary pillar. Among the cryptographic goals we find secrecy, authenticity and preservation of the integrity of the data that it is protecting. We can split cryptographic systems in two groups: symmetric-key and public-key systems. In the first group, the same key is used both for encryption and the decryption. In the second, encryption and decryption are done using different keys.

Cellular automata (CA) is a decentralized computing model capable of providing a platform for performing complex computations using only local information. They are composed by a number of components, called cells, with local connectivity. Each cell is at one state at a time and its temporal change follows a simple transition rule that depends on both the current state and on the states of its neighboring cells.

Despite its simple structure, cellular automata can be as expressive as Turing machines [TM90]. Due to their capacity of executing complex computations with high efficiency and robustness, they can be a very useful tool for simulating the behavior of complex systems. They have been the subject of rigorous mathematical analysis and their applications have extended into a myriad of fields, such as, simulations of gas behavior, percolation processes and forest fire propagation, in the conception of massive parallel computers and in the study of urban development. Together with its easy implementation in parallel architectures, cellular automata have become a very interesting and efficient tool in the cryptographic field. However they have been mainly employed in symmetric key cryptosystem.

The main goal of this thesis is to explore how cellular automata have been used in cryptography, and therefore we start by exposing the most relevant cryptosystems that make use of cellular automata, including pseudo-random generators, secret sharing schemes, S-Boxes, hash functions and others. Because of both the scarcity of CA-based public key systems and the lack of a good description of these systems, we have dedicated an entire chapter to a particular system and to an implementation of it. Our main goal is to attempt to provide a more clear and detailed description of it.

1.2 Thesis outline

The thesis is organized as follows. We start by introducing the basic concepts of cryptography in chapter 2, presenting its goals, systems and main characteristics.

In chapter 3, we present a brief introduction to CA, the history, formal definitions, main characteristics, applications and classification.

Chapter 4 displays an overview of the most relevant cryptosystems and works on cryptography that uses CA as their main component.

In chapter 5, we present and discuss Kari's CA-based public key cryptosystem and an algorithm for its implementation.

Finally, chapter 6 ends the thesis with a short conclusion and a presentation of a road-map for future research.

Chapter 2

Fundamentals of Cryptography

2.1 Cryptology

Cryptology is a very broad term. It encompasses both the study of techniques related to all aspects of data security as well the study of the principles and methods of transforming one message into another (see figure 2.1). We can say, cryptology is the science and study of cryptanalysis and cryptography.

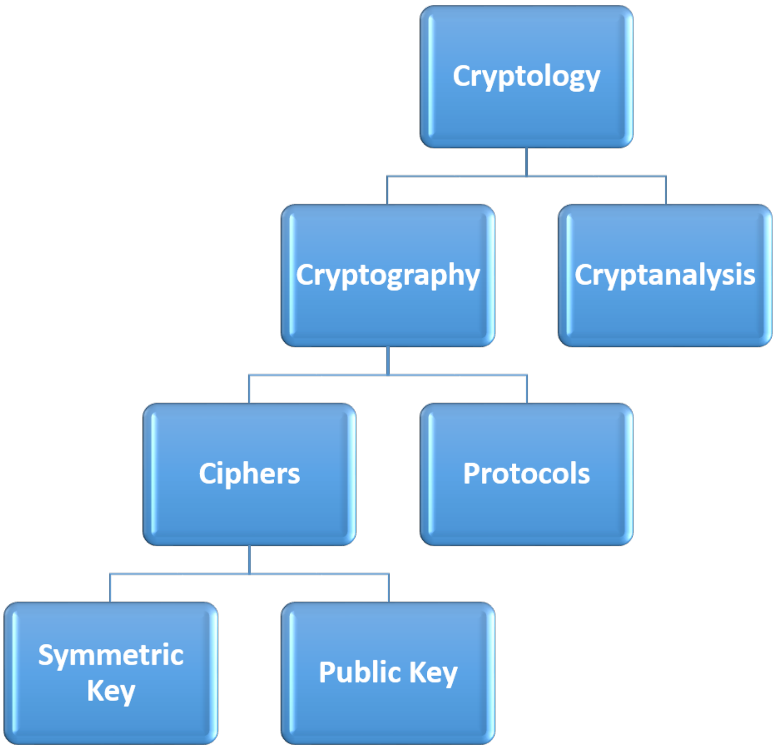


FIGURE 2.1: The field of cryptology.

Cryptography is a crucial part of any secure communication. The need of security and privacy in communications has been an essential necessity since antiquity, with references dating back as far as 1900 B.C. [WM07]. It has extended far beyond its initial application in communications. Now we see cryptography in countless areas, from database content protection to communications authentication.

Despite its age, most modern cryptographic techniques are based on advances made in the past few decades. Singh's book [Sin00] provides a fascinating look at the long history of encryption and its applications. Extensive overviews on cryptographic systems can be found in many classical works by authors as Schneier [Sch95], Stallings [Sta11] and Menezes [MVO96].

2.2 The Importance of Cryptography

Two communicating entities, which we will call Alice and Bob, want to communicate with each other securely. As examples of such entities we find web browser/server for electronic transactions, on-line banking client/server, DNS servers or routers exchanging routing table updates. Another entity, Eve, acts as an intruder with the goal of interfering with the communication between Alice and Bob. Among her goals we find message modification, connection hijack and impersonation, among others.

Consider the following security risks that Alice and Bob can face when communicating in an unprotected channel, as illustrate in figure 2.2.

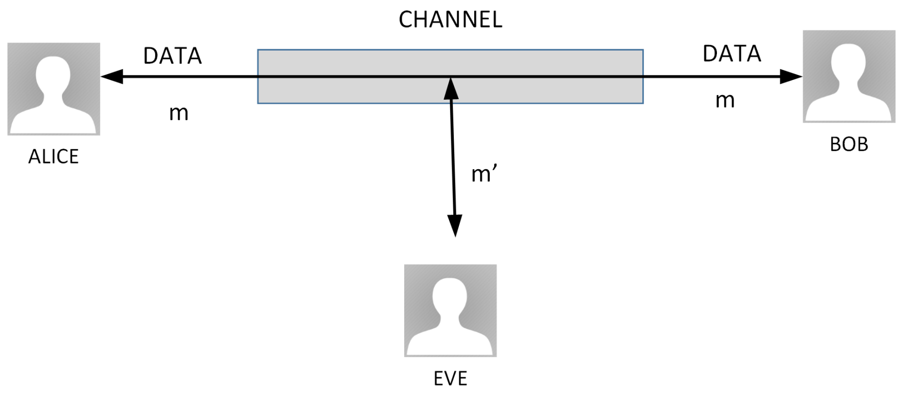


FIGURE 2.2: Unprotected communications.

If Eve intercepts and understands all secret messages by eavesdropping on the communication, then we have *loss of privacy* and *confidentiality*.

On the other hand, if Eve is able to modify the message while in transit, and if either Bob or Alice are unable to detect this, then we have *loss of integrity*.

If Eve sends messages to Bob, by pretending to be Alice, and Bob is unable to verify the source of the information then we *lack authentication*.

Both Bob and Alice can also *repudiate* any message sent by each other.

From all the above, we can see that security must be used on both sides of the communication channel. The solution for these problems passes by the use of cryptography to secure the channel and the data (see figure 2.3).

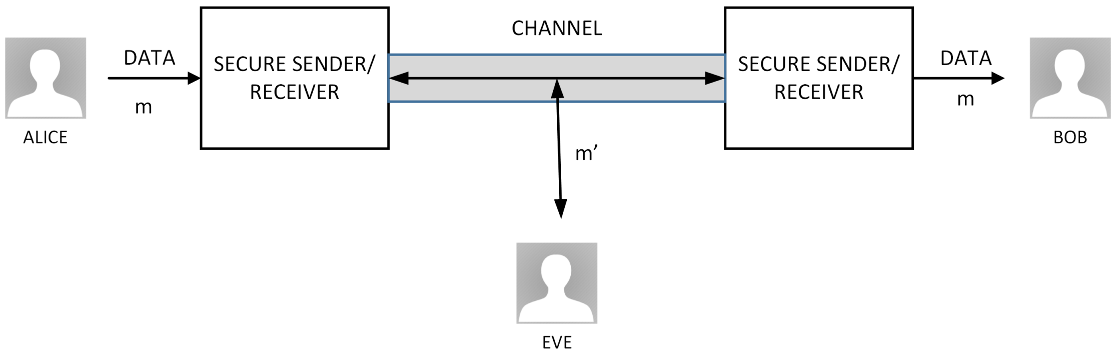


Figure 2.3: Protected communications.

We are now able to identify the following cryptographic goals [MVO96]: confidentiality, data integrity, authenticity and non-repudiation. Confidentiality or privacy keeps the contents of a message secret from all non-authorized third-parties. Integrity deals with the non-authorized alteration of data, including insertion, substitution and deletion. Authenticity deals with the identification of the message, its origins. Non-repudiation prevents an entity from denying previous actions or commitments.

2.3 Definition of a Cryptosystem

When can define a cryptosystem as a five-tuple $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{M}$ is a finite set of plaintexts (original messages), $\mathcal{C}$ a finite set of ciphertexts (coded messages), $\mathcal{K}$ a finite set of keys (keyspace), $\mathcal{E}$ a finite set of enciphering transformations $e_{k_1} : \mathcal{M} \to \mathcal{C}$ and

$\mathcal{D}$ a finite set of deciphering transformations $d_{k_2} : \mathcal{C} \to \mathcal{M}$. For each key $k_1 \in \mathcal{K}$ there is an encryption transformation $e_{k_1} \in \mathcal{E}$ and a corresponding decryption transformation $d_{k_2} \in \mathcal{D}$. Also, for each e_{k_1} and d_{k_2} : $d_{k_2}(e_{k_1}(m)) = m, \forall m \in \mathcal{M}$.

A representation of a generic cryptosystem can be seen in figure 2.4.

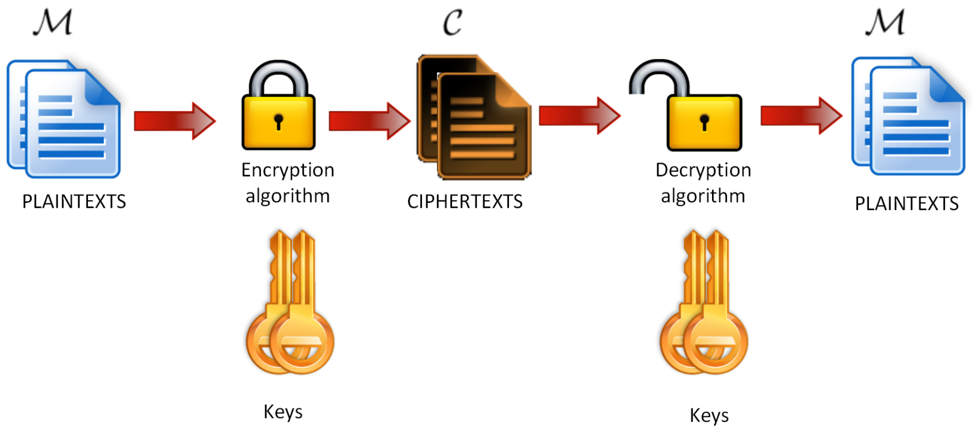


FIGURE 2.4: A generic cryptosystem.

A cryptosystem can be characterized according to:

- The type of operations used in the transformation of the plaintext into ciphertext.
- Number of keys used, usually single key for private or symmetric systems and two-key for public or asymmetric systems.
- How the plaintext is processed: block or stream cipher.

2.4 Private-Key Cryptosystems

In these type of cryptosystems, all communicating parties share a common secret key, $k_1 = k_2 = k$, which is used both for the encryption and the decryption operations (see figure 2.5).

There are hundreds if not thousands of symmetric key algorithms. Examples include the Data Encryption Standard (DES) [NBoS77], the International Data Encryption Standard (IDEA) [Lai92], the Advanced Encryption Standard (AES) [PT01], the RC5 [Riv94] and the RC6 [RRSY].

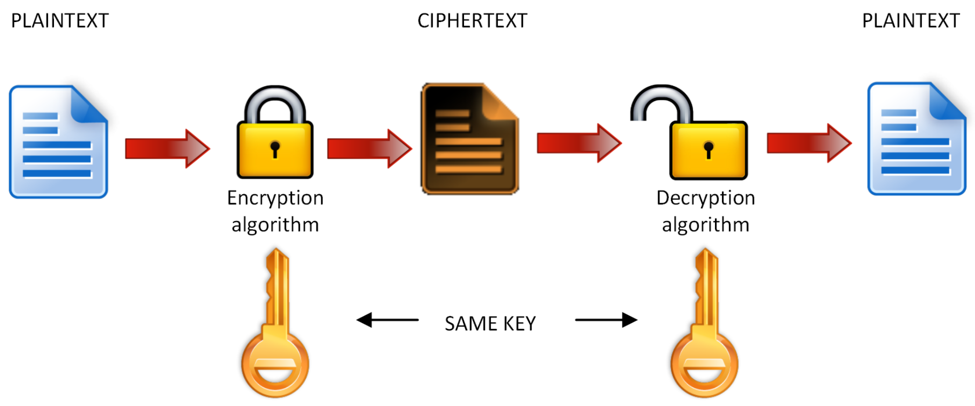


FIGURE 2.5: Symmetric key operation.

Symmetric key systems are much faster than public key cryptography, in the order of 1000 to 10000 faster, and even faster if they are implemented with dedicated hardware.

Despite their good performance and security, they do have some significant downsides. A serious problem is the key management problem. This issue appears even in small-scale networks. As the number of users increases, the number of keys required to provide secure communications exhibit a quadratic growth. For example, a network of N users requires at least $N(N-1)/2$ keys and in this case each user would need to keep $N-1$ keys in secret. As we can observe in figure 2.6, both the numbers of keys and the number of secure channels to distribute the keys rapidly starts to increase with the number of users. There is also the necessity for a secure channel for the exchange of the secret

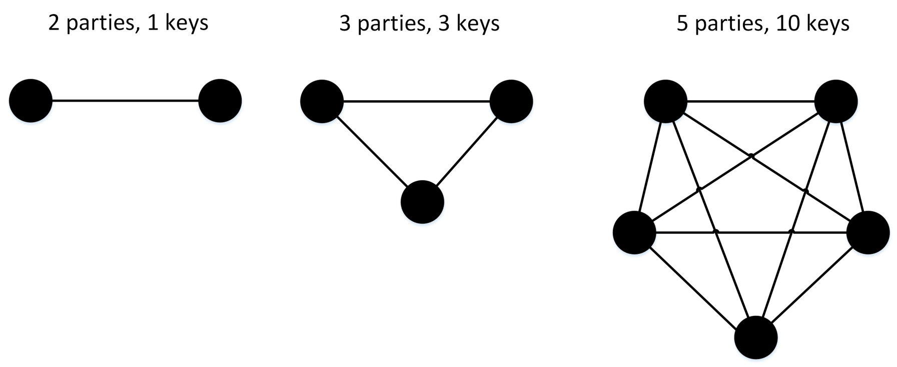


FIGURE 2.6: The problem of the numbers of keys in symmetric key systems.

key. To solve this issue, Deffie-Hellman methods or a trusted key distribution center (KDC) acting as intermediary between entities are used. Also if a single secret key is shared between N users, then it requires only one to betray the secret to compromise all communications. Since both parties share the same key, there are also problems with the authenticity of a message.

We can see that there are four requirements for the secure use of symmetric encryption: a strong encryption algorithm, a secret key known only to sender / receiver, always assume that the encryption algorithm is known and a secure channel to distribute key.

2.5 Public-Key Cryptosystems

Public key or asymmetric systems were developed to address two key issues: how to have secure communications without having to trust a KDC with the key and how to verify if a message comes intact from the claimed sender (digital signatures).

The idea of public-key cryptosystems was proposed by Diffie, Hellman [DH76] and Merkle. They demonstrated the principle with their Diffie-Hellman Key Exchange algorithm, which revealed to be a radically different and elegant approach for secure communications. Since this milestone, several others algorithms have emerged, such as RSA [RSA78]), Merkle-Hellman [MH78], El Gamal [Elg85] and Elliptic curves [Lan05], all of them still in use today.

In public key cryptography keys are created in related pairs, one for encryption and a different one for decryption, $k_1 \neq k_2$, a public key and a private key (see figure 2.7). The

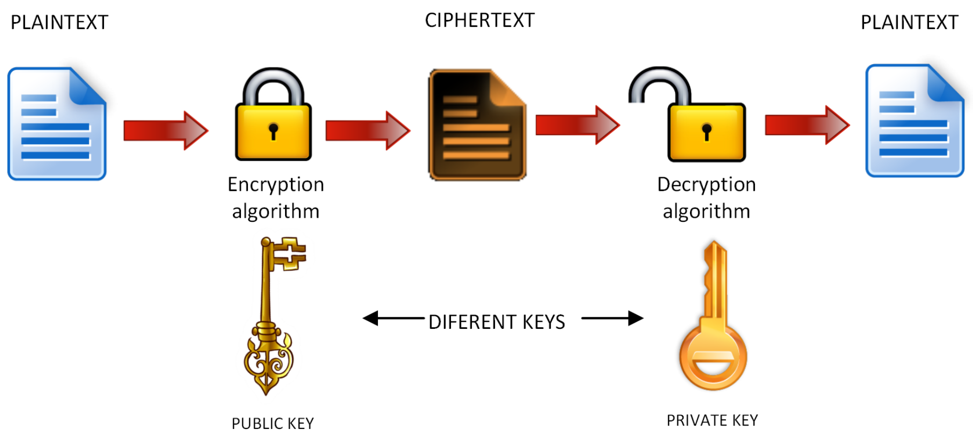


FIGURE 2.7: Asymmetric key operation.

public-key can be freely published without compromising the private one, which can also be known by anybody, and can be used to encrypt messages and to verify signatures A private key, which should be kept in secret, should only be known to the recipient, and it is used both to decrypt messages and to sign or create signatures.

Public key algorithms are based on mathematical functions. Furthermore the public and the private keys are related in such a way that even with the knowledge of the public key, it is virtually impossible to deduce the private key. Therefore, these cryptosystems depend on the existence of one-way functions. Despite the fact that there are no evidences on the existence of such functions, we can speculate their existence, by considering some assumptions about their complexity to be true. Examples of such functions include multiplication/factorization and exponentiation/logarithms. The key point is to find a trap door in these function so that it becomes relatively easy to find their inverse given some knowledge.

Public key systems present then three main characteristics:

- It is computationally infeasible to find the decryption key knowing only the algorithm and the encryption key.
- It is computationally easy to encrypt and decrypt messages when the respective key is known.

Because public key systems requires the use of very large numbers and expensive mathematical operations, these systems are slow when compared with symmetric key schemes. Generally, encrypting large messages using public key systems can be considered impractical. Therefore, public key cryptography should be considered a complement rather than a replacement of the private key cryptography: use a public key cipher (such as RSA) to distribute keys and use a private key cipher (such as DES) to encrypt and decrypt messages.

Deterministic public-key systems are susceptible to dictionary attacks. But, if public key systems are used only to encrypt small messages, such as keys, these types of attacks are not a practical problem. The most significant attack against these systems is the *man-in-the-middle* attack. In this attack, Eve intercepts messages sent by Bob to Alice or by Alice to Bob. Since Eve knows the public keys of both Alice and Bob, all she needs is to hijack the communication channel between Alice and Bob during the exchange of their public key, by sending her public key instead of theirs. Now Eve can intercept and read every message between Alice and Bob and even changing them in transit.

There are however several misconceptions about public key encryption. Among them, we can find the following two: the public key encryption is more secure than symmetric key encryption and public key encryption is a general purpose technique that has made symmetric encryption obsolete. Since the security of any good encryption scheme depends on the length of the key and on the computational resources required to break the cipher, there are no major differences between the two systems. Also, due to the higher computational requirements of current public key systems when compared to the symmetric ones, conventional encryption will continue to be used in the foreseeable future.

2.6 Block Cipher

Block ciphers splits the plaintext into blocks of fixed length, and the encryption is executed on one block at a time. Typical block sizes are 64 bits or 128 bits and messages are padded (with extra bits) to fit the block size. In mathematical terms we have the plaintext $\bar{M} \to \bar{M} = m_1, m_2, ..., m_n$, where m_i represents a block and the corresponding encrypted text given by $\bar{C} = c_2, ..., c_n$, with $c_i = e_k(m_i)$ and k fixed.

The encryption function e_k requires a complex operation. To achieve the desired level of complexity all block ciphers are iterative algorithms where the data passes through multiples rounds of one or more simple operations, such as, permutations and shifts.

Examples of block ciphers includes DES and IDEA. There are several factors that influence the security of a symmetric block cipher, such as, the quality of the algorithm and the length of the key (e.g., 64 bit blocks are questionable, but 128 bit blocks are considered more than adequate). Therefore designers of cryptographic systems should consider a trade-off between efficiency and security.

Block ciphers can operate in one of several modes [MVO96], been the four most common the Electronic Codebook Book (ECB), the Cipher Block Chaining (CBC), the Cipher FeedBack (CFB) and the Output FeedBack (OFB).

2.7 Stream Ciphers

In stream ciphers the encryption of the plain text is performed at the bit or byte level, i.e., individual bits/characters of plaintext are encrypted immediately into bits/characters of ciphertext, as illustrate in figure 2.8.

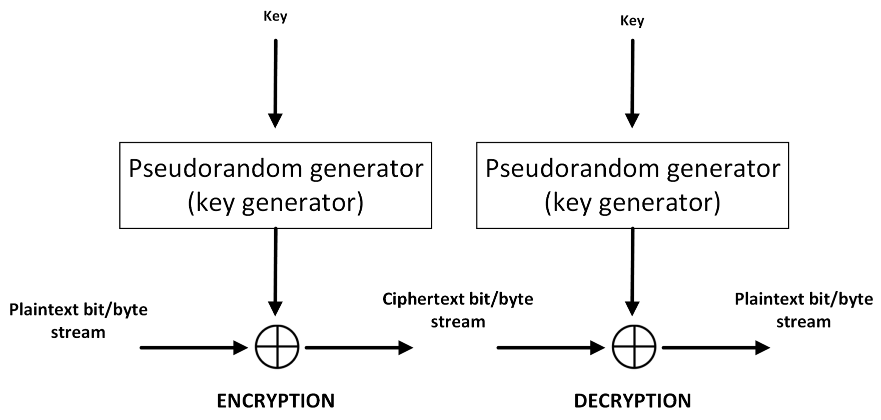

Key
Key
Pseudorandom generator (key generator)
Pseudorandom generator (key generator)
Plaintext bit/byte stream
Ciphertext bit/byte stream
Plaintext bit/byte stream
ENCRYPTION
DECRYPTION

FIGURE 2.8: Encryption and decryption operations.

Mathematically speaking, we have $c_1, c_2, ..., c_n = e_{s_1}(m_1), e_{s_2}(m_2), ..., e_{s_n}(m_n)$, where $s_1, s_2, ..., s_n$ is the key stream. The encryption function e_{s_i} is a simple operation, usually a simple XOR. The key component of these ciphers lies in the generation of the key stream.

Stream cipher are similar to one-time pad cipher [Kah96], but with a pseudo-random key instead of random key. The randomness of the stream key completely destroys any statistically properties in the message. The stream key should never be reuse, otherwise it would be possible to recover messages. Example of such ciphers includes the RC4.

2.8 Good Ciphers

The strength of any encryption method comes from the algorithm, secrecy of the key, length of the key, initialization vectors, and how all these elements work together.

A cipher needs to obscure statistical properties of original message. In 1945 Shannon suggested two characteristics for thwarting cryptanalysis based upon statistical analysis: *diffusion* and *confusion*. Diffusion dissipates the statistical structure of plaintext into long-range statistics, i.e., the cipher should be able to spread the information from the plaintext over the entire ciphertext. This means that the interceptor needs access to a lot more ciphertext to infer the algorithm. The confusion property makes the relationship between ciphertext and key as complex as possible. This cause the interceptor not to be able to predict what changing one character in the plaintext does to the ciphertext.

In 1949, Shannon [Sha49] proposed five characteristics that a good cipher should exhibit:

1. The amount of secrecy required should determine the amount of labor appropriate for encryption/decryption, i.e., the more security, the more encryption.
2. The set of keys and enciphering algorithm should be free from complexity.
3. The implementation should be as simple as possible.
4. The errors in ciphering should not propagate and cause further corruption of the message. One error should not compromise the entire encryption process.
5. The size of the ciphertext should be no larger than the size of the plaintext

Because these criteria were developed before modern computers, some points are no longer a limitation. For example, the complexity of the cipher is no longer a main issue in today's modern fast computers.

2.9 Cryptanalysis and Attacks

Cryptanalysis is the study of techniques to defeat cryptographic techniques. Its goal is to find weakness in ciphers, ciphertexts and cryptosystems, in order to recover the plaintext or any other useful information. It usually involves the detailed study on how the system works and in solving carefully constructed mathematical problems. Cryptanalysis usually excludes methods of attack that do not target weaknesses of the cryptographic system, e.g., using brute force against an algorithm without any knowledge of the algorithm.

We split attacks in two main classes: *passive* and *active*. Passive attacks includes eavesdropping and network sniffing. They are passive because the attacker does not affect the cryptographic system, making it hard to detect. Active attacks include altering messages and system files and hijacking communications. It is called active because the attacker is actively affecting some aspect of the cryptographic protocol.

We can also classify attacks in another two classes: *cryptanalytic attacks* and *implementation attacks*. While in cryptanalytic attacks the cryptanalyst attempts to attack mathematical weakness in the algorithms, in the implementation attacks class the cryptanalyst focus his attack on the specific implementation of the cipher.

Depending on what a cryptanalyst has to work with, attacks can be classified into the following models, which can refer to either of the two above classes:

- Ciphertext only attack – the cryptanalyst has knowledge of some ciphertexts $c_1 = e_k(m_1), c_2 = e_k(m_2), ...$, and his goal is to obtain $m_1, m_2,$ or the key k.
- Known plaintext attack – the cryptanalyst knows some pairs $(m_1, c_1 = e_k(m_1)), (m_2, c_2 = e_k(m_2)), ...$, and his ultimate goal is to obtain the key k.
- Chosen plaintext attack and Adaptive-chose plaintext attack – the cryptanalyst chooses the plaintext to be encrypted by the system and analyzes the resulting ciphertext. In a adaptive-chosen plaintext attack, he can choose a message to be encrypted based on previously achieved results. Hence, the cryptanalyst knows some pairs $(m_1, c_1 = e_k(m_1)), (m_2, c_2 = e_k(m_2)), ...$, of which he can choose $m_1, m_2, ...$ and his goal is to obtain the key k. Often this attack implies that the cryptanalyst will attempt to feed a planned sequence of messages, in order to reveal the most about how the data is being encrypted.
- Chosen ciphertext attack and Adaptive-chosen ciphertext attack – In the chosen and adaptive-chosen ciphertext attacks, the cryptanalyst chooses a ciphertext and obtains the corresponding plaintext, therefore he can select ciphertexts based on previous results. He has some pairs $(m_1, c_1 = e_k(m_1)), (m_2, c_2 = e_k(m_2)), ...$, of which he can choose $c_1, c_2, ...$, and once again, his goal is to obtain the key k.

There are countless attack techniques, among which we can find side channel attack, brute force attacks, implementation exploitation and replay attacks. In side channel attacks, the cryptanalyst uses incidental information that can reveal important information about the key and plaintext, such as the sounds produced by keystrokes while the plaintext is typed. In a brute force attacks, the cryptanalyst performs an exhaustive search over the keyspace in hopes to find the right one. Cleary, this type of attack is expensive and time consuming. To give an idea on the time consuming of this type of attack, in table 2.1 we can observe the required times to perform an exhaustive search over different key spaces.

Key size	Key space	Time required at 1 decryption/μs	Time required at 10^6 decryptions/μs
32 bits	2^{32}	35.8 min	2.15 ms
56 bits	2^{56}	1142 years	10.01 h
128 bits	2^{128}	5.4×10^{24} years	5.4×10^{18} years
168 bits	2^{168}	5.9×10^{36} years	5.9×10^{30} years
26 chars permutations	26!	6.4×10^{12} years	6.4×10^{6} years

Table 2.1: Required times for brute force attacks.

Perhaps from some poor judgment and testing from ciphers designers and engineers, attacks on the implementation of a system are possible, where the cryptanalyst directly attacks or uses the underlying hardware and existing faults in cryptosystems to achieve his goals. One example of such faults was the poor random generator of PlayStation 3. Replay attacks are network attacks, where the attacker copies for example a ticket and breaks the encryption or copies an authentication session. Then he resubmits the ticket or session in order to gain unauthorized access to a resource.

Chapter 3

Cellular Automata

3.1 Introduction

3.1.1 Brief History

Cellular automata theory started in the 40s with Stanislas Ulam, who was deeply interested in the concept of self-replicating automata. He considered a space composed by a two-dimensional array of cells, where each cell could be in one of two states: on or off. From an initial seed, the temporal evolution of the cells was determined by some mathematical relation between the neighbors. This simple system allowed the generation of some rather complex patterns, having some of them a behavior similar to biological processes of self-reproduction.

As Ulam developed his theory on cellular automata, von Neumann also began the development of a general theory of cellular automata, using it as a modeling tool of biological processes in an attempt to create self-replicating machines [vN51]. Following a suggestion by Ulam, he focused on discrete, two-dimensional grids. However, unlike Ulam that only used two states, von Neumann presented examples of automata with up to 29 different states. Neumann often discussed with Ulam, who gave several suggestions and contributions to Neumann's work.

Note that Ulam's contributions are rather extensive. In fact, Schrandt and Ulam [SU67] were able to create self-reproducing automata with simpler models than those used by von Neumann. In 1960, Ulam posed problems about CA in terms of infinite matrices

[Ula60] and also discussed growth models with the plane tessellated into regions which are squares or equilateral triangles [Ula62]

While the original concept of cellular automata is credited to Ulam, its development and expansion is due von Neumann. However it was not until the publication of the Conway's Game of Life by Martin Gardner [Gar70] that cellular automata became widely known.

3.1.2 Applications

The popularity of CA comes from their simplicity, their potential in executing complex computations, and the ability to model complex systems. For the last 50 years, the simple and flexible structure of CA have called the attention of researchers from an myriad number of disciplines, such as biology, chemistry, physics, astronomy and mathematics.

Because of their inherent properties, CA are quite suitable for hardware implementation. Cellular automata have been proposed for the construction of parallel computers [BMS01, MSC01], more specifically to implement some computational intensive operations such as finite field $GF(2^m)$ arithmetic [KY04, Atr65, LZ02], prime number generators [Fis65], and fast one-way hash functions [MZI98]. Since the first study of the application of CA in cryptography by Wolfram [Wol85], researchers have attempt to include CA in all cryptographic areas, from pseudo-random generators [TP01, NKC94, Wol86] to S-Boxes [SS08]. CA's potential to create high speed cryptographic applications is enormous. Other applications in computer science include pattern recognition using generalized multiple attractor CA [MGS$^+$02] and image processing [PDL$^+$79].

From the dynamics of cell growth and tissue architecture [Tvr00] to the modeling of the immune system [CS92], CA have also been used to model a myriad of different biological systems. Savill and Hogeweg [SH97] used them to achieve basic morphogenesis (the formation and differentiation of tissues and organs) and self-organization of organisms. Relations between CA and differential equations were also explored by physicists, biologists and mathematicians [Tof84, DDRR04, HR03].

Another popular application field of cellular automata is urban development. In our modern world, urban development is a constant headache for local authorities, and many predictive and explanatory models have emerged. For instance, urban traffic models [DSS09, TVB09, ES97, YCXJL12, WL05, CDL97] are useful tools to study urban traffic, where correlations between traffic patterns and control mechanisms can be carefully studied and analyzed. Using CA, the complexity of these models can be greatly reduced, without losing accuracy. Also, the dynamics of urban evolution [Bat07] can easily be achieved thanks to local actions of automata and it can go as far as enabling the creation of experimental scenarios for the development of virtual cities under realistic conditions [LP03].

3.2 Formal Definition

3.2.1 Cellular Automata

Before giving the definition of CA, we first introduce some concepts. A *cell* is a machine or object that can have one state at a time, where the states take then values from a finite alphabet, S. A *d-dimensional lattice* or *cellular space* is an ordered grid, usually denoted by $\mathbb{Z}^d$ (or $\mathcal{L}$ by some authors), with $d \in \mathbb{Z}_+$ as the dimension of the lattice and where each node represents a cell. A *neighborhood*, denoted by N, is a finite ordered subset of $\mathbb{Z}^d$.

Definition 3.1. A d-dimensional CA, is a 4-tuple (d, N, S, f), where d is the dimension, N the neighborhood, S the alphabet and f is a function $f : S^{|N|} \to S$, called the local transition function.

3.2.2 Local Function

In the local transition function, $f : S^{|N|} \to S$, $S^{|N|}$ represents the set of all possible states that the neighborhood can be in, with each of the values is a tuple of states $(s_0, s_1, ..., s_{|N|-1})$, with $s_i \in S$. For example, if $S = \{0, 1\}$ and $|N| = 3$ we can represent $S^{|N|}$ by the set $\{(0, 0, 0), (0, 0, 1), ..., (1, 1, 1)\}$.

The local transition function determines how the state of each cell is changed from an instant to the next. This decision is usually based on the cell's own current state and of its neighbors. Moreover, local functions can be either deterministic or probabilistic.

Every CA evolves in discrete time steps and all cells are updated simultaneously. The *global state* or *global configuration*, ϕ, is the state of each cell in the lattice at t:

$$\phi : \mathbb{Z}^d \to S$$

The state of an arbitrary cell at $x \in \mathbb{Z}^d$ can now be specified by $\phi(x)$. Let $N = \{z_0, z_1, ..., z_{n-1}\}$ be the neighborhood size n of an arbitrary CA and let ϕ^{t+1} be the global state of the cells at $t+1$. The state of an arbitrary cell x at the instant $t+1$ is given by:

$$\phi^{t+1}(x) = f[\phi^t(x+z_0), \phi^t(x+z_1), ...\phi^t(x+z_{n-1})], \ \ \forall x \in \mathbb{Z}^d \tag{3.1}$$

The temporal evolution of a cell is given by the recursive application of the local function to the cell:

$$\phi^t(x) \to \phi^{t+1}(x) \to \phi^{t+2}(x) \to \cdots$$

3.2.3 Elementary Cellular Automata

A neighborhood of a cell x with radius r is the set of the r cells both to the left and right of x, including cell x. Its size is given by $|N| = 2r+1$.

If $r = 1$ and $|S| = 2$, we have a three-cell neighborhood $N = \{-1, 0, 1\}$ with two possible states for each cell. Therefore we have $S^{|N|} = 2^3 = 8$ possible state configurations of the neighborhood, which can be represented as depicted in figure 3.1, where the black and white squares represent the states '1' and '0' respectively.

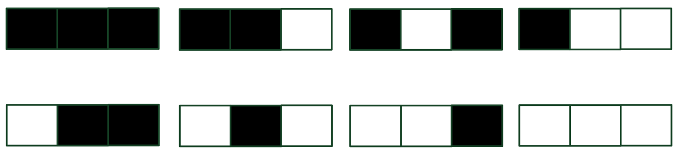


FIGURE 3.1: All possible states configurations of the neighborhoods in any ECA.

Definition 3.2. An elementary cellular automata (ECA) is any cellular automata with a radius 1 neighborhood and a binary state set $S = \{0, 1\}$.

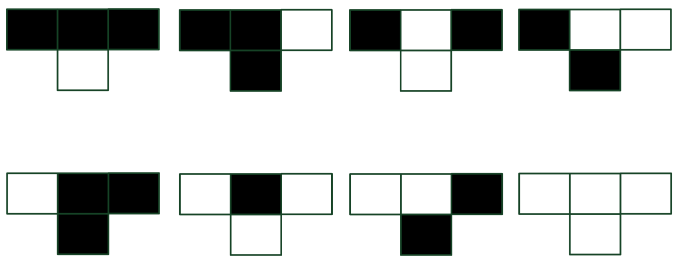
In the case of ECA, equation (3.1) becomes:

$$\phi^{t+1}(x) = f[\phi^t(x-1), \phi^t(x), \phi^t(x+1)]$$

We can see that we can have up to $2^{2^3} = 256$ possible elementary cellular automata. Consider the following simple example, where the transition function depends only on the left and right neighbors of x: $\phi^{t+1}(x) = \phi^t(x-1) + \phi^t(x+1) \pmod 2$. We can represent this function either by a table that maps the the state of the neighborhood to the next state for the center cell:

$\phi^t(x-1)$	$\phi^t(x)$	$\phi^t(x+1)$	$\phi^{t+1}(x)$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

or through the following graphic representation:



where the top row represents the state of a cell and its neighbors at instant t, while the bottom row represents the state of a cell at $t+1$.

To encode this table, Wolfram suggested reading the last column as a 8-bit binary number. Since 01011010 in binary is 90 in decimal, Wolfram named this CA rule 90. Hence, the 256 ECAs have come to be known as Wolfram rules and the associated numbers as Wolfram numbers.

3.2.4 Global Function

Lets represent c as the current configuration of the automata with $c \in \mathbb{Z}^d$. The CA's next configuration is given by $\Phi(c)$, where $\Phi : \Sigma^{\mathbb{Z}^d} \rightarrow \Sigma^{\mathbb{Z}^d}$. We call Φ *global map* or *global function.* The CA's temporal evolution is then:

$$c \to \Phi(c) \to \Phi^2(c) \to \cdots$$

We name the sequence $c, \Phi(c), \Phi^2(c), ...$ the *orbit* of c. In Figure 3.2 we can observe all possible types of periodic evolutions of any CA:

- *fixed point*: $\Phi(c) = c$
- *periodic*: $\Phi^t(c) = c$ for some $t \in \mathbb{Z}_+$
- *eventually fixed*: $\exists t \in \mathbb{N} : \Phi^{t+1}(c) = \Phi^t(c)$
- *eventually periodic*: $\exists t \in \mathbb{N}, k \in \mathbb{Z}_+ : \Phi^{t+k}(c) = \Phi^t(c)$

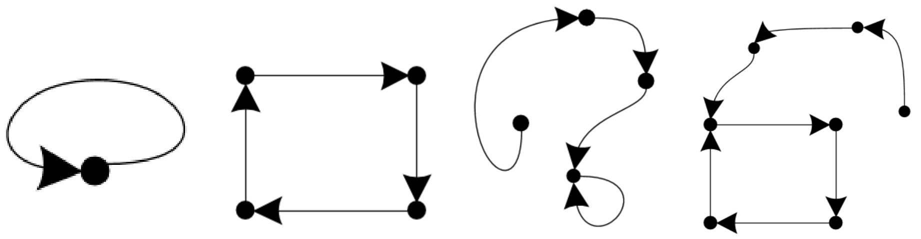

(a) Fixed point. (b) Periodic. (c) Eventually fixed. (d) Eventually periodic.

FIGURE 3.2: Orbits

Besides periodic evolutions, cellular automata can still present a non-periodic or a incompressible non-periodic evolution.

It is common to use graphs, called *space-time diagrams*, for the representation of the temporal configuration evolution of one-dimensional CA. In these diagrams, configurations are depicted horizontally and their evolution are drawn downwards as we can see in figure 3.3. Naturally, these diagrams are only possible for $d \leqslant 2$.

3.2.5 Neighborhoods

The most usual neighborhood for one-dimensional CA is referred to as the *first neighbors neighborhood*, and it consists simply of the central cell of the neighborhood and its right and left neighbors: $\{-1, 0, 1\}$. Extending this notation for a radius r, the neighborhood set is given by $\{-r, -r+1, ..., 0, ..., r-1, r\}$, as mentioned in section 3.2.3.

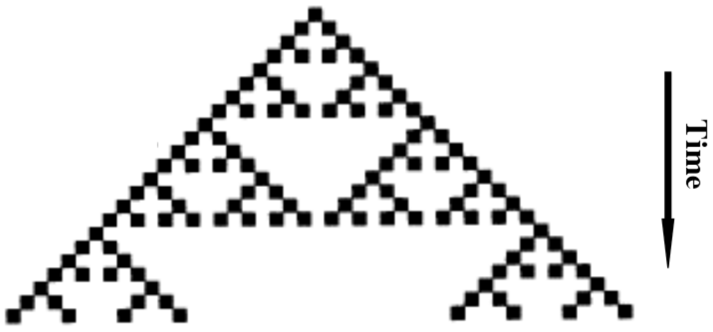
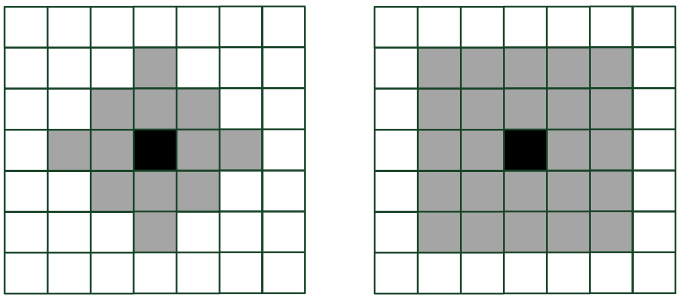


FIGURE 3.3: Example of a space-time diagram.

For two-dimensional CA, the von Neumann and Moore neighborhoods are the two most common neighborhood shapes (see figure 3.4).



(a) von Neumann neighborhood. (b) Moore neighborhood.

FIGURE 3.4: The two most common neighborhoods for two-dimensional CA with radius 2.

For $d \geqslant 1$ and radius r these neighborhoods are defined by:

$$N_v(y) = \{x \mid x \in \mathbb{Z}^d, d_1(x, y) \leqslant r\}$$

for the von Neumann, containing $(2r + 1)^d$ elements and

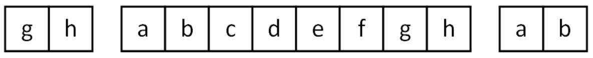
$$N_M(y) = \{x \mid x \in \mathbb{Z}^d, d_\infty(x, y) \leqslant r\}$$

for Moore's neighborhood, where d_1 and d_∞ are metrics associated with the Manhattan norm $\parallel y \parallel_1$ and the maximum norm $\parallel y \parallel_\infty$ respectively. For a n dimensional vector y, these norms are given by $\parallel y \parallel_1 = \sum_{i=1}^{n} |y^i|$ and $\parallel y \parallel_\infty = max\{|y_i| \; |i \in \{1, 2, ..., n\}\}$. Note that in one dimensional CA, the von Neumann and Moore neighborhoods are identical.

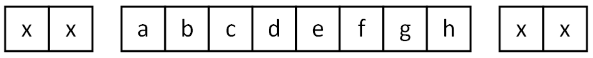
Another usual neighborhood is the second-order neighborhood or the neighborhood of the neighborhood, which will be used and explained later, in chapter 5 pp. 70.

3.2.6 Boundaries of Finite Latices

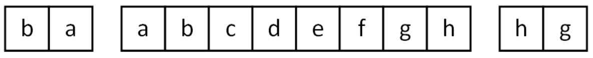
Naturally infinite CA have no boundaries descriptions associated with them. When we are dealing with CA with a finite lattice, the neighborhood used by the local transition function trespass the lattice boundaries. There are two main solutions for this problem: we can either connect the boundaries cells (creating a Möbius strip for one-dimensional CA or torus for two-dimensional CA), or we can define boundaries values for the values of the trespassed parts of the neighborhood. We name the boundaries of the first solution as *periodic* or *cyclic* boundaries. As for the second, there are many possible ways we can set the boundary values, been the two most common the *fixed* (a constant value on the boundary) and the *reflective* (reflection of the lattice at the boundary) boundaries. In figure 3.5 we can observe examples, where the 8 central cells represent the finite lattice and the four adjacent cells represent the trespassed parts of the neighborhood.



(a) Periodic or cyclic.



(b) Fixed.



(c) Reflective.

FIGURE 3.5: Three different types of boundaries.

3.2.7 Classification Systems

Wolfram proposed a classification scheme which divided cellular automata rules into four classes, according to the result of the evolution of the system from an initial state [Wol84]. These Wolfram classes are the following:

- W1: automata of this class die completely after just a few or no steps of evolution. The system evolution leads to uniform fixed point configurations after a finite number of steps, after which no changes happen. Approximately 86% of the 256 ECA belong to this class. Examples of such rules are 119 and 222.

- W2: evolution leads to a periodically repeating configuration, i.e., after some iterations, configurations starts to repeat themselves. Approximately 9% of the 256 ECA belong to this class. Rules 4 and 126 as examples for this cases.
- W3: evolution lead to a chaotic behavior. Practically all initial conditions leads to patterns that never repeat. Small changes usually spread at a uniform rate and eventually affecting all parts of the system. Approximately 4% of the 256 ECA belong to this class. Rules 30 and 90 are examples.
- W4: evolution leads to localized and propagated structures with complex patterns that can last for arbitrary lengths of time. Only 4 of the 256 elementary cellular automata belong to this class. Examples include rules 54 and 110.

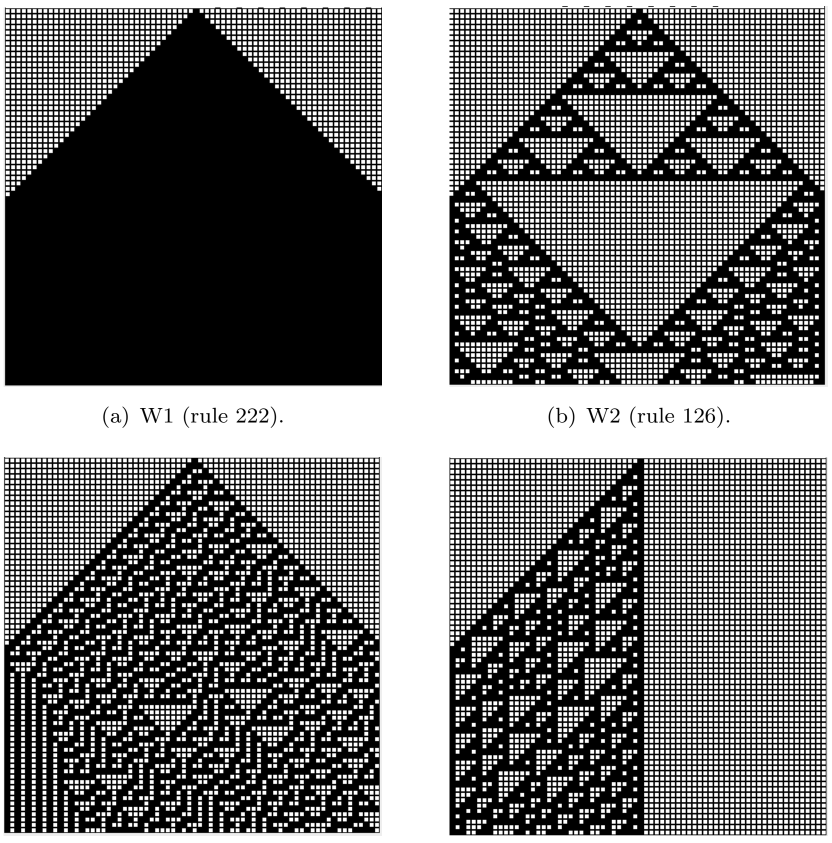

(a) W1 (rule 222). (b) W2 (rule 126). (c) W3 (rule 30). (d) W4 (rule 110).

Figure 3.6: Examples of space-time diagrams of the four Wolfram classes.

However this classification is not unique. Other authors such as Li-Packard [LPL90, LP90], Kurka [K97, K03] and Culik & Yu [IY88], have also provided alternative classification schemes.

3.3 Reversibility

Reversible cellular automata (RCA) or invertible cellular automata (ICA), are cellular automata capable of recovering the initial state of the automata. In figure 3.7 we can observe two simple examples of reversible rules.

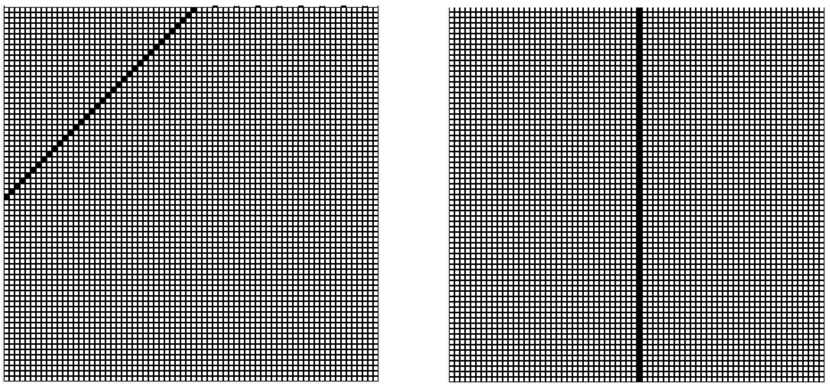

(a) Rule 170. (b) Rule 204.

Figure 3.7: Examples of reversible cellular automata.

Reversible cellular automata are CAs for which the global map is invertible. To be reversible, its global function Φ must be a bijective function $\Phi : \Sigma^{\mathbb{Z}^d} \to \Sigma^{\mathbb{Z}^d}$. We can say that a cellular automaton A is reversible if there exists a cellular automaton B that reverts it, that is such that:

$$\Psi \circ \Phi = Identity\ function$$

where Ψ and Φ are the global functions of A and B respectively.

We can then state the following proposition:

Proposition 3.3. *[Hed69, Ric72] A cellular automaton is reversible if and only if it is injective.*

The problem of deciding if a one-dimensional automaton is invertible, is a decidable problem, even for infinite, one-dimensional CA, as demonstrated by Amoroso and Patt [AP72].

Proposition 3.4. *[AP72] It is decidable given a one-dimensional cellular automaton to decide whether it is reversible.*

However the reversibility and surjectivity in dimensions higher than one are undecidable.

Proposition 3.5. *[Kar90, Kar94] It is undecidable given a two-dimensional cellular automaton to decide whether it is reversible.*

Because invertible cellular automata constitutes a very small subset of CA [Sea71] and because reversibility can be an undecidable question, we can provide convenient ways to construct reversible cellular automata, providing that we narrow even more the invertible CA set, by ensuring that during the construction of an automaton the backward rule is constructed at the same time as the forward rule. Following we present two simple techniques that can be used to create simple RCA. For other techniques, we refer to the work of Toffoli and Margolus [TM90].

Trivial constructions. They can be build with neighborhood consisting of only one cell [AP72] and with invertible local functions, for instance, permutations of S:

$$\phi : S \to S$$

The simplest case is created by using a local update rule that only copies one of its neighbors. If it copies from the left or from right, we have a simple shift, which is a trivial invertible operation.

Second-order CA. Following the work of Fredkin, Vichniac and others authors [Vic84, Wol02] suggested the use of special rules to achieve invertibility. In these RCAs, the new state of a cell is determined not only by the cell itself and its neighbors one instant back but, also by the cell's state two instants back (see Figures 3.8 and 3.9). However the reversible rules are not the reverse of the original ones. In figure 3.10 we can see in more detail the differences between a normal rule and a modified one.

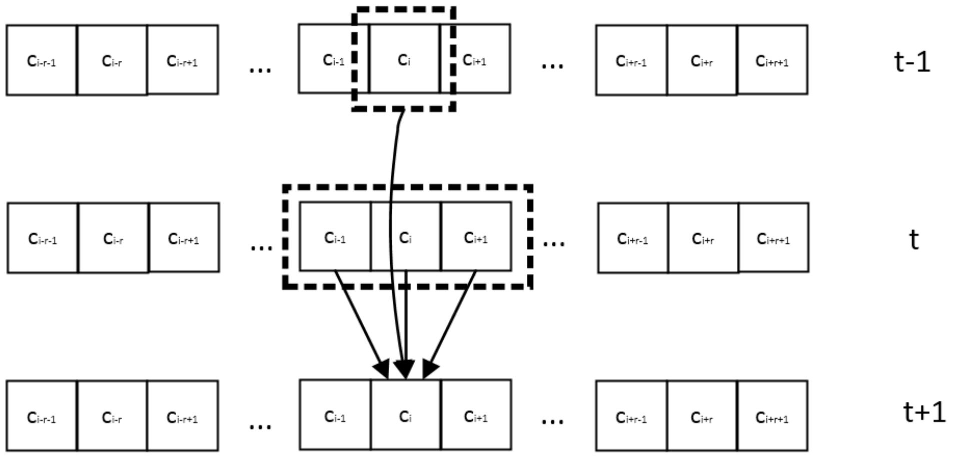


Figure 3.8: Simple CA explicitly set to be reversible

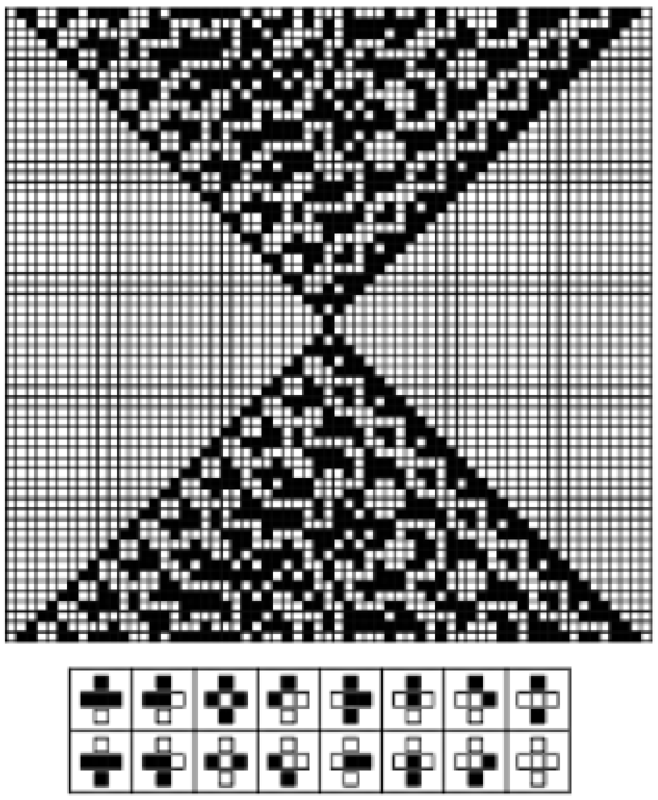
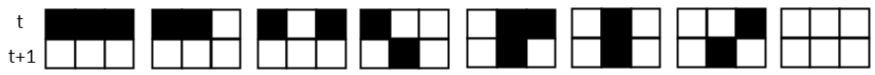


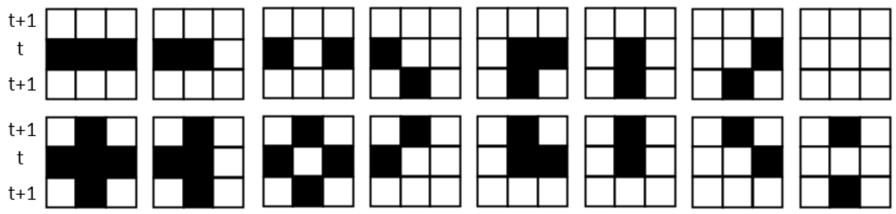
Figure 3.9: Simple CA explicitly designed to be reversible.

The configuration c^{t+1} is determined by the linear combination of the two previous configurations c^t and c^{t-1}:

$$c^{t+1} = \tau(c^t) - c^{t-1}$$



(a) Rule 30



(b) Rule 30R

FIGURE 3.10: Rule 30 and reversible rule 30R

where $c \in \Sigma^{\mathbb{Z}^d}$ and τ is the dynamic law of the system, which in our case is the global function Φ . By simply rearranging the above formula, we obtain the reverse:

$$c^{t-1} = \tau(c^t) - c^{t+1}$$

In 1977, Toffoli [Tof77] demonstrated that a two-dimensional cellular automaton is Turing complete. However, he left the Turing completeness of one-dimensional CA as an open problem. This problem was tacked and solved in 89 by Morita and Harao [MH89]. They proved that one-dimensional RCA are Turing complete, by embedding a reversing Turing machine in a one-dimensional RCA, which is in fact a variant type of CA, called one-dimensional partitioned cellular automata. With this variant, they were able to design a globally reversible partitioned cellular automata, capable of simulating a reversible Turing machine. Since a reversible partitioned cellular automata can be simulated by a reversible one-dimensional CA, the Turing completeness was proved.

Chapter 4

Brief Overview of Research on CA-based Cryptography

Ever since the introduction of cellular automata in the cryptographic field, and also because cellular automata have the same expressiveness than Turing machines, cellular automata have found their way into a plethora of encryption schemes, either as their main basis or simply as one of their components. We believe that the presentation of an overview of the most relevant CA-based cryptographic systems and primitives would be useful, both in the contextualization and in laying down the extension of the applicability and usefulness of cellular automata in the development of cryptographic systems.

4.1 Pseudo Random Number Generators

The generation of random numbers is a crucial task in cryptography. They are used not only in the generation of cryptographic keys but also in other crucial parts of cryptographic algorithms and protocols, for examples, the generation of initialization vectors.

A pseudo-random number generator (PRNG) is a deterministic algorithm that produces numbers whose distribution is indistinguishable from uniform. This cryptographic mechanism uses an initial seed of randomness (e.g. from some input streams, such as the timing of keystrokes and timing of disk I/O response times [JSHJ98]), and attempts to generate outputs that in practice should be indistinguishable from truly random sources

[SV86, Gut98]. A good cryptographic PRNG should produce a sequence of repeatable, but high-quality, random numbers. For an excellent overview on PRGs see Knuth [Knu97] and Andreas Röck Master thesis [Roc05].

Be as it may be, despite the numerous advices given by various researchers, these are often ignored, resulting in insecure generators, which in turn produces encryption keys that are relatively easier to guess, crippling the underlying cryptosystems where they are used in. Weakly designed PRNGs can easily destroy the security of any system even if strong cryptographic primitives are used.

Presently, linear-feedback shift registers (LFSR) and linear congruential generators are the most popular technique in the design of PRG, because of their compact and simple design. Nonetheless, the LFSR generator is not sufficiently good for cryptographic applications. [Roc05].

The application of cellular automata in cryptographic PRNGs started in the 80's with Wolfram. In 1985 he suggested a cryptosystem [Wol85] using a PRNG based on one-dimensional CA with periodic boundaries and using rule 30 [Wol86]. The generated random stream is used as the key stream with which the plaintext could be XORed. The random number is obtained from the center cell of the lattice. A stream of random numbers is then obtained from the continuous evolution of the CA. The choice of the center cell as the source of the random bits was based on the fact that this cell presented no significant statistical regularities. During his research, Wolfram noted that the periodicity of the configurations of the automata (the minimum number of evolutions required for the automata to repeat a configuration) depends not only on the size of the cellular space but also on the initial seed. His PRNGs was able to pass a suite of seven statistical tests, even surpassing classical PRNGs based on LFSR of the same size, but not as good as linear congruential generators. According to his estimations, if the cellular space of the automata is larger than 53 cells, its maximum period should be approximately $2^{0.61N}$, where N represents the size of the cellular space. Wolfram noted that the statistical properties of the random stream is better if its size is much shorter than the period of the generator.

In 1991, Meier et at. [MS91] broke Wolfram's cryptosystem with chosen plaintext attacks where they demonstrated that the sequence of configurations of the system is not hard to recover. Wolfram [Wol86] also claimed that the problem of recovering the seed of its PRNG was an NP-complete problem. However, Koc and Apohan [KA97] proved that this was not the case and they presented a computationally feasible algorithm capable of inverting rule 30 in $O(n)$ for some seeds and $O(2^{n/2})$ in the worst case.

To increase the period and the statistical properties of CA-based PRNGs, a new type of CA emerged: the hybrid or non-uniform CA, where instead of a single rule, the automata uses multiple rules. To practically achieve this flexibility, a new type of CA structure emerged: the programmable CA (PCA) [NKC94]. In figure 4.1 we can observe a simplistic representation of the structure of a PCA, with which we can form different additive CAs (one having a combination of XOR and XNOR rules), by simply manipulating the controlling signals, which controls which neighboring cells are combined. This flexible

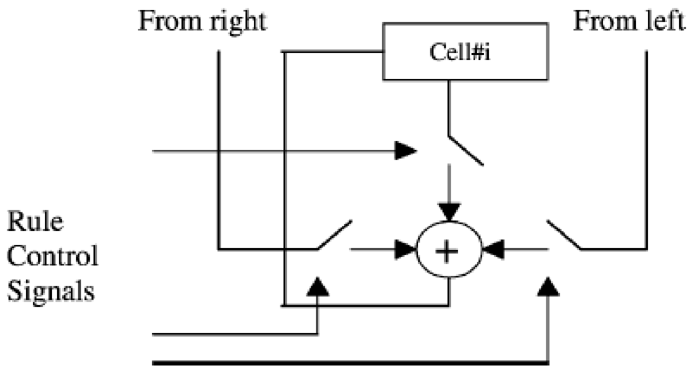


Figure 4.1: Structure of an additive PCA.

structure enables the use of nonlinear enciphering functions, capable of generating a large number of multiple keys.

Hortensius et al. [HMC89, HMP+89] proposed a non-uniform cellular automata, consisting of a pair of rules, $(90, 150)$ or $(30, 45)$, with the intent of generating test vectors in VLSI manufacturing. Statistical tests showed that the performance of these generators exceeded Wolfram's. They also noted that non-uniform CA with null boundary had on average longer period. Hortensius at al. also demonstrated that their CA had a better statistical performance than LFSRs.

Nandi, Chaudhuri and Kar [NKC94] proposed a block and stream cipher but, they focus their attention on non-uniform CA with null boundaries, operating exclusively with linear rules (rules that exclusively employs XOR logic operations), namely rules

51, 153 and 195 for block ciphers and rules 90 and 150 for stream ciphers. However their block scheme was broken by Murphy et al. [BBM97]. Another successful attack on Nandi et al. cryptosystem was made by Mihaljevic [Mih], but this time using only ciphtertext. In the attempt of eliminating the weakness present in [NKC94], Sen et al. [SSC^{+}02] combined affine and non-affine transformations in order to achieve non-linearity. However, Bao [Bao03] was able to show that the cryptosystem based on this combination could also be easily broken with a chosen-plaintext attack and with low computational cost.

Rubio et al. [REW^{+}04] tested 28 linear hybrid CAs, formed by the combination of two linear ECA, for example (60, 90), where the corresponding rule to apply to any cell is dependent on its current state. For their tests, they considered both periodic boundaries and null boundaries and also several random initial configurations. They tested their pseudorandom properties through a battery of cryptographic statistical tests, such as frequency, serial and poker tests. Because Meier & Stafelbach [MS91] demonstrated the feasibility of attacking PRGs generators with a size lesser than 500 cells, with attacks based on the algebraic properties of CAs, Rubio et al. considered hybrid CAs with 512 cells. From the 28 rules only 10 hybrid CAs with periodic boundaries and 3 with null boundaries passed the tests.

More information on non-uniform CAs with null boundaries and its relation with length cycle and their application on encryption schemes can be consulted in the works of Anghelescu et al. [Ang11, AIS07, AIB09].

From the results of the studies mentioned above, we see that for a PRNG to provide both a good randomness quality and a satisfactory period length, the choice of the transition rules is of the utmost importance. Because of the sheer number of rules and CA configurations, many researchers turned into methods of automatically select the best set of rules for a given CA. Among those methods we find the use of genetic algorithms. Tomassini et al. [ST96] demonstrated that the use of genetic algorithms can give birth to good PRNGs. These algorithms usually employ entropy as the fitness function to trigger mutations. However, high entropy is a necessary but not sufficient condition for a good quality PRNG, and therefore statistical tests should also be applied after the evolution and selection of the PRNG. Tomassini showed that a good PRNG can be obtained by randomly mixing the rules 90, 105, 150 and 165. Tomassini and other researchers continued their work on evolutionary rules selection, giving rise to a plenitude of papers [TP01, GZ03, TSZP99, SBZ04, SSB06, TSP00].

4.2 Block Encryption With Second-Order Cellular Automata

In a second-oder CA, the state of each cell at instant $t+1$ depends on the neighborhood state configuration at t and the state of the cell at $t-1$:

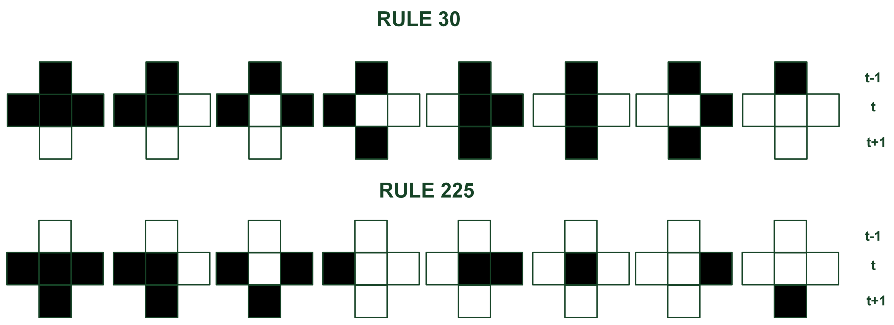
$$\phi^{t+1}(x) = f(\phi^t(x-r), ..., \phi^t(x), ..., \phi^t(x+r), \phi^{t-1}(x-r), ..., \phi^{t-1}(x), ..., \phi^{t-1}(x+r))$$

where $\phi^t(x)$ is the state of cell x at instant t, f the local transition function and r the neighborhood radius.

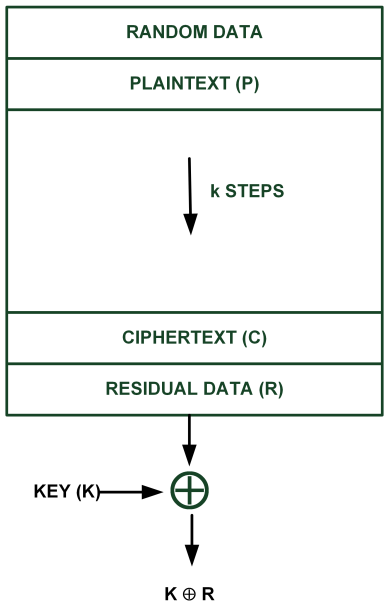
With this extra dependency and associating two elementary cellular automata we are able to build reversible rules. The two ECA rules R_1 and R_2, must be related to each other by the condition:

$$R_2 = 2^d - R_1 - 1$$

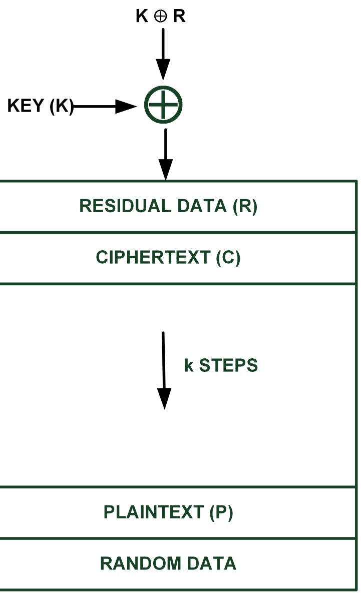
where $d = 2^{2r+1}$. The first rule defines the state transition when the cell at instant $t-1$ was in state '1', and the second one when the cell was in the state '0'. For example, selecting $R_1 = 30$ and $r = 1$ we have $R_2 = 2^{2^3} - 30 - 1 = 225$, then we can represent our second order rule by:



Next we present how these second order reversible rules can be used in a cryptosystem. After the set of the first two configurations (for example, with random data and plain-text), the encryption process is done by evolving the CA by a pre-defined number of k steps, as illustrated in figure 4.2. When encryption is complete, the last configuration is considered residual or junk data, which can be XORed with the private key. The last but one configuration is our ciphertext. Since every step requires the two previous configurations, these two final configurations (the residual and the ciphertext) should be saved, since they are going to be required for the decryption process as the two initial configurations of the automata. The CA then must iterate the same number of steps as in the encryption process, k steps, and the resulting next to last configuration is then the recovered plaintext.



(a) Encryption.



(b) Decryption.

FIGURE 4.2: Simplified encryption and decryption process.

Since is quite obvious that a cryptosystem based merely on a single second order reversible rule and on a few iterations is not safe, Bouvry et al. [SB04b, SB04a] suggested a more secure cryptosystem. In their system, the two initial configurations of the CA are populated by random data and by the plaintext as illustrated. To increase security, they used multiple rounds of data manipulation and transformations. One round of this system can be seen in figure 4.3 (source: [SB04a]). The system is composed by four one-dimensional CA: CA_L, CA_R, CA_C and CA_S. All CA are radius-2 except for CA_C which is radius-3. Each plaintext block is 64 bits long and the key is 224 bits, which specifies the four RCA rules to be used and number of evolutionary steps. CA_L, CA_R and CA_S are composed of 32 cells and CA_C by 64 cells. Encryption of each plaintext block is composed of a predefined number of rounds, where the data suffers splittings, shiftings, recombinations and configuration evolution in each round by the respective

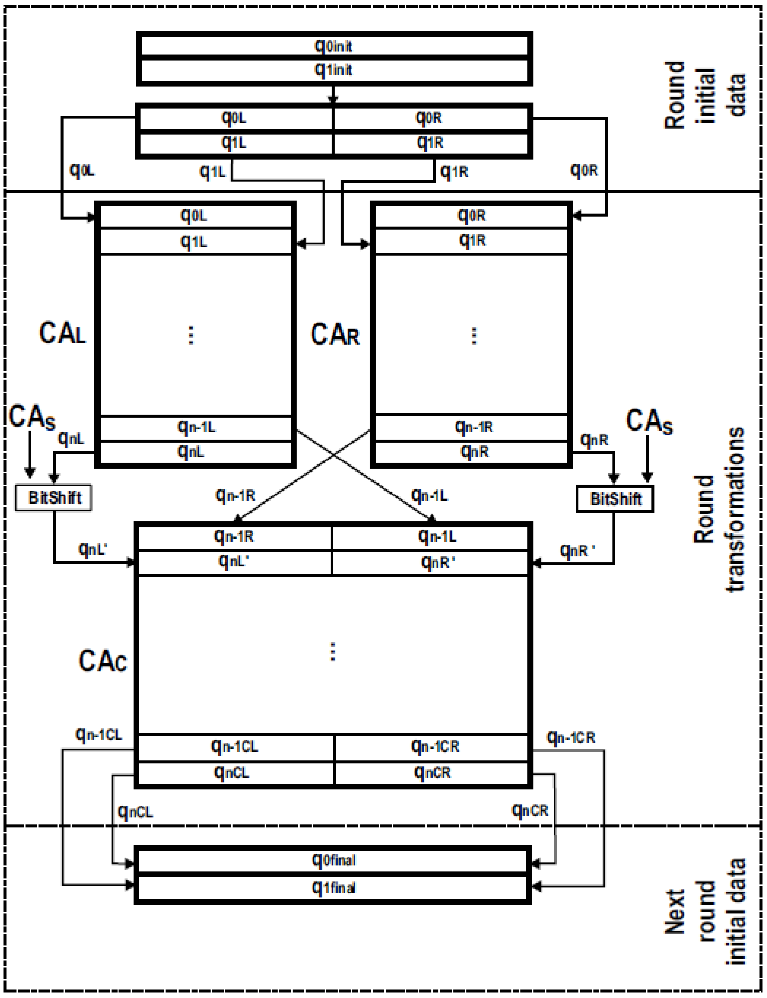


FIGURE 4.3: Single round of Bouvry' system.

CA: CA_L and CA_R applies the MCA rule to the left and right part of the data respectively k_1 times, CA_C applies the MCA rule to all bits of the data k_2 times and CA_S generates the shift to apply. The initial data is randomly generated and if CBC mode is used, the final data will be used as the initial data for the subsequent encryption blocks. In addition of saving the generated ciphertext, the final data and the final two configurations of CA_S must also be saved. The decryption process is done by reversing the order, including the shifts. The number of evolutionary steps required by each CA to achieve an avalanche effect and optimal effect were obtained through experimental evaluation.

Brute force attacks on the key is out of question, as a result of the key space size: 2^{224}. It is also computably infeasible to discover the initial configurations of the automaton.

Although finding the configurations of CA_C and CA_S (2^{64} and 2^{32} respectively) is computationally feasible, the attacker would still have to test all 2^{32} possible configurations of both CA_L and CA_R, if he uses a brute force approach. To prevent any possibility of brute force attacks, the block size can be increased, for instance, from 64 to 128 bits.

4.3 Secret Sharing Schemes

Usually, in secret sharing schemes, a secret is shared between different parties so that only a selected subset of those parties can recover the secret. Applications of such schemes includes the Byzantine agreement [Rab, Wan11], multi-party computations [CDM], access control [NW98], threshold cryptography [DF92], attribute-based encryption [GPSW] and e-voting [Sch99].

In a (k, n) *threshold scheme*, the secret S, is decomposed by a trusted authority, called a dealer, into n pieces and each piece is distributed among n parties in such a way that the only way to reconstruct the secret is if k or more parties get together their pieces. The quantity of information of each party in this scheme is uniform, i.e., the secret is equally shared among members in a group and where every member has an equal amount of partial information. A series of (k, n) threshold schemes were simultaneously introduced during the 70's by Shamir [Sha79] and Blakley [Bla79].

Since all existing works tackle this problem in a similar way, we first define a few common concepts. As we have seen in the previous section 4.1, pp. 31, a *linear cellular automata* (LCA) is a CA with rules that exclusively employs XOR logic operations, i.e., the next-state is determinate by a function that employs only XOR logic operations. Hence, the state of each cell is updated according to equation (3.1), which now assumes the following form for one-dimensional LCA with a neighborhood N radius-r:

$$\phi^{t+1}(i) = \sum_{j=-r}^{r} \alpha_j \phi^t(i+j) \pmod 2, \quad \forall i \in \mathbb{Z},\ 0 \leqslant i \leqslant |N| - 1,$$

where $\phi^t(i)$ denotes the state of cell i at instant t and $\forall j : \alpha_j \in \mathbb{Z}_2$. Because there are $2r + 1$ neighboring cell (including the origin) for each i, there are 2^{2r+1} LCAs and each one can be specified by an integer w called rule number, which is defined by:

$$w = \sum_{j=-r}^{r} \alpha_j 2^{r+j}, \quad 0 \leqslant w \leqslant 2^{2r+1} - 1$$

However, as we have seen in section 3.3, we can also make the state of a cell at $t+1$ depend not only on its state and of its neighbors at t, but also on the state of those same cells or of some other groups of cells at $t-1, t-2, ..., t-k$. This CA is then called k-th order cellular automata or *memory cellular automata* (MCA). A k-th order *linear memory cellular automata* (LMCA) the state transition of each cell is given by:

$$\phi^{t+1}(i) = f_1(V_i^t) + f_2(V_i^{t-1}) + \cdots + f_k(V_i^{t-k+1}) \pmod 2, \quad \forall i \in \mathbb{Z},\ 0 \leqslant i \leqslant |N| - 1, \tag{4.1}$$

where f_i represents the local transition function of a particular LCA with radius r for $1 \leqslant i \leqslant k$ and $V_i^t \subset (\mathbb{Z}_2)^{2r+1}$ denotes the states of the neighbors of cell i. Because of its dependency on previous configurations, to start its evolution, the k-th order LMCA, requires then k initial configurations $(\Phi^0(c), \cdots, \Phi^{k-1}(c))$, where $\Phi^0(c) = c$ represents the initial configuration of the CA.

The application of automata in secret sharing schemes, was introduced in 2003 by Encinas et al. [MEdR03]. They proposed a (k, n)-threshold scheme based on two-dimensional LMCA with periodic boundaries, Moore neighborhood with radius 1 and an alphabet S. They targeted specifically images of size $r \times s$ bits as the secret, and therefore the number of cells of the CA is $r \times s$. Since we are dealing now with two-dimensional CA, the above equations should be modified in order to consider two-dimensional cellular automata with Moore neighborhood. The state of cell (i, j) is now updated according to:

$$\phi^{t+1}(i,j) = \sum_{\beta,\gamma \in \{-1,0,1\}} \alpha_{\beta,\gamma} \phi^t(i+\beta, j+\gamma) \pmod c,$$

with $0 \leqslant i \leqslant r-1, 0 \leqslant j \leqslant s-1$, $\alpha_{\beta,\gamma} \in \mathbb{Z}_2$ and $c = 2^b$ denotes the number of colors of the image (b&w image, then $b = 1$; for gray-level images; $b = 8$, if color image, $b = 24$). Because each cell has now 9 neighboring cells, there are $2^9 = 512$ possible two-dimensional LCAs and each one is specified by the rule number w, which is given by:

$$w = \alpha_{-1,-1}2^8 + \alpha_{-1,0}2^7 + \alpha_{-1,1}2^6 + \alpha_{0,-1}2^5 + \alpha_{0,0}2^4 + \alpha_{0,1}2^3 + \alpha_{1,-1}2^2 + \alpha_{1,0}2^1 + \alpha_{1,1}2^0,$$

with $0 \leqslant w \leqslant 511$.

Let $V_{ij}^t \subset (\mathbb{Z}_2)^9$ denote the state of the neighboring cells of cell i. Equation (4.1) now becomes:

$$\phi^{t+1}(i,j) = \sum_{m=0}^{k-1} f_{m+1}(V_{ij}^{t-m}) \pmod{c} \tag{4.2}$$

with $0 \leqslant i \leqslant r-1, 0 \leqslant i \leqslant s-1$ and as before f_l, with $1 \leqslant j \leqslant k$, denotes the local update function of some k two-dimensional LCAs. Equation (4.2) is reversible if the following condition holds:

$$f_k(V_{ij}^{t-k+1}) = \phi^{t-k+1}(i,j)$$

Then the inverse is another another LMCA with local transition function given by:

$$\phi^{t+1}(i,j) = -\sum_{m=0}^{k-2} f_{k-m-1}(V_{ij}^{t-m}) + \phi^{t-k+1}(i,j) \pmod{c} \tag{4.3}$$

with $0 \leqslant i \leqslant r-1, 0 \leqslant i \leqslant s-1$.

In this scheme, both the shared and the recovered image have the same size as the original. Also, $\Phi^0(c), \Phi^1(c), ..., \Phi^{k-1}(c)$ are populated both with data from the image and with random matrices. The configuration $\Phi^0(c)$ is populated with the secret image, while the others $k-1$ configurations are randomly generated. Each participant or group receives one subsequent CA configuration randomly selected, i.e., if there are n participants, then each participant will receive one of the n final configurations of the CA. Summarizing the main steps of the system:

1. Setup phase:
   - The dealer generates a sequence of $k-1$ random integers where each number corresponds to a respective update function, $\{w_1, \cdots, w_{k-1}\}$, where $0 \leqslant w_l \leqslant 511$ and $1 \leqslant l \leqslant k-1$.
   - The dealer constructs equation (4.3) with $f_{w_l} : (\mathbb{Z}_c)^9 \to \mathbb{Z}_c$.

- The dealer populates the configurations $\{\Phi^0(c), \Phi^1(c), ..., \Phi^{k-1}(c)\}$, where the configuration $\Phi^0(c)$ stays with the bit representation of the secret image while the remaining $k-1$ configurations are randomly populated.

2. Sharing phase:

   - The dealer evolve the two-dimensional CA $m+n-1$ steps, by choosing m and ensuring that $m \geqslant k$ (to avoid overlaps between initial configurations and shares);
   - The dealer then distributes to each of the n participant, one of the last n consecutive final configurations of the CA, $\{\Phi^m(c), \Phi^{m+1}(c), ..., \Phi^{m+n-1}(c)\}$ and the set rules generated in the previous phase, so that the reverse of the linear equation (4.3) can be constructed.

3. Recovery phase:

   - Obtain a set of consecutive k shares of the form $\{\Phi^{m+\alpha}(c), \Phi^{m+1+\alpha}(c), ...,$ $\Phi^{n+m-1+\alpha}(c)\}$ with $0 \geqslant \alpha \geqslant n-k$;
   - By inverting the configuration order and iterating $m+n-1$ times the inverse of equation (4.3), we are able to recover the initial configurations of CA including the secret image ($\Phi^0(c)$).

According to the authors, the security of the system lies on the fact that if any of the k configurations is missing, we would need to solve an underdetermined system in order to recover the secret image.

In 2008, Encinas et al. [AEdR08] extended their previous work [MEdR03] to a (n, n)-threshold multi-secret sharing secret scheme, such that that each of the n participants shares a secret color image out of n. Unlike in their previous work, where the initial configurations were populated with both the secret image and random data, in this scheme, all secret images populate the n initial configurations of the CA. During the setup phase, each image can be padded, if necessary, in order for all images to have the same size. As before, the final size of the automata depends on the number of bits of the images. The security of this system was demonstrated to be utterly capable of resisting the most important statistical attacks. As before, the absence of a single part of the secret leads to a underdetermined system of linear equations. The scheme presented excellent values for both the confusion and diffusion properties and also presented good statistical results.

Despite the good security results of the above schemes, Jafarpour et al. [JNKS07] proved that they are vulnerable against dishonest participants collusion by either cheating with bogus shares or with bogus transition rules. These cheats allows L cheaters, with $L \leq k$, to obtain the correct secret by computing the *cheating value* for every cell at each step, while the others participants receive corrupted shares without knowing that cheating is done in the background. To reduce this vulnerability, authentication methods were introduced, such as the ones by Eslami et al. [EA10, ERA10] and Sujata et al. [SS13].

4.4 S-Boxes

Substitution Boxes or S-Boxes are fundamental pieces in symmetric cryptosystems, such as, DES [NBoS77], Blowfish [Sch94] and AES [PT01]. In general, an S-Box takes m input bits and transforms them into n output bits. Basically there are n component functions each being a map from m bits to 1 bit, i.e., each component function is a Boolean function in m Boolean variables. This is called an $m \times n$ S-Box and is often implemented as a lookup table. These S-Boxes are carefully chosen to resist linear and differential cryptanalysis.

Since cellular automata have the same expressiveness as Turing Machines, any Boolean function can be simulated with a cellular automata. Also, considering that S-Boxes are merely Boolean functions that map n inputs to m outputs, we are able to create CA-based S-Boxes, where the automata generates a n bit output through the temporal evolution of a n bit input (the seed of the automata). Szaban and Seredynski [SS08] provide an excellent example of how such system can be implemented with cellular automata. Their system corresponds to a 8×8 S-Boxes similar to the AES S-Boxes. The configuration vector of the system contains all the necessary information: the initial state of the CA, rules, number of evolutions and which cells work as the input (the output cells are the same a the input). The applied rules were selected after a series of tests, where non-linearity and autocorrelation were measured at different evolutionary instants. The injectivity of the box were also measured for different CA sizes. These measurements are essential to increase the security of the system against differential cryptanalysis [BS91] and linear cryptanalysis [Mat94], especially in private-key block ciphers. From their experiments, Szaban and Seredynski came to the conclusion that for a 8-bit CA (representing a S-Box with a maximum of 8-bit input), rules 30, 57, 86, 99, 135 and 149 presented comparable results to classical designs [CJS05, NM07, MBC$^+$99]. They also verified that by increasing the size of the automata, rules 57 and 99 were not longer suitable, due to the lower number of different outputs generated by the S-Box (8% of 256 possible values).

In a later paper [SS09], Szaban et al. proposed the construction of 6×4 S-Boxes using CA. These boxes had a functionality equivalent to the S-Boxes used the DES system. They reused the same rules used in the previous paper [SS08] as the basis of construction and compared them with DES' S-Boxes in terms of linearity, autocorrelation, balance and strict avalanche criterion [WT86, For, CSS$^+$05, KMI91]. The experiments showed that their S-Boxes presented non-linearity, were balanced and exhibited a low autocorrelation, and in many cases the results were better than the classical tables of DES S-Boxes.

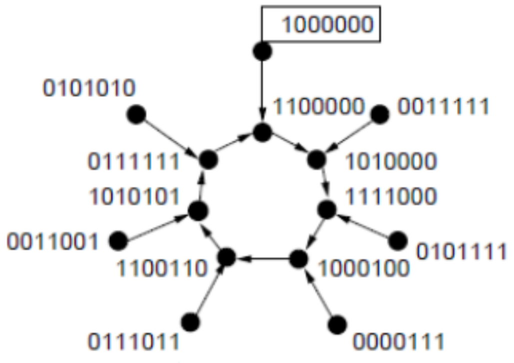
The results of Szaban et al. demonstrated that using cellular automata yields S-Boxes that have the same, if not better performance than the classical ones. Also CA-based S-Boxes have three main advantages against classical S-Boxes: they are dynamic (changing the rules, we change the S-Box), their memory footprint can be reduced and due to the parallel nature of CA, their generation and implementation are fast and efficient.

We can also mention the work of Joshi, Mukhopadhyay and Chowdhury [JMC06], with their proposed CA-based involutional block cipher capable of resisting conventional cryptanalysis, such as linear and differential cryptanalysis. In this system, CAs are used to achieve confusion through S-Boxes and diffusion or avalanche effect through Diffusion Boxes or D-Boxes, where the data is operated on by linear transformations. For hardware implementation efficiency, this block cipher uses a self-invertible structure, and therefore there is no need for different structures for the encryption and decryption operations. According to the authors, these S-Boxes are as strong as the ones of Seberry et al. [SZZ] and simpler to implement. However, this scheme presents a fatal cryptographic vulnerability. This vulnerability was demonstrated in 2007 by Sung et al.

[SHH07], where they were able to construct the decrypt function without the knowledge of the the secret key.

4.5 CA Attractors

Abdo et al. [ALI+13] proposed a cryptoscheme based on the cyclic behavior of some elementary cellular automata. As we have seen in section 3.2.4, pp. 19, after a number of steps, the CA enters in a cyclic trajectory, which we call *attractor.* All other trajectories that are not part of the cycle, are called transients and they start in inaccessible states, i.e., in states that cannot be reached from another state. An attractor and all the transients that lead to it form a basin, for example, take the following one:



While the trajectories 0101010 → 0111111 and 1000000 → 1100000 are transients, the trajectory 0111111 → 1100000 → · · · → 1010101 → 0111111 is the cyclic trajectory of the basin. In figure 4.4 we can observe all possible basins of rule 60 in a one-dimensional CA of size 7. Note that the basins B_3 through B_7 are similar to B_1 and B_2.

Let k denote the period of a cyclic trajectory or attractor. Hence the automata should return to its initial state after k steps. Some attractors exhibit, called *unity attractors*, exhibit the property that applying a XOR operation to all configurations of a cyclic trajectory, yields zero, i.e.:

$$\bigoplus_{i=1}^{k} c^i(x) = 0$$

where k is the length or period of the attractor and $c^i(x)$ is the state of a cell at step i. For example, consider basin B_8 from figure 4.4. If we apply a bitwise XOR operation to

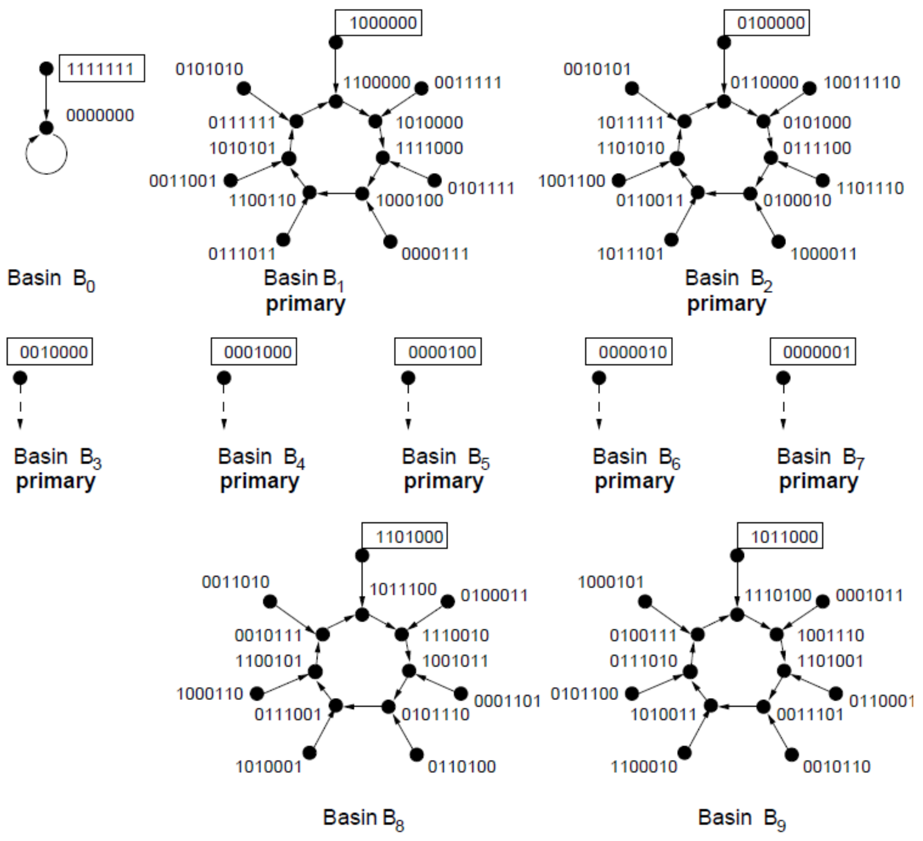


FIGURE 4.4: All basins of rule 60 applied to a one-dimensional CA of size 7.

all seven configurations that form the cyclic trajectory of the basin we have:

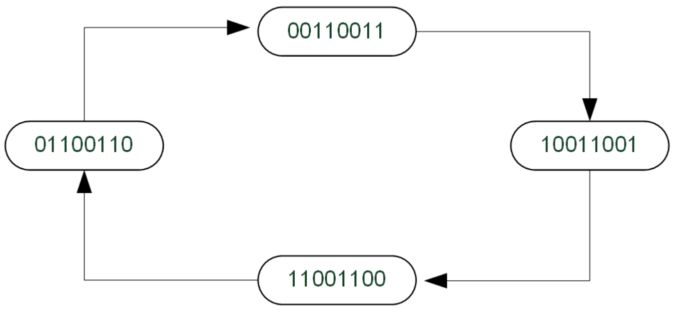
$$1011100 \oplus 1110010 \oplus 1001011 \oplus 0101110 \oplus 0111001 \oplus 1100101 \oplus 0010111 = \mathbf{0000000}$$

This property forms the basis of the cryptographic system. The encryption is done by performing a XOR operation between the plaintext and t configurations of a cyclic trajectory, with $0 < t < k$. To decrypt the ciphertext, we just need to XOR the ciphertext with the remaining configurations of the unity attractor. If M be the plaintext and C the ciphertext, the encryption/decryption is performed according to the equations:

$$\begin{aligned} C &= M \bigoplus \left(\bigoplus_{i=1}^{t} c^i\right); \quad \textit{with}\;\; t < k \\ M &= C \bigoplus \left(\bigoplus_{i=t+1}^{k} c^i\right) \end{aligned}$$

where c^i represents the configuration of the CA at instant i.

Consider the following example applied to plaintext $M = 1001001$, with $t = 2$ and the unity attractor of rule 85:



The ciphertext is then:

$$C = \mathbf{1001001} \oplus (00110011 \oplus 10011001) = 11100011$$

To recover the plaintext, we XORed the ciphertext with the remaining states in the cycle:

$$M = 11100011 \oplus (11001100 \oplus 01100110) = \mathbf{1001001}$$

The control parameters of this cryptosystem include the rule to apply, the attractor and the number of steps. These are generated with the help a non-linear system and by the plaintext itself. Therefore different plaintexts generate different control parameters, resulting in an effective way to resist chosen and known-plaintext attacks. For image encryption, Shannon's entropy presented by this system appears to be superior to both AES and DES systems. They also present a very low correlation of adjacent pixels, a high resistance to differential attacks and a high sensitivity to parameter variations.

4.6 Hash Functions

Cryptographic hash functions play a major role in modern cryptography. Its uses include software integrity, timestamps, message authentication, one-time passwords and digital signatures. A hash function maps a message of an arbitrary length to an output of different or equal size known as the fingerprint or the message digest. If the message is altered, the fingerprint is no longer valid, flagging then a possible data corruption.

A one-way hash function maps an input message m of arbitrary length to a fixed-length output h (hash) such that:

- pre-image resistant — given the hash value h, it is hard to find the message m;
- second pre-image resistant — it is computably infeasible to find two messages, m and m', such that $h(m) = h(m')$;
- collision resistant — given a message m, it is computably infeasible to compute another message m' such that $h(m) = h(m')$.

In 89, Damgård [Dam89] made the first attempt to build fast and collision free one-way hash functions using CAs. He presented three examples of collision-free functions, based on well know constructions: modular squaring [Gir88], Wolfram pseudorandom generator [Wol85] and on the knapsack problem. However, because their poor choice of constructors, all their functions have been successfully attacked, the knapsack-based hash function by Camion et al. [CP91] and the Wolfram based by Boer and Daemen et al. [DGV93].

Daemen et al. [DGV93] presented a framework for the direct design of collision free hash functions based on CA. Mihaljevic [MZI99] proposed a family of fast and dedicated one-way hash functions based in linear CA over GF(q). However, since they provide neither any specific neighborhood nor any applicable rule, the validity of their system is still unknown, since without the CA rules, it is not possible to determine the characteristic of the scheme. Also, they did not provide enough experimental results on the security of the system. The only sure thing, is that the security of the system is based on confusion and diffusion.

In a more recent work, Jeon et al. [Jeo13] proposed an architecture which is based on the parallelism and logical bitwise operations on CA, which makes the entire structure somewhat simple. However, this scheme lacks a comprehensive security analysis.

Chapter 5

Kari's Cryptosystem

As far as we could find, there are only two proposed CA-based public-key cryptosystems: Kari [Kar92] with his scheme based on the problem of computing the inverse of high-dimensional RCAs, and Guan's [Gua87], which is based on nonlinear systems of equations, whose solution is a NP-complete problem. In this chapter, we focus on Kari's system and on a possible implementation suggested by Clarridge and Salomaa [CS09].

5.1 Kari Proposal

Kari's idea is based on the intractable computing problem of determining the inverse of d-dimensional RCAs, with $d \geq 2$:

Theorem 5.1. *[Kar94] It is undecidable whether a given two-dimensional cellular automaton with the Moore neighborhood is surjective. This is also true even with more restricted neighborhoods, such as von Neumann.*

From the above theorem we can infer the following corollary:

Corollary 5.2. *[Kar92] For every algorithm that finds the inverses of all two-dimensional RCA given as input, and for every computable function f, there is an RCA, B, such that the algorithm does not find the inverse of B in time $f(s)$ where s is the size of B, e.g., the number of states.*

Thus, we can publish a d-dimensional RCA, B (with $d \geq 2$), without impairing the secrecy of its inverse B^{-1}. However, the main problem with this idea lies on how to create these hard to invert high dimensional RCA. Kari suggested the use of composition, where we combine multiple and simple RCAs, in order to create a more complex and hard to invert RCAs. Hence, by successively applying simple reversible operations, such as permutations, to a cellular automata, we are able to obtain a hard to invert cellular automata. Let C_i represent a CA, for $i = 1, 2, ..., n$, upon which a simple reversible operation is applied. If our final CA is the result of their composition:

$$C = C_n \circ C_{n-1} \circ \cdots \circ C_2 \circ C_1 \tag{5.1}$$

we are able to obtain a CA, C, whose inverse C^{-1}, can be hard to determine without the knowledge of the inverse of each C_i:

$$C^{-1} = C_1^{-1} \circ C_2^{-1} \circ \cdots \circ C_{n-1}^{-1} \circ C_n^{-1} \tag{5.2}$$

Kari suggested the usage of a special type of CA, called *marker CA*, which is a CA whose local rule is given by something called a marker. As illustrated in figure 5.1, a marker consists of a finite set of patterns, $\mathcal{P}$, together with a permutation π of the state set S, the update function of the marker. Each pattern is essentially a set of values that must be present around a cell for its state to be updated according to π, i.e., the update function operates on a cell, if and only if, there is a match between the states of its neighboring cells and a pattern in $\mathcal{P}$. However, since there is no order by which each pattern of $\mathcal{P}$ is compared to the states of the neighbors of a cell, no pattern should be a sub-pattern of another.

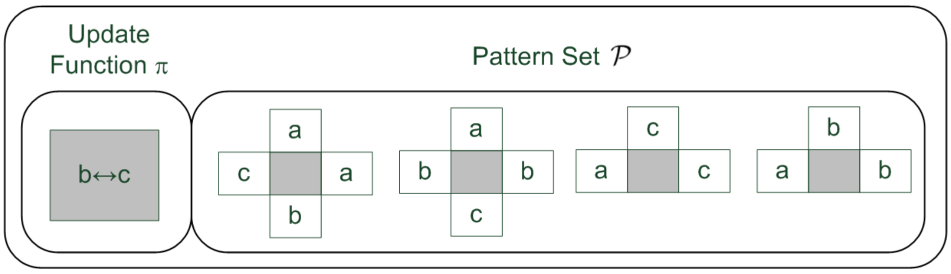


FIGURE 5.1: Example of a marker. The gray square in the patterns represents the cell on which the marker is been applied to.

More precisely, a pattern, $P \in \mathcal{P}$, is given by a mapping from a finite subset X_P of $\mathbb{Z}^d$, with $0 \notin X_P$, into the state set S:

$$P : X_P \to S$$

The subset X_P represents the shape of the pattern P, giving the neighborhood around the cell on which the marker is been applied to. The values that P takes are to be compared, for each cell x, to the states of the cells $x + X_P$ in the current CA configuration. Take, for instance, the first pattern of the previous marker. In this case, X_P is $\{ (1,0), (0,1), (-1,0), (0,-1) \}$, representing the right, top, left and bottom cells, and for the sake of simplicity, we are going to express its corresponding pattern P by the tuple (a, a, c, b).

The marker operates by checking, for each cell $x \in \mathbb{Z}^d$, if the values of one of its patterns P coincide with values of the neighbors of x in the current CA configuration, i.e., if at neighborhood $x + X_P$ of x:

$$\phi^t(x + r) = P(r),\ \forall r \in X_P$$

where $\phi^t(x)$ denotes the state of cell $x \in \mathbb{Z}^d$ at instant t (as defined in section 3.2.2). If the above condition if verified, then ϕ is applied to the cell and $\phi^{t+1}(x) = \pi(\phi^t(x))$; if not, the state of the cell remains inalterable, $\phi^{t+1}(x) = \phi^t(x)$.

Because the pattern set $\mathcal{P}$ can be composed by patterns with different shapes, the dimensionality of a marker is then given by the highest dimension of the vector space on which the patterns of the marker are defined. In the case of the marker presented in figure 5.1, we have a two-dimensional marker, because all patterns lies on the vector space $\mathbb{Z}^2$.

Definition 5.3. A *Marker CA* is a 5-tuple $(d, S, \pi, \mathcal{X}, \mathcal{P})$, where d is the dimension of the cellular space, S the state set, π the update function, $\mathcal{X}$ the set of shapes of the pattern set with each element $X_P \subseteq \mathbb{Z}^d$ and $0 \notin X_P$, and finally the pattern set $\mathcal{P}$, where each element $P \in \mathcal{P}$, has a corresponding shape defined by X_P and where each element is not a sub-element of another. The update function π is applied to a cell x if and only if the state of its neighboring cells matches an element of $\mathcal{P}$.

Consider the marker presented in figure 5.1, and apply it to the following text arranged in a 2-dimensional grid and considering periodic boundaries:

b	b	b	a
a	a	b	c
c	b	a	a
b	c	c	a

In this case, $d = 2$, $S = \{a, b, c\}$, $\pi = (b\,c)$, $\mathcal{X} = \{\{\,(1,0),(0,1),(-1,0),(0,-1)\,\},\{\,(1,0),(0,1),(-1,0),(0,-1)\,\},\{\,(1,0),(0,1),(-1,0)\,\},\{\,(1,0),(0,1),(-1,0)\,\}\}$ and $\mathcal{P} = \{(a,a,c,b),(b,a,b,c),(c,c,a),(b,b,a)\}$.

In the transition step from instant t to $t+1$, we check for every cell of the text, if the state of that cell's neighbors coincide with a pattern. If so, the update function is applied to the cell, changing its state. In figures 5.2 and 5.3 we can observe the application of the marker on two cells.

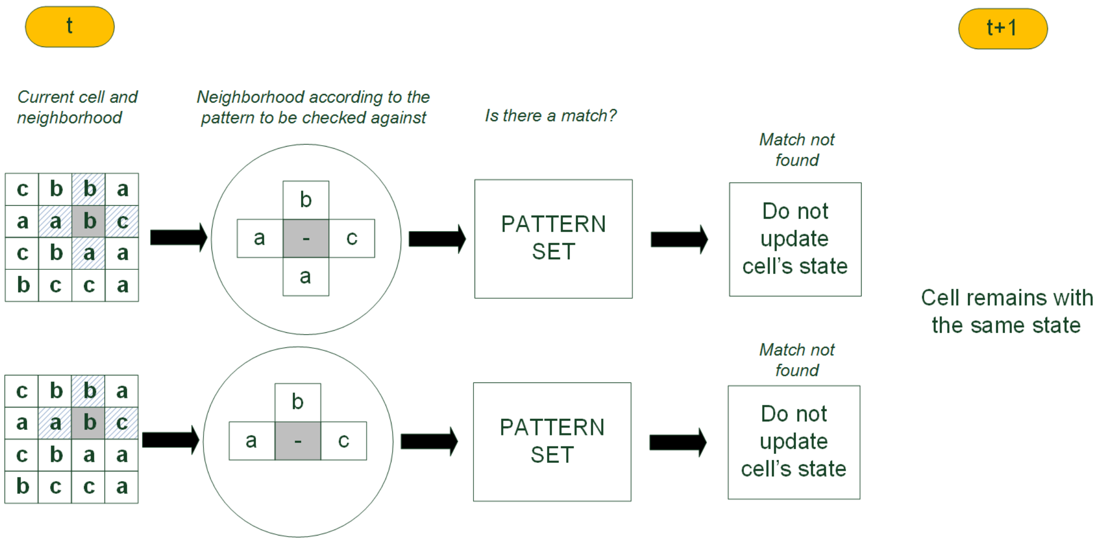

t
t+1
Current cell and neighborhood
Neighborhood according to the pattern to be checked against
Is there a match?
Match not found
PATTERN SET
Do not update cell's state
Match not found
PATTERN SET
Do not update cell's state
Cell remains with the same state

FIGURE 5.2: Marker operating on a cell.

Applying the same process to the remaining cells, the configuration at $t + 1$ is:

b	b	b	a
a	a	b	c
c	b	a	a
b	c	c	a

$\longrightarrow$

c	b	b	a
a	a	b	c
c	b	a	a
b	c	b	a

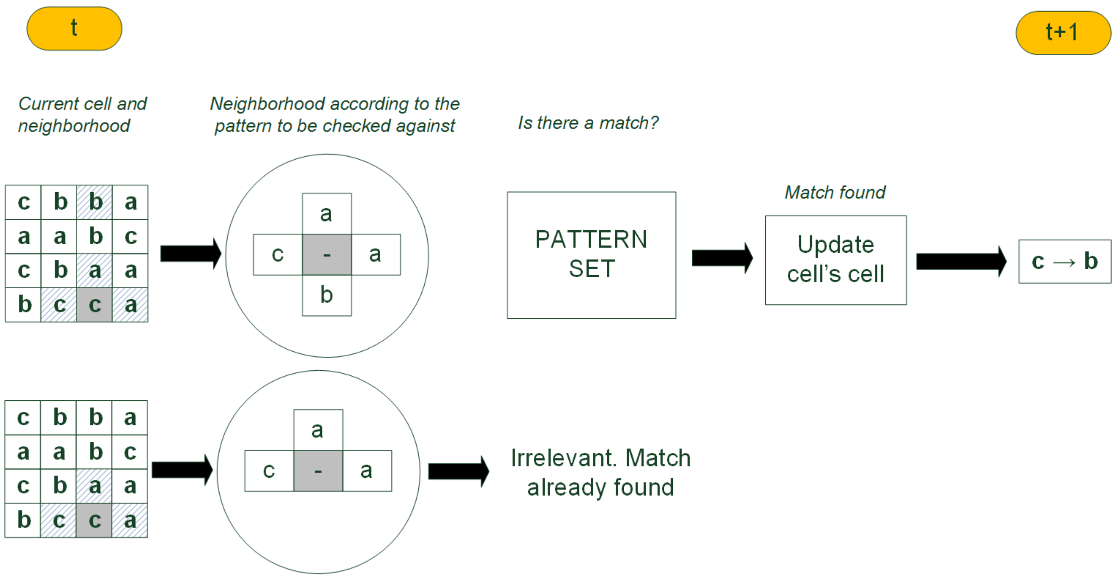

t
t+1
Current cell and neighborhood
Neighborhood according to the pattern to be checked against
Is there a match?
Match found
PATTERN SET
Update cell's cell
c → b
Irrelevant. Match already found

FIGURE 5.3: Marker operating on another cell.

The shaded the cells are the only ones that were modified by the marker.

5.1.1 Self-Invertible Markers

Since we are interested in the application of markers in public key cryptography, we need to demand local rules capable of providing a CA with the capacity of recovering information. To achieve that, all markers CA present in the composition should be invertible.

Let M be an arbitrary d-dimensional marker with $d \geq 2$, state set S, pattern set $\mathcal{P}$ and update function π. We are only interested in a special type of invertible markers, self-invertible markers, i.e., markers with the following property:

$$M \circ M = Id$$

where Id is the *identity marker* on the set S. An identify marker, is a marker whose update function π is an involutory function. If, for example, π is defined by a permutation of the state set S, then it must be an identity permutation, which maps every element of the set to itself.

Let S_π^M (or simply S_π) represent the set of states that are changed by the update function π of a marker M. For instance, if $a, b, c, d, e \in S$ and $\pi = (a\,b)\,(c\,e)$, then $S_\pi = \{a, b, c, e\}$.

Consider a cell x and its respective neighbors at instant t. Let M be a reversible marker. In instant $t+1$, after the application of M, the states of either cell x or some of its neighbors may have been altered by M. Depending on the states of the neighboring cells, M must be able to recover the original configuration of x and its neighborhood.

If no neighbors of cell x have states in S_π, then at $t+1$ the states of those cells remain the same as they were at t, since, regardless of their respective neighbors, π did not change them. So, if, from t to $t+1$, M changed x (which means there was a match between the states of the neighbors of x and a pattern), a second application of M at $t+1$ will once again change the state x back to its original state, because there is still a match between the states of the neighbors of x and a pattern of M at $t+1$, and π is an involution.

On the other hand, if some neighbors of cell x have states in S_π, then at $t+1$, after the application of M, not only the value of cell x could have changed but also the value of its neighbors. Therefore, in order to return x to its original state, it is necessary to ensure that there is still a match between a pattern of M and the potentially modified states of its neighbors. This is easily achieved by ensuring that, if there is a pattern containing cells with states in S_π, then the pattern set must also contain all patterns that can be created by populating these cells with all possible states combinations of the elements of S_π. For example, if $\pi = (a\,c)$, then $S_\pi = \{a, c\}$. If M contains the following pattern with both upper cells having states in S_π:

a	c
-	b

then M must also contain the next three patterns:

a	a
-	b

c	c
-	b

c	a
-	b

which cover all possible states combinations of S_π in both upper cells.

Constructing self-invertible markers. Given a marker M with pattern set $\mathcal{P}$, for each $P \in \mathcal{P}$, we let $Y_P = \{r \in X_P \mid P(r) \in S_\pi\}$. For M to be self-invertible, then either $\forall P \in \mathcal{P}, Y_P = \emptyset$ or for every $P \in \mathcal{P}$ such that $Y_P \neq \emptyset$, there is a $\mathcal{P}' \subseteq \mathcal{P}$ where all its elements are created by populating the cells specified by Y_P with all possible combinations of the states in S_π. The size of this subset $\mathcal{P}'$ is therefore: $|\mathcal{P}'| = |S_\pi|^{|Y_P|}$. $\mathcal{P}'$ must also met three conditions:

- All its patterns must have the same shape, i.e., $\forall Q, R \in \mathcal{P}' : X_Q = X_R$.
- All its patterns must originate the same set Y_P: $\forall Q \in \mathcal{P}' : Y_Q = Y_P$.
- All its patterns must have the same states of the cells not in Y_P: $\forall Q, R \in \mathcal{P}', \forall r \in (X_P \setminus Y_P) : P_Q(r) = P_R(r)$.

It is trivial to verify that by applying the marker's update function to any of its patterns, it creates another pattern that must also belong to the pattern set, i.e., $\forall P_i \in \mathcal{P}, \exists P_j \in \mathcal{P}$, $X_i = X_j$, and $\forall r \in X_i : \pi(P_i(r)) = P_j(r)$.

Example 1 Consider marker M_1 depicted in the following figure 5.4.

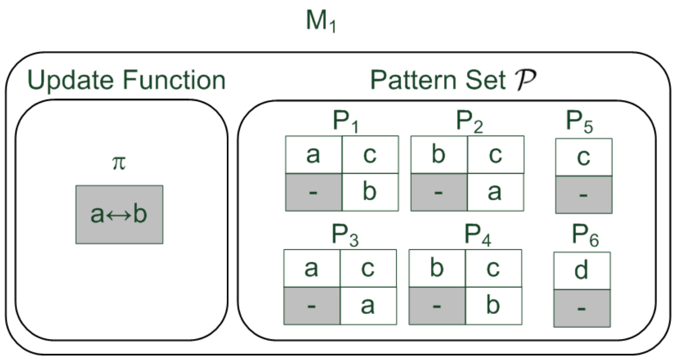


FIGURE 5.4: A reversible marker.

The update function of M_1 permutes only two states 'a' and 'b', therefore $S_\pi = \{a, b\}$.

Consider pattern P_1. The cells on the right and top of the shaded cell have states in S_π, thus $Y_P = \{(0,1),(1,0)\}$. According to our constructor, the pattern set should contain a subset with patterns matching all possible combinations that are possible to populate Y_P with S_π. Also, all patterns in this subset must have not only the same shape and share the same Y_P but also share the same states on all cells not in Y_P (the top right cell), which in this case is 'c'. Clearly, all these conditions are satisfied by the set $\{P_1, P_2, P_3, P_4\}$.

Consider now pattern P_5. The state of the top cell is not in S_π, and therefore $Y_P = \emptyset$. The same goes for P_6. According to our constructor, this marker is then self-invertible.

In figure 5.5 we can observe that for the given configuration of the 4×4 cellular grid, the double application of M, considering cyclic boundaries, does lead back to the initial configuration.

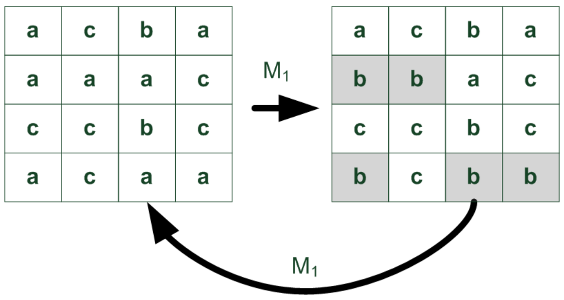


FIGURE 5.5: The effects of a reversible marker on a given configuration.

Example 2 Consider marker M_2 represented in figure 5.6. Again, $S_\pi = \{a, b\}$. But

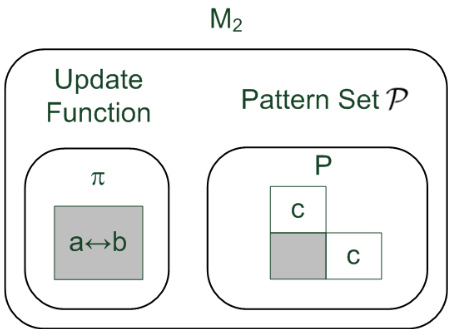


FIGURE 5.6: Another reversible marker.

since the only pattern in $\mathcal{P}$ does not contain states present in S_π, we have $Y_P = \emptyset$. Hence, this marker is self-invertible.

Example 3 As for a non-reversible marker, consider marker M_3 represented in figure 5.7.

Again, $S_\pi = \{a, b\}$ and $Y_P = \{(0, 1), (1, 0)\}$. Not all combinations of that are possible to populate Y_P with S_π, $\{aa, ab, ba, bb\}$, are covered, since the pattern set has only $\{aa\}$. Therefore M_3 cannot be self-invertible.

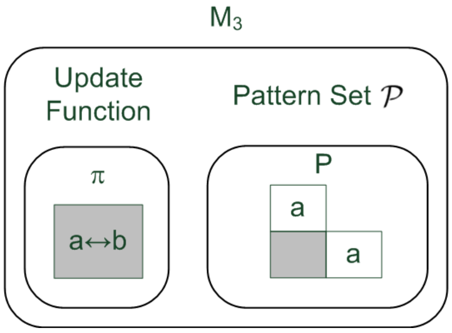


FIGURE 5.7: A non-reversible marker.

In figure 5.8, and considering again periodic boundaries, we can attest that for the presented configuration of the grid, the double application of M_3, does not lead back to the initial configuration.

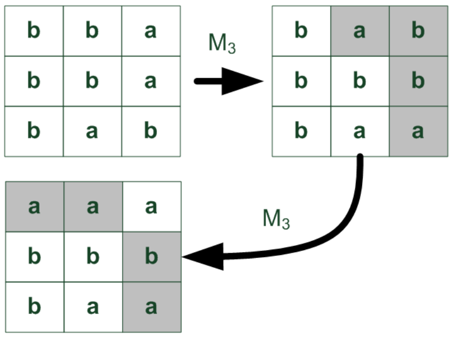


FIGURE 5.8: The effects of a non-reversible marker on a given configuration.

5.1.2 Public and Private Keys

Let $\mathcal{M}$ represent a set of n reversible markers, M_i, and S the state set of a CA. The private key of the system is the set $\mathcal{M}$ and the public key is given by the composition of its elements according to equation (5.1), which is published in the form of a table. The order of the markers in the in the composition is a factor quite important, and this importance will be explained later in this section.

Let $N \subseteq \mathbb{Z}^d$ represent the shape of the public key. The best way to interpret the public key, is to considered it exactly as any other CA rule, with N as the neighborhood vector. Hence we must have by one hand all possible combinations that it is possible to populate N with the state set S and on the other, the state that the origin cell (the cell that is been acted upon) should transition to for each one of the combinations. Therefore the table must be formed by two columns. On the first column, we have the states of neighboring cells and on the second, the final state of the origin cell.

Consider, for instance, a table representing a public key with a cross shape around the origin, $N = \{(0,0),(0,1),(0,-1),(1,0),(-1,0)\}$ (origin, top, bottom, right and left), and state set $S = \{a,b,c\}$. We have then a table with $3^5 = 243$ entries. Figure 5.9 depicts a partial graphical representation of this hypothetical table.

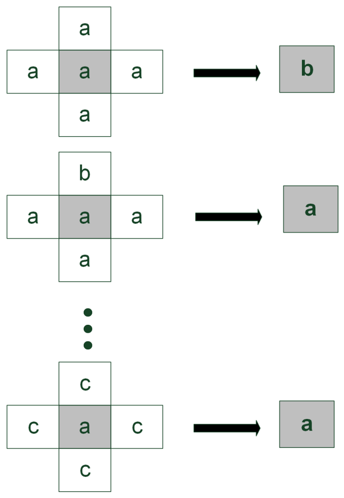
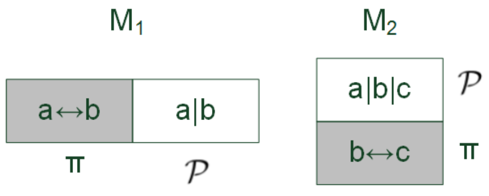


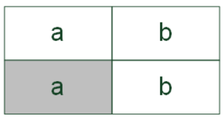
FIGURE 5.9: A few entries in a hypothetical public key table.

Since the public key table is just like any other CA rule, its application is straightforward and an example is given in section 5.1.3, pp. 60.

Now we must tackle the problem of how to determine the transition state of the origin cell for each entry of the table. Consider the following set of two two-dimensional reversible markers $\mathcal{M} = \{M_1, M_2\}$ with an alphabet $S = \{a,b,c\}$ and a single permutation as their update functions:

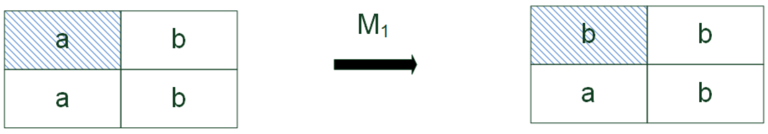


The notation $a|b$ denotes that the cell can be either in the state 'a' or 'b'. Lets also consider a square neighborhood shape for the public key $N = \{(0,0),(0,1),(0,1),$ $(1,1)\}$, where $(0,0)$ denotes the origin cell and it corresponds to the cell at the bottom left of the square. Let us now determine the transition state of the origin cell for the following table's entry:

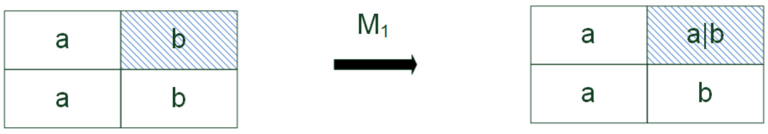


In order to determine the transition state, we must apply all the markers $M_i \in \mathcal{M}$ to every cell of the entry. Lets consider the application order as M_1 followed by M_2.

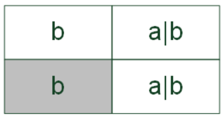
Let us start by applying M_1 to the top left cell. The update function π will be applied to the cell, if and only if the state of its right neighbor is either 'a' or 'b', which in this case, it is true, and therefore π is applied to the cell, changing its value to 'b':



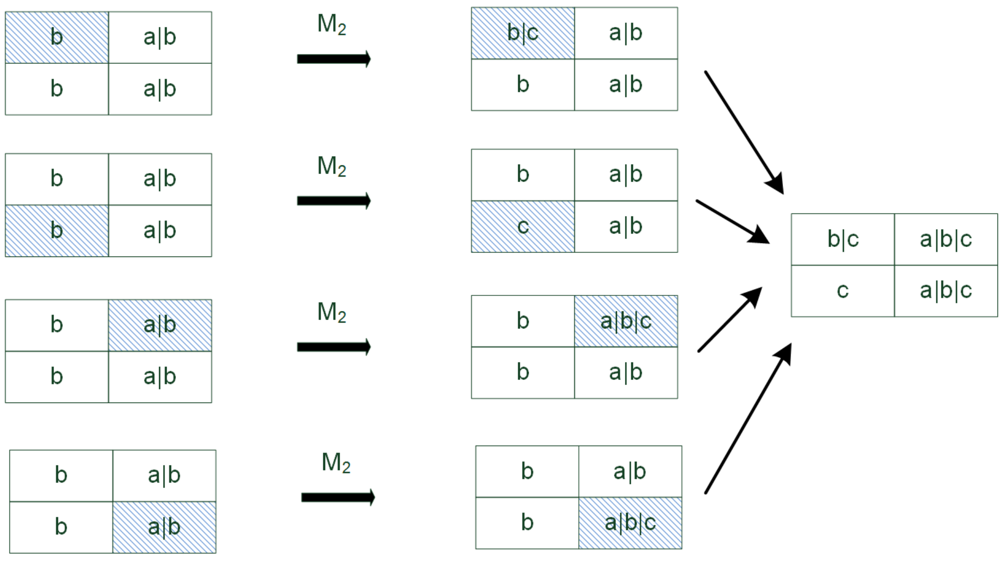
Let us pass to the top right cell. Because its neighbor trespass the limits of the shape, we must assume that the state of the neighboring cell could be any state of S and therefore we are unable to know if π should be applied to the cell or not. Hence, the final state of the top right cell must reflect this uncertainty by considering both possibilities: if π is applied, then the state of the cell changes from 'b' to 'a', otherwise the state remains unchanged. So, the final state of this cell should be either'a' or 'b':



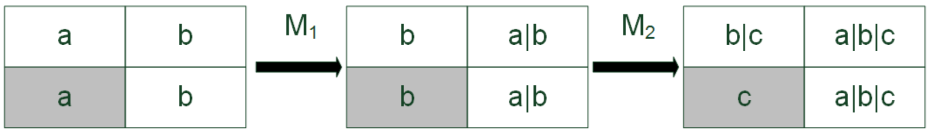
Using the same reasoning for the remaining two cells, we end up with the final configuration:



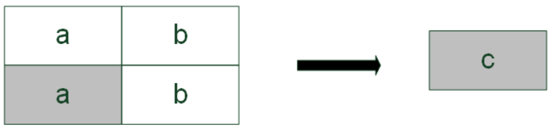
Following the same logic for the application of M_2 to the previous configuration:



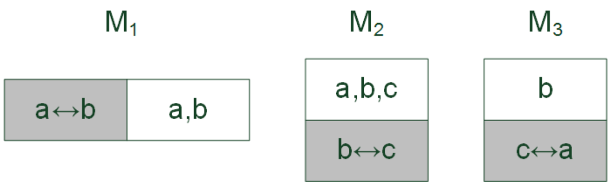
So we have the following sequence of configuration changes caused by $\mathcal{M}$:



The final configuration of the origin cell is then 'c' and we can add to the public key table the entry and its corresponding transition state:

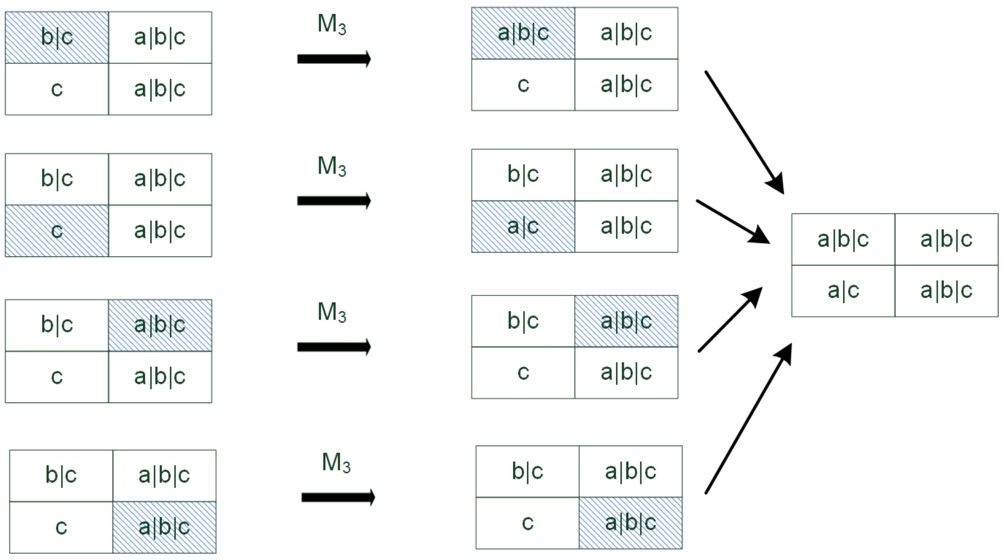


Let us now add a new marker M_3 to the set $\mathcal{M}$:

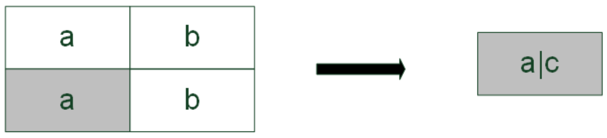


Consider the application order of the markers as M_1 followed by M_2 followed by M_3.

Using the same example entry and the previous resulting configuration, the application of M_3 results in:



Hence the corresponding transition state of this entry is no longer a single state but two possible states: either'*a*' or '*c*', i.e., it is uncertain:



Since we do not know which final state we should use, it is quite obvious that we cannot use this entry, because otherwise it would fail to be injective. We can then conclude that the set $\mathcal{M}$ should be not only exclusively formed with reversible markers $\mathcal{M}$, but should also be able to create a deterministic public key, i.e., all entries should have one and only one corresponding transition state.

For the entry above there are two solutions to solve the uncertainty problem. We could, for instance, increase N. But, this solution is undesirable for practical reasons, as will be seen in section 5.1.6. A more elegant solution is to eliminate the uncertainty all together. Because the problem appears only when the third marker is applied, a suitable solution can easily be found: we only need to ensure that after the application of marker M_3 on cell $(0, 0)$, this cell remains with only one state. This can be achieved by ensuring that one of two conditions are met:

- The pattern set of marker M_3 contains all possible states that the cell $(0, 1)$ have before the application of M_3, which in this case are '*b*' and '*c*', resulting on either M_{3A} or M_{3B} as alternatives to M_3 (see figure 5.10).

- The pattern set of marker M_3 does not contain any of the states of cell $(0, 1)$ before the application of M_3, resulting on M_{3C}, M_{3D} or M_{3E} as illustrate in figure 5.10.

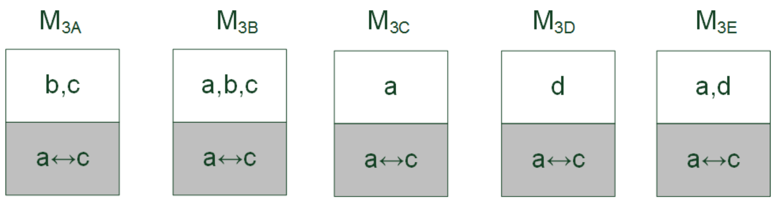


FIGURE 5.10: Alternatives to M_3.

Naturally, even after the changes, the alternative M_3 must also be self-invertible. Therefore, according to section 5.1.1, markers M_{3A}, M_{3C} and M_{3E} are discarded because they are not self-invertibles. We have now the following alternative valid sets to $\mathcal{M}$: $\mathcal{M}' = \{M_1, M_2, M_{3B}\}$ and $\mathcal{M}'' = \{M_1, M_2, M_{3C}\}$. Although the new M_3 solves the problem for this particular entry, there are no guarantees that this new marker will not cause the same problem to any of the others 254 entries.

From this example, we can see that, for the generation of a valid set of markers (all markers are self-invertible and the set can only generate a deterministic public key), a generation algorithm should not only be able to create invertible markers but should also take into account all previous markers in the set by tracking their potential actions on each cell. This technique is used by Clarridge and Salomaa in their algorithm to keep the neighborhood size constant, as we will see in section 5.2. We can also conclude that the application order of the markers is of the utmost importance, because different application orders can generate invalid public key's entries.

5.1.3 Encryption and Decryption

Now that we know how to create both the private and public keys, lets see how to use them. Figure 5.11 illustrates how the encryption is executed using the public key table. For each of the cells, we find a match between the states of its neighboring cells with an entry of the public key. Once founded, we replace the state of the cell by the state of the corresponding entry.

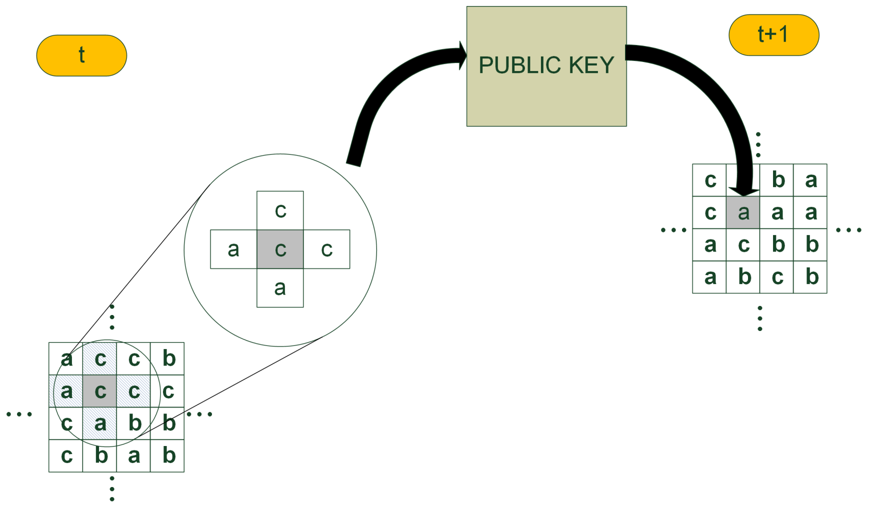


FIGURE 5.11: Encryption using the public key.

The decryption process is simply done by the successive application of the markers' inverse, M_i^{-1}, but in the reverse order of the one used in the generation of the public key.

To clarify the encryption and decryption processes, we next present a simple example.

Example. Let us consider a two-dimensional CA, with the ternary alphabet, $S = \{a, b, c\}$ and null boundaries (with state 'a' as the null state), a set of markers formed by four one-dimensional markers, $\mathcal{M} = \{M_1, M_2, M_3, M_4\}$ as illustrated in figure 5.12 and a public key with a 2×2 square shape, $N = (\,(0,0),(1,0),(0,1),(1,1)\,)$.

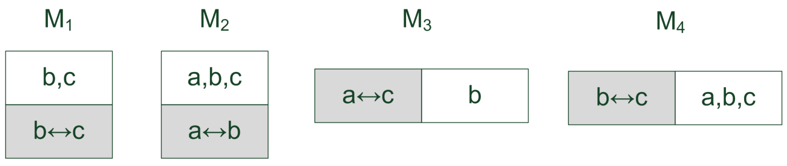


FIGURE 5.12: A set of reversible markers.

As previous discussed, we first compute the public key table through the composition:

$$M = M_4 \circ M_3 \circ M_2 \circ M_1$$

In this case, the table has $3^4 = 81$ entries, where all markers in $\mathcal{M}$ are successively applied to each entry in order to determine the transition state of the origin cell. In figure 5.13 we present the determination of corresponding states of three entries, where S symbolizes that the cell can be in any state of S and '$-$' represents "it is irrelevant".

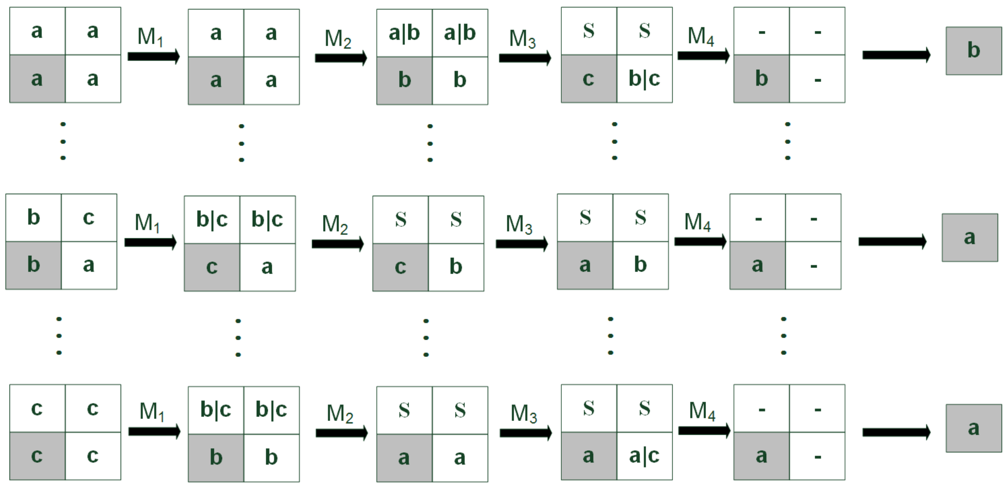


FIGURE 5.13: Creating the entries of the public key.

The encryption process using the public key is rather straightforward. In the transition instant from instant t to $t+1$, we proceed through every cell of the plaintext and use the state of the cell and of its neighbors as the key value for the table. Once we find it, we replace the state of the cell with the entry's respective value. Figure 5.14 shows the result of an example encryption of the text using the public key.

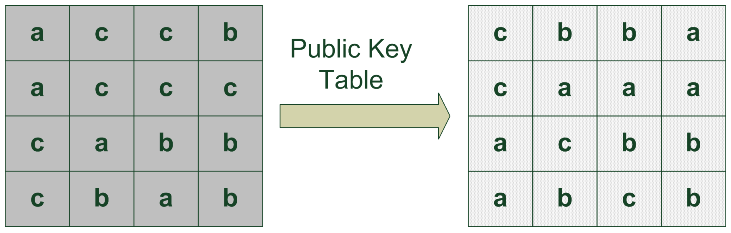


FIGURE 5.14: Encryption of a plaintext using the public key.

Since all markers of $\mathcal{M}$ are self-invertibles ($M_i = M_i^{-1}$), the inverse composition is determine by:

$$M^{-1} = M_1 \circ M_2 \circ M_3 \circ M_4$$

So, we decrypt the ciphertext by applying the same markers but in the exact reverse order as the one used to create the public key and thus obtaining the original plaintext (see figure 5.15).

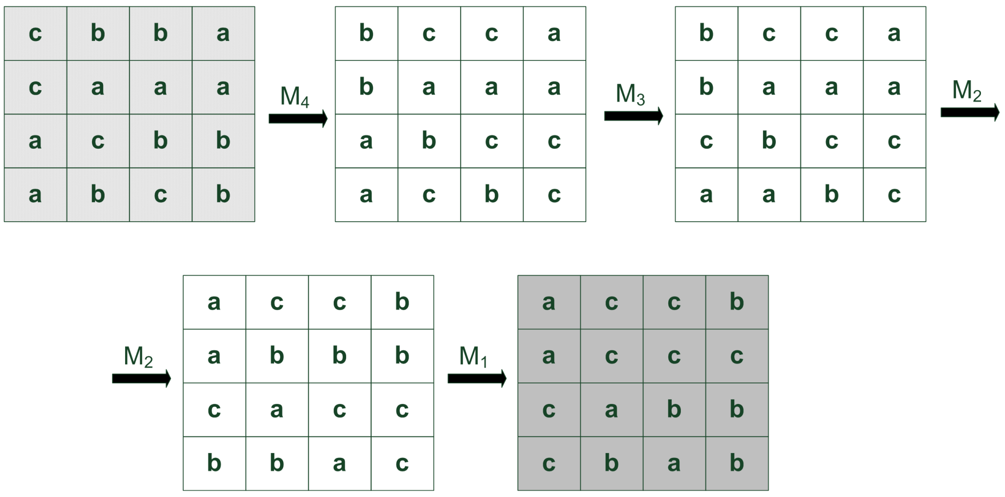


FIGURE 5.15: Decryption of the ciphertext.

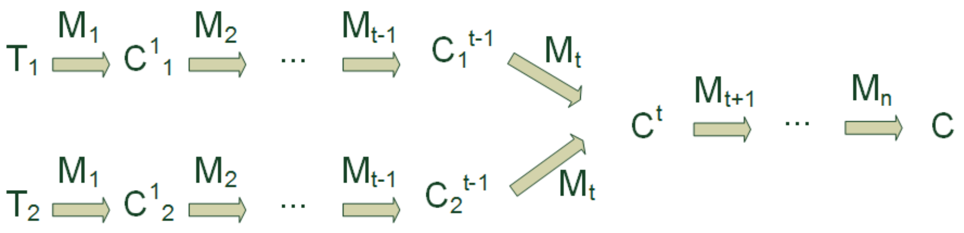
5.1.4 Injectivity

Let $\mathcal{M}$ represent a set of reversible markers and T a plaintext with C as its corresponding ciphertext after the successive application of all markers in $\mathcal{M}$:

$$T \xrightarrow{\mathcal{M}} C$$

Lemma 5.4. *The same ciphertext cannot be created from two distinct plaintexts with the same set of reversible markers.*

Proof. Let $M_t \in \mathcal{M}$ be a reversible marker, with π as the update transition function and $\mathcal{P}$ the pattern set. Let α and β represent two cells at the same position in two distinct texts, T_1 and T_2, with s_α and s_β as their respective states and $S_N(\alpha)$ and $S_N(\beta)$ as the set of states of their respective neighbors at instant t, with $n = |\mathcal{M}|$ and $t \leq n$. For two different plaintexts to originate the same ciphertext, there must be an instant t, in the markers' application chain, after which, both of them become equal:



Since the inequality between the two texts implies that they must be distinct, at least, between two cells, to prove the lemma we only need to prove that if before instant t these cells are different then, after the application of marker M_t, the texts remain different.

To show this, we analyze all relevant cases that can occur to the state of cells α and β and the state of their respective neighbors, during the application of marker M_t.

Before the application of marker M_t, the relation between cells α and β can fall into one of two relevant cases:

$$s_\alpha = \pi(s_\beta)$$
$$s_\alpha \neq s_\beta \ \ and \ \ s_\alpha \neq \pi(s_\beta)$$

Since the effects of the application of the marker on the cells, depends on the state of their respective neighbors, we must also consider all possible relations between them and the marker's pattern set. We have, therefore, four possible cases, depending whether the states of those neighbors have or not a match in the pattern set:

$$S_N(\alpha), S_N(\beta) \in \mathcal{P}$$
$$S_N(\alpha) \in \mathcal{P}, S_N(\beta) \notin \mathcal{P}$$
$$S_N(\alpha) \notin \mathcal{P}, S_N(\beta) \in \mathcal{P}$$
$$S_N(\alpha), S_N(\beta) \notin \mathcal{P}$$

Let us analyze each possible case.

Case 1

$$s_\alpha = \pi(s_\beta) \ \ and \ \ S_N(\alpha), S_N(\beta) \in \mathcal{P}$$

Since the state of both neighborhoods are in the pattern set, M_t is applied to both cells. But since both states are images of each other, the application of π just swaps their states, and therefore after the application of M_t the texts are still different.

Cases 2 and 3

$$s_\alpha = \pi(s_\beta) \ \ and \ \ S_N(\alpha) \in \mathcal{P}, S_N(\beta) \notin \mathcal{P}$$

or

$$s_\alpha = \pi(s_\beta) \ \ and \ \ S_N(\alpha) \notin \mathcal{P}, S_N(\beta) \in \mathcal{P}$$

In these cases M_t operates on either α or β, resulting $\phi(s^t_\alpha) = \phi(s^t_\beta)$. However due the reversibility property that we forced upon the markers, we know that elements of $\mathcal{P}$ must be mapped to $\mathcal{P}$, so it is impossible for the states of both their neighborhood ever be equal. Hence the texts remain different.

Case 4

$$s_\alpha = \pi(s_\beta) \ \ \textit{and} \ \ S_N(\alpha), S_N(\beta) \notin \mathcal{P}$$

Marker M_t does not operate on either cell, and so the cells remain the same state.

Case 5

$$s_\alpha \neq s_\beta \ \ \textit{and} \ \ s_\alpha \neq \pi(s_\beta) \ \ \textit{and} \ \ S_N(\alpha), S_N(\beta) \in \mathcal{P}$$

M_t is applied to both cells. Although the state of both cells can change, they are neither equal nor are images of each other, i.e., it is impossible for both to be equal. Consequently, the texts remain different after t.

Cases 6 and 7

$$s_\alpha \neq s_\beta \ \ \textit{and} \ \ s_\alpha \neq \pi(s_\beta) \ \ \textit{and} \ \ S_N(\alpha) \in \mathcal{P}, S_N(\beta) \notin \mathcal{P}$$

or

$$s_\alpha \neq s_\beta \ \ \textit{and} \ \ s_\alpha \neq \pi(s_\beta) \ \ \textit{and} \ \ S_N(\alpha) \notin \mathcal{P}, S_N(\beta) \in \mathcal{P}$$

Although M_t changes either α or β, they are not images of one another, hence they remain different after t.

Case 8

$$s_\alpha \neq s_\beta \ \ \textit{and} \ \ s_\alpha \neq \pi(s_\beta) \ \ \textit{and} \ \ S_N(\alpha), S_N(\beta) \notin \mathcal{P}$$

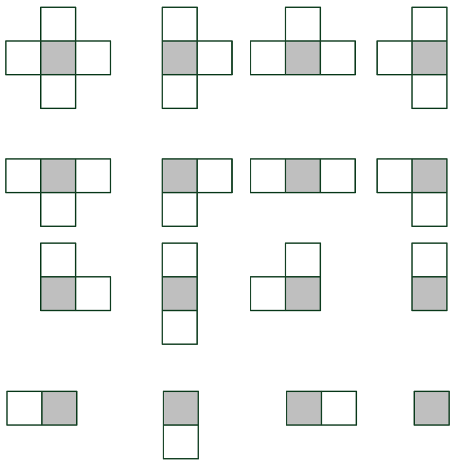
Since neither the state of their neighborhood are in the markers' pattern set, the transition function is not applied, i.e., the cells preserve their state. Hence, both states continue different.

□

5.1.5 Number of Markers

To give an idea on the size of the universe of markers, we now proceed to count them. However, we set two limits: all elements in the pattern set share the same geometrical shape and no assumptions are made about their reversibility. We also assume that the update function of the every marker is a permutation of the state set.

If we consider a public key with a von Neumann neighborhood, $|N| = 5$, the shape of the patterns of the pattern set can have any of the following geometrical shapes:



On top of that, we should also not only take into account that the update function of every marker can be any possible permutation of the set S but also each pattern can be populated with any possible combination of states.

Let $n = |N|$ represent the size of the neighborhood without the origin cell and $s = |S|$ the size of the alphabet S. The number of available geometrical dispositions of the neighborhood is the number of possible geometrical shapes N' that N can contain – $N' \subseteq N \subseteq \mathbb{Z}^d$. This number is simply given by:

$$N_{geo}(n) = 2^n$$

The number of possible pattern sets per neighborhood, is given by all possible ways to populate the neighborhood with the state set S:

$$NPS(s,n) = \left[\sum_{i=1}^{s^n} \binom{s^n}{i}\right] + 1$$

The number of possible permutations of S is:

$$NP(s) = \sum_{i=1}^{\lfloor s/2 \rfloor} \frac{s!}{i!\ 2^i\ (s-2i)!}$$

Finally, the total number of markers is given by:

$$Number\_of\_Markers = \sum_{i=1}^{n} (N_{geo}(i) \times NPS(s,i)) \times NP(s)$$

As we can observe in table 5.1, even with small neighborhoods and state sets, the possible number of markers is quite staggering, and this number have an exponential growth with the size of both N and S.

|S|	**|N|**	N_{geo}	NPS	NP	$Number\_of\_Markers$
3	1	2	8	3	48
3	2	4	512	3	12288
3	3	8	1.34×10^8	3	9.66×10^9
4	1	2	16	9	288
4	2	4	65536	9	4718592
4	3	8	1.84×10^{19}	9	3.98×10^{21}
5	1	2	32	25	1600
5	2	4	33554432	25	6.71×10^9
5	3	8	4.25×10^{37}	25	2.55×10^{40}

TABLE 5.1: Number of distinct markers depending on the number of states and neighborhood size.

5.1.6 Public Key Size

In practical terms, the neighborhood size of the composing RCA should be as small as possible. The reason lies on the memory requirements, necessary to save the public key, as it grows exponentially with $|N|$. A public key with a neighborhood size $|N|$ and with $|S|$ states is represented by a table with $|S|^{|N|}$ entries, representing $|S|^{|N|} log_2 |S|$ bits of storage.

To illustrate this issue, consider the following example. Let our public key have a von Neumann neighborhood with radius 1 ($|N| = 5$) and a state set with $|S| = 10$ elements. We then have a public key with $10^5 = 100000$ entries, requiring at least $10^5\ log_2\ 10 = 332000$ bits or 40.5 KB of memory space.

5.1.7 Safety Against Brute Force Attacks

The basic security of this scheme is guaranteed by corollary 5.2. It is also doubtful that brute force attacks can manage to break the system if large neighborhoods and/or number of states are used. Considering ciphertext blocks of $m \times n$, any attempt in guessing the plaintext from the ciphertext, would result in $|S|^{n \times m}$ possible permutations to tests, which can very easily become computably unfeasible.

Also, seeking to guess the set of markers used in the composition of the public key, would again be computationally infeasible, because of the high number of markers and possible combinations between them, even for small neighborhoods and alphabets sizes.

However, there are two main factors that can affect the vulnerability of this scheme against brute force attacks: the non-uniqueness of the public key and the fact that the set of markers must not only be formed with reversible markers but their order cannot be random. At the moment, the true extend of both weakness can only be verified experimentally on a case-by-case basis.

Consider for example a set composed by two one-dimensional reversible markers as illustrate in the following figure 5.16, with $N = \{(1)\}$ and $S = \{a, b, c\}$. Experimentally

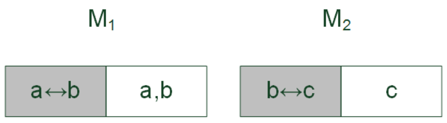


Figure 5.16: Two 1-dimensional reversible markers.

we were able to discover several sets, such as the ones presented in figure 5.17, capable of generate the same public key.

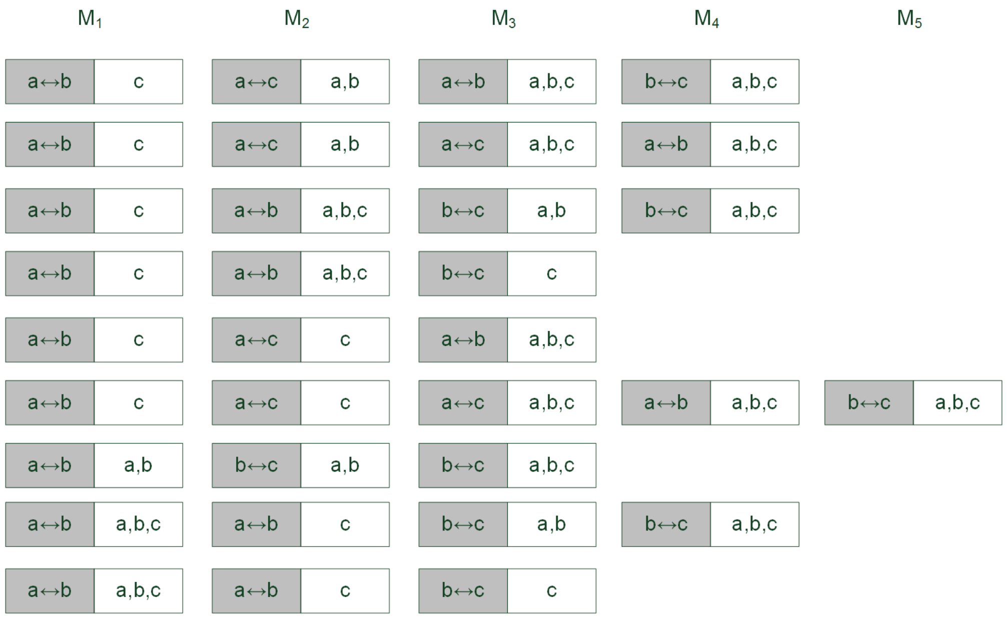

M1
M2
M3
M4
M5
a↔b c | a↔c a,b | a↔b a,b,c | b↔c a,b,c
a↔b c | a↔c a,b | a↔c a,b,c | a↔b a,b,c
a↔b c | a↔b a,b,c | b↔c a,b | b↔c a,b,c
a↔b c | a↔b a,b,c | b↔c c
a↔b c | a↔c c | a↔b a,b,c
a↔b c | a↔c c | a↔c a,b,c | a↔b a,b,c | b↔c a,b,c
a↔b a,b | b↔c a,b | b↔c a,b,c
a↔b a,b,c | a↔b c | b↔c a,b | b↔c a,b,c
a↔b a,b,c | a↔b c | b↔c c

FIGURE 5.17: Equivalent sets of markers.

5.2 Clarridge and Salomaa Algorithm

From the previous section we are able to identify two main problems with Kari's scheme: the exponential growth of the public key with the size of the neighborhood and how to efficiently generate the sets of markers.

In a practical implementation of the system, the neighborhood size of the composition should be kept as small as possible without, of course, compromising its security. We saw that it is not enough for the markers in the set to be reversible; they should also be able to generate a valid (deterministic) public key, without increasing the size of the neighborhood. In our previous example (see section 5.1.2, pp. 58), we solved this problem by considering all possible actions that each previous marker in the set had in the neighborhood. By doing so, we were able to specifically design the following marker in the set while keeping the neighborhood size constant. The algorithm presented by Clarridge and Salomaa [CS09], is based precisely on this idea. However they diverge from Kari's proposal, by imposing two severe limitations: all markers in the composition have the same neighborhood as the final result of their composition and all patterns have the same shape. Nevertheless, these simple limitations allows the creation of a rather simple and efficient implementation.

5.2.1 Definitions

We define *Fixed-Domain Marker Cellular Automaton* (FDM CA) by the 5-tuple $(d, S, N,$ $\pi, A)$, where d denotes the dimension, S the state set, N the neighborhood vector size k with $N = (n_1, n_2, ..., n_k)$ and $n_i \in \mathbb{Z}^d$ for $i = 1, 2, ..., k$, π which represents the update rule $\pi : S^k \to S$ and finally A as the acting set, defined by $A \subseteq S^k$ with entries corresponding to the positions defined by N. In a FDM CA, all patterns in $\mathcal{P}$ have the same shape, which is specified by the marker's neighborhood, i.e., $\forall Q, R \in \mathcal{P} : X_Q = X_R = N$. Hence, a FDM CA is a marker where for all $P \in \mathcal{P}$,

$$P : N \to S$$

5.2.2 Neighborhood Size

Let M be an arbitrary automaton with $d \geq 1$, state set S, neighborhood $N = (n_1, n_2, ...n_k)$ with $n_i \in \mathbb{Z}^d$ and update function $\pi : S^k \to S$. Let s_{n_i} represent the state of the cell at n_i. We define the state configuration of the neighborhood of cell x, $S_N(x)$, as the current states of the neighbors of x, and we represent it by the tuple:

$$S_N(x) = (s_{n_1}, s_{n_2}, ..., s_{n_k}), \quad s_{n_i} \in S \ , \quad for\ i = 1, ..., k$$

with s_{n_i}, representing the state of the neighboring cell n_i.

We also define the set of all possible state configurations of the neighborhood of cell x with neighborhood N by:

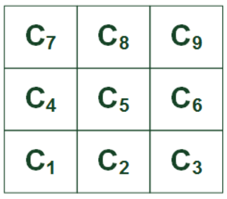
$$\mathcal{S}_N(x) = \{(s_{n_1}, s_{n_2}, ..., s_{n_k}) \mid s_{n_i} \in S \ , \quad for\ i = 1, ..., k\}$$

Also, let the set of all possible state configurations of the second order neighborhoods of a cell x or the state configurations of the neighborhood of the neighbors of x, with the states of x's neighbours given by $s = (s_{n_1}, s_{n_2}, ..., s_{n_k}) \in \mathcal{S}_N(x)$, be represented by the matrix:

$$\overline{\mathcal{S}}_N(s) = \left\{ \begin{bmatrix} s_{n_1+n_1} & s_{n_1+n_2} & \cdots & s_{n_1+n_k} \\ s_{n_2+n_1} & s_{n_2+n_2} & \cdots & s_{n_2+n_k} \\ & \vdots & & \\ s_{n_k+n_1} & s_{n_k+n_2} & \cdots & s_{n_k+n_k} \end{bmatrix} \in S^{k\times k} \mid \forall n \in N_M \right\}$$

where each row represents the state configuration of the neighborhood of each neighbor of x.

Take, for instance, a two-dimensional CA, M, with a neighborhood $N = \{(0,0), (1,0), (0,1), (1,1)\}$, state set S, with $s_{c_i} \in S$ representing the state of cell c_i and a cellular grid represented by the 3×3 square:

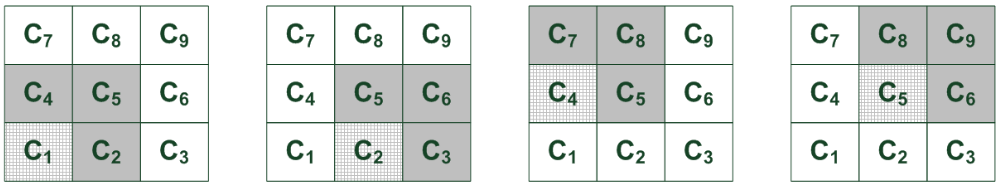


The state configuration of the neighborhood of cell c_1, is then represented by the tuple $S_N(c_1) = (s_{c_1}, s_{c_2}, s_{c_4}, s_{c_5})$.

If the state set is $S = \{a, b, c\}$, the set of all possible states that the neighborhood N of cell c_1 is:

$$\mathcal{S}_N(c_1) = \{\, (a,a,a,a), (a,a,a,b), (a,a,b,a), ..., (c,c,c,c)\,\}$$

Let $s = (s_{c_1}, s_{c_2}, s_{c_4}, s_{c_5})$, $s_i \in S_N(c_1)$. Considering that each cell can be part of its own neighborhood, the second order neighborhood of cell c_1 are the following shaded cells:



Thus we represent the set of all possible state configurations of the second order neighborhoods of cell c_1 with a state configuration of the neighborhood s, by:

$$\overline{\mathcal{S}}_N(s) = \left\{ \begin{bmatrix} s_{c_1} & s_{c_2} & s_{c_4} & s_{c_5} \\ s_{c_2} & s_{c_3} & s_{c_5} & s_{c_6} \\ s_{c_4} & s_{c_5} & s_{c_7} & s_{c_8} \\ s_{c_5} & s_{c_6} & s_{c_8} & s_{c_9} \end{bmatrix} \in S^{4\times 4} \right\}$$

The importance of the second order neighborhood of a cell, lies on the fact that it is this neighborhood that determines the next state configuration of the neighborhood of a cell and therefore if the state of a cell is changed or not by the marker. Let h_M represent the function that maps the state configuration of the second order neighborhoods to the next state configuration of the neighborhood of cell x:

$$h_M = \overline{\mathcal{S}}_N(s) \to \mathcal{S}_N(x)$$

We can then describe the set of all possible next state configurations of the neighborhood of cell x by:

$$\textit{next\_neighborhood}_M(s) = \{h_M(r) | r \in \mathcal{S}_N(s)\}$$

Now, let M be an arbitrary CA (d, S, N, π_M), with $d \geq 1$ and state set S. Let also C be a FDM CA defined by (d, S, N, A_C, π_C) (note that both have the same neighborhood). The composition $C \circ M$ has neighborhood N, if and only if for a cell x, $\forall s \in S_{N_M}(x) : \pi_C(h_M(s)) \neq h_M(s)$ then either $\textit{next\_neighborhood}_M(s) \subseteq A_C$ or $\textit{next\_neighborhood}_M(s) \cap A_C = \emptyset$.

By satisfying this condition, the neighborhood size of a composition remains the same, or in other words, we can determine the effect of the application of the composition $C \circ M$ on the state of a cell x, by just knowing the states of its neighbors before the application of the composition. If $\pi_C(h_M(s)) = h_M(s)$, then C always maps $h_M(s)$ to itself, and therefore, the neighborhood of the composition does not changes. If $\pi_C(h_M(s)) \neq h_M(s)$ and $\textit{next\_neighborhood}_M(s) \subseteq A_C$, then marker C knows from s if it acts or not, independently to which state configuration of the neighborhood s was mapped to by M.

In this case, π_C changes $h_M(s)$. If $\pi_C(h_M(s)) \neq h_M(s)$ and $next\_neighborhood_M(s) \cap A_C = \emptyset$, the same happens, but in this case π_C does not change $h_M(s)$ and therefore $\pi_C(h_M(s)) = h_M(s)$

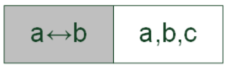
If the condition was untrue, then the composition $C \circ M$ would not have neighborhood N, because it would have dependencies outside the confines of the neighborhood.

5.2.3 Examples

One-Dimensional Our goal will be to create a set of markers, $\mathcal{M}$, composed by three one-dimensional FDM CA, with the state set $S = \{a, b, c, d\}$ and a neighborhood $N = \{(1)\}$ (right neighbor).

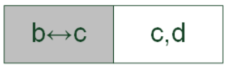
With the state set S, we can have nine permutations of S: $(a\ b)$, $(a\ c)$, $(a\ d)$, $(b\ c)$, $(b\ d)$, $(c\ d)$, $(a\ b)(c\ d)$, $(a\ c)(b\ d)$ and $(a\ d)(b\ c)$.

Starting with the first marker, M_1, lets arbitrarily chose $\pi_{M_1} = (a\ b)$ as its update function. Since M_1 is the first marker in the set, all we need to do, is to guarantee that it is self-invertible. So, lets chose an acting set with $\{a, b, c\}$:

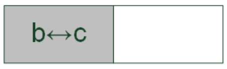


For our next marker, M_2, we select again any permutation in S: $\pi_{M_2} = (b\ c)$. To preserve the neighborhood size of the composition, the acting set of M_2 must obey one of the two conditions: either $next\_neighborhood_{M_1}(s) \subseteq A_{M_2}$ or $next\_neighborhood_{M_1}(s) \cap A_{M_2} = \emptyset$.

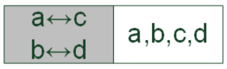
From M_1, we know that the set $next\_neighborhood_{M_1}(s)$ is $\{a, b\}$, i.e., the set of states that π_{M_1} can change. Now, if we chose the second condition, then A_{M_2} can not have any element present in $next\_neighborhood_{M_1}(s)$. Therefore, it can have have 'c', 'd' or both, or even none of them (empty acting set). Choosing an acting set with 'c' and 'd', M_2 would then be:



However, this marker it is not reversible. In order to be reversible, the acting set must contain, at least, all states present in $S_\pi^{M_2} = \{b, c\}$, or none of them. Then clearly we cannot add 'b', because that would break the above condition. Therefore we end up with three choices: we either chose a pattern set with only the state 'd', or an empty set or we can try to fulfill the other condition instead ($next\_neighborhood_{M_1}(s) \subseteq A_{M_2}$). For this markers, let us chose an empty acting set which implies that π is applied to every cell, regardless of its neighbors:



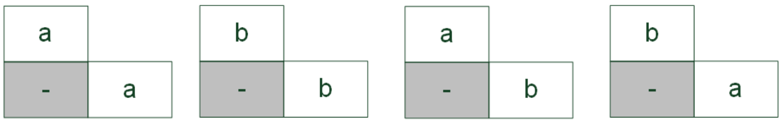
For M_3, we chose $\pi_{M_3} = (a\ c)\ (b\ d)$ and the condition $next\_neighborhood_{M_2 \circ M_1}(s) \subseteq A_{M_3}$. From M_1 and M_2, we have $next\_neighborhood_{M_2 \circ M_1}(s) = \{a, b, c\}$. Therefore, A_{M_3} must contain, at least, $\{a, b, c\}$. Thus, we can chose either $\{a, b, c\}$ or $\{a, b, c, d\}$. Let us chose the latter one, and thus M_3 becomes:



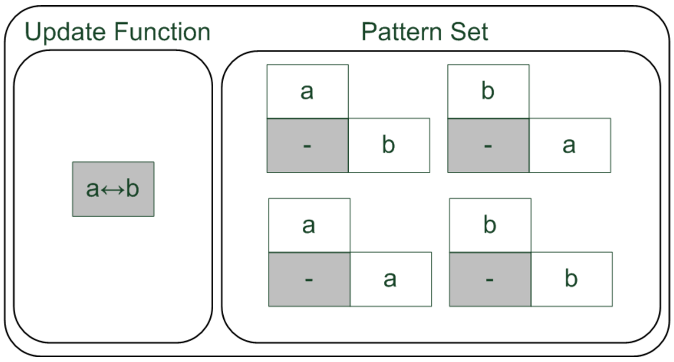
This acting set fulfills the reversibility condition because $S_\pi^{M_3} \subseteq A_{M_3}$, where $S_\pi^{M_3} = \{a, b, c, d\}$.

Two-Dimensional Once again let us create a set of three Two-dimensional FDM CAs, with the alphabet $S = \{a, b, c, d\}$, but this time with a neighborhood $N = \{(0, 1), (1, 0)\}$ (right and top neighbors). With this alphabet we have once again nine possible permutations: $(a\ b)$, $(a\ c)$, $(a\ d)$, $(b\ c)$, $(b\ d)$, $(c\ d)$, $(a\ b)(c\ d)$, $(a\ c)(b\ d)$ and $(a\ d)(b\ c)$.

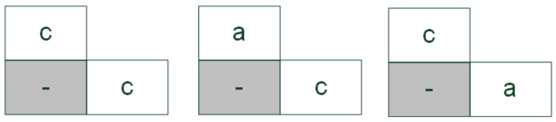
For the first marker, M_1, lets arbitrarily chose $\pi_{M_1} = (a\ b)$, and the following acting set A_{M_1}, making it self-invertible:



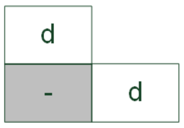
Our marker then become:



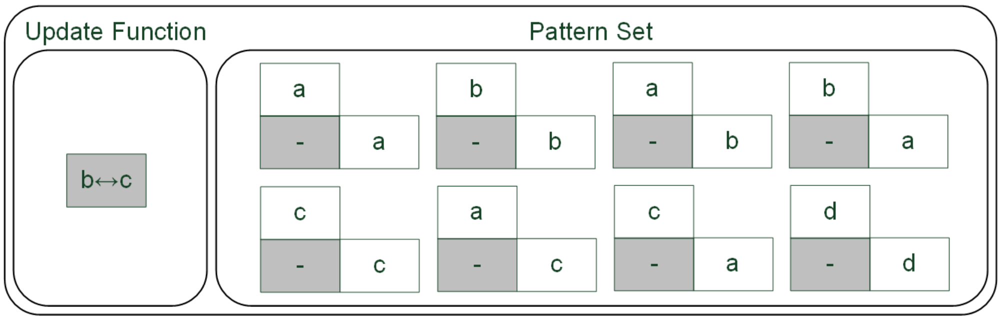
For the second marker, M_2, lets select $\pi_{M_2} = (b\ c)$. Once again, in order to preserve the neighborhood size of the composition, the acting set of M_2 must obey one of the two conditions: either $next\_neighborhood_{M_1}(s) \subseteq A_{M_2}$ or $next\_neighborhood_{M_1}(s) \cap A_{M_2} = \emptyset$. Let us chose the first condition. Since M_1 is a self-invertible marker, we know that $next\_neighborhood_{M_1}(s)$ is already contained in the acting set of M_1. Therefore, $A_{M_1} \subseteq A_{M_2}$. Since $b \in S_\pi^{M_2}$ and considering the reversibility conditions, we must add the following patterns to A_{M_2}:



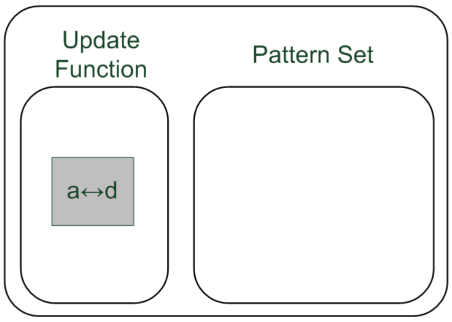
Let us also add an extra pattern that does not affect the marker's reversibility:



And with these 8 patterns, we have M_2 completely defined:



For our final marker, we chose $\pi_{M_3} = (a\ d)$ and an empty acting set, automatically making the marker self-invertible:



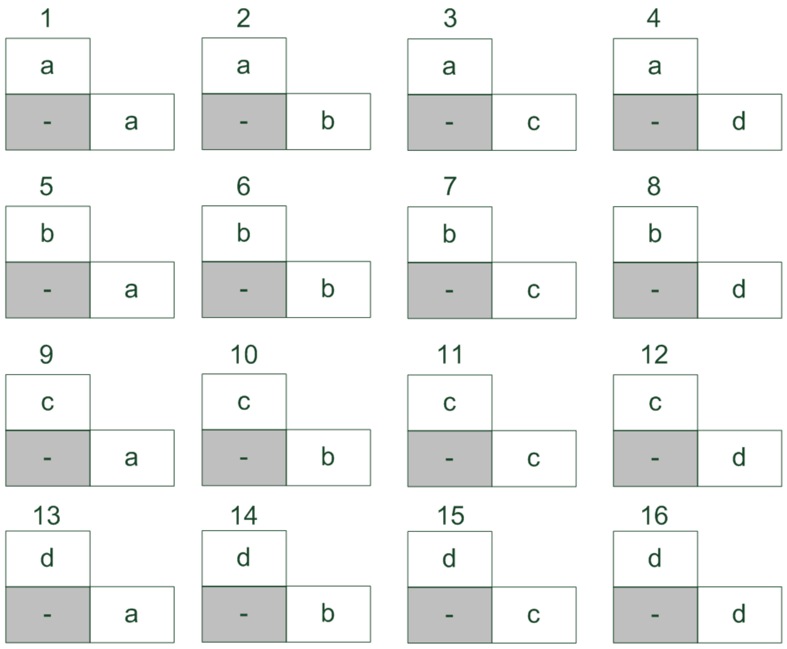
5.2.4 Algorithm and Security Analysis

Algorithm description The goal is to generate a set of n FDM CA markers. The inputs are: space set – S, neighborhood – N, size of the set $\mathcal{M}$ – n, probabilities constants – p and q, both taking values in $[0, 1]$.

Let T represent the set of all states involved in the update functions of all markers processed so far. Thus, if $\mathcal{M} = \{M_1, M_2, M_3\}$ and $\pi_{M_1} = (a\ b)$, $\pi_{M_2} = (b\ c)$, $\pi_{M_3} = (d\ e)$, then, T just prior to the application of M_3, is $T = \{a, b, c\}$, resulted from $S_\pi^{M_1} \cup S_\pi^{M_2} = \{a, b\} \cup \{b, c\}$, where we consider the application order of the markers to be M_1 followed by M_2 followed by M_3. Initially, T should contain two random elements of S.

The following procedure is executed for each marker. Before any new marker is generated, a new element of $S \setminus T$ is added to T with probability p. The update function π is a random permutation of set T. The main part of the algorithm is the random generation of acting sets. This randomization is done with the help of a random generated binary string of length $|N|$, where each bit represents a neighbor cell. The algorithm considers as possible candidates, all neighborhood state configurations that are possible to form by populating N with S. For each candidate, it checks each cell of the configuration if it has a state currently in T. If true and there is a '1' in the corresponding position of the binary string, then the candidate is added to the acting set. If, on the other, the candidate has no states present in T, then it is added to the acting set with probability q. The complexity of the algorithm is $O(n\,|N|\,|S|^{|N|})$.

Consider the following example, where currently the values of the algorithm are: $S = \{a, b, c, d\}$, $T = \{c, d\}$, $p = q = 0.5$. For this example we set our marker to have a L shape described by $\{\,(1, 0), (0, 1)\,\}$ (right and top neighbors). Consider also that our binary string is '10', where '1' corresponds to the right cell and '0' to the top cell. The candidates are all combinations that are possible to form by populating N with S:



Let us start with the first candidate. Clearly, no state in both cells are in T, and therefore we can add this pattern to our pattern set with a probability 0.5. The same for the second candidate. As for the third candidate, the right cell is in state $c \in T$, and since its respective position in the binary string is '1', we automatically add this candidate to our pattern set. The same for the fourth candidate, but instead of 'c' we have 'd'. Passing to the ninth candidate, its top cell has state $c \in T$, but since its respective position in the binary string is '0', we do not add it to the pattern set. As for the last candidate, clearly, both cells have states in T. This candidate is automatically added to the pattern set for the same reason as the third and fourth candidates.

Dividing all 16 candidates into groups of acceptance, we have:

- Automatically added to the pattern set: 3, 4, 7, 8, 11, 12, 15, 16;
- Added with 0.5 probability: 1, 2, 5, 6;
- Not added: 9, 10, 13, 14.

As we mention in section 5.1.1, a fundamental property of self-invertible markers, is that π maps all patterns in $\mathcal{P}$ back into $\mathcal{P}$. Since the expansion of set T happens before the creation of the acting set and the selection of π is done over T, the reversibility of each marker is automatically guaranteed.

An implementation of this algorithm in python can be founded in appendix A. In Clarridge Master thesis [Cla09] we can find a more complete implementation, including all the necessary tools for the generation of public keys and to execute encryption/decryption of plaintexts.

Number of Markers From table 5.2, we can have an idea on the number of markers that the algorithm generates, depending on the number of states and neighborhood size.

$\|\mathbf{S}\|$	$\|\mathbf{N}\|$	*Markers*
3	1	72
3	2	144
3	3	288
3	4	576
3	5	1152
3	6	2304
4	1	864
4	2	7776
4	3	223776
5	1	8000
5	2	1072000

Table 5.2: Number of distinct markers generated by the algorithm.

As expected, the number of markers grows exponentially with both the size of S and N.

In table 5.3, we can observe the growth of the number of available sets with N and S, but with the update function of each marker limited to a single permutation between two states of T.

Algorithm Issues and Security Analysis Apart of some remarks on the security of their algorithm against brute force attacks, Clarridge and Salomaa [Cla09] do not perform any serious cryptanalysis, and therefore the security of their system is still greatly unknown. We can however reach some conclusions on same issues with the algorithm as presented, which can directly affect the security of this scheme.

First, the algorithm does not give any assurance that, at the end, the sets T and S are equal. If both sets are not equal, it means that the states in $S \setminus T$ remain untouched by

$\|\mathbf{S}\|$	$\|\mathbf{N}\|$	$\|\mathcal{M}\|$	$Markers$
3	1	1	24
3	1	2	240
3	1	3	1824
3	2	1	48
3	2	2	624
3	2	3	6240
4	1	1	96
4	1	2	3264
4	1	3	72956
4	2	1	864
4	2	2	86013
4	2	3	6.2×10^6

Table 5.3: Number of available sets of markers.

the markers, i.e., these states are preserved at the same position in both the plaintext and in the ciphertext. Theoretically, to achieve $T = S$, n should be at least $(|S| - 2)\, p^{-1}$.

There are also valid acting sets that are ignored by the algorithm. Take the candidates of the previous example. Considering the same parameters, there are quite a few valid acting sets that are completely ignored by the algorithm, for example, $\{3, 4\}$, $\{7, 8\}$, $\{3, 4, 7, 8\}$, $\{1, 2, 3, 4\}$, and so on. This happens because the algorithm automatically adds all candidates that fall under the first acceptance group, regardless of valid subsets. A solution to this problem could pass through the random selection of a group of k candidates for the acting set and complete it, so the pattern set fulfills all the necessary conditions. The only drawback would be the increase of the complexity of the algorithm.

Because there is still no studies on the relation between the security of this cryptosystem and the working parameters of the algorithm, such as, the size of the set of markers and the probabilities factors p and q, extreme care should be taken when selecting them. For now, both Kari and Clarridge recommend the use of large neighborhoods or alphabets and encryption blocks (CA size), in order to defend against brute force attacks. However, as we have seen, as the size of the alphabet increases, so does the size of the set of markers required to achieve $T = S$.

Chapter 6

Conclusions

In this thesis we gave a general introduction to cellular automata and their application to cryptography and we have seen that cellular automata are a valid, efficient and robust tool for cryptographic applications.

We presented and discussed Kari's public key cryptosystem based on cellular automata, followed by a suggested implementation of this cryptosystem done by Clarridge. Despite the fact that the algorithm presented by Clarridge. is a great leap forward, it is not without limitations. They made the technical assumption that both the markers and their composition must have the same neighborhood size. Furthermore, as we have demonstrated, this algorithm operates on a restricted key space since their algorithm ignores several markers and therefore viable keys.

Regardless of its simplicity and potential, the security of the system is still greatly unknown. There are no clear methods for establishing secure operating parameters, such as key sizes, number of markers and required number of iterations. Also, considering the entire key space, the problem on how to efficiently generate the set of markers to generate viable keys is still an open problem. Likewise, the degrees of diffusion and confusion achieved by this system are still unknown.

In the future, we hope to continue the work of Kari and Clarridge, by not only performing a detailed cryptanalysis of their work, including a study of the avalanche criteria of the system, but also to implement Kari's idea without any limitation.

Appendix A

Clarridge and Salomaa Algorithm Implementation

A.1 Introduction

This thesis includes a python implementation of the Clarridge and Salomaa algorithm. The objective of the code is to generate a set of markers according given a neighborhood, set of states and two probabilities constants.

A different python implementation of the algorithm, including all the necessary tools to execute encryption/decryption of plaintexts and ciphertexts, can be found in the Clarridge master thesis [Cla09].

A.2 Documentation

class `cs.`**CSMarker**

Data structure representing a Clarridge and Salomaa marker

actingset = None
list of tuples, with each tuple representing a pattern

neighborhood = None
list of tuples with coordinates [(y,x)]

permutation = None
tuple representing the update function

`cs.`**getIntersectionComplement**(*l1*, *l2*)

Returns the difference of two lists: l1-(l1-l2).

`cs.`**getMarkers**(*states*, *neighborhood*, *number_of_markers*, *p*, *q*)

Implementation of the Clarridge and Salomaa algorithm.

Parameters

- **states** (*list*) – state set.
- **neighborhood** (*list of tuples*) – list with neighbors coordinates.
- **number_of_markers** (*int*) – size of the set of markers.
- **p** (*float*) – probability constant [0,1].
- **q** (*float*) – probability constant [0,1].

Returns list of CSMarkers

`cs.`**getRandomElement**(*lst*)

Returns a random element of a list

A.3 Prerequisites

Other than the default python libraries, there are no dependencies to execute the code.

A.4 Source Code

```
import random
import itertools

class CSMarker:
    """
        Data structure representing a Clarridge and Salomaa marker
    """

    def __init__(self):
        #: list of tuples with coordinates [(y,x)]
        self.neighborhood = []
        #: list of tuples, with each tuple representing a pattern
        self.actingset = []
        #: tuple representing the update function
        self.permutation = ()


def getRandomElement(lst):
    """ Returns a random element of a list """
    return lst[random.randint(0,len(lst)-1)]


def getIntersectionComplement(l1,l2):
    """ Returns the difference of two lists: l1-(l1-l2). """
    intersection = [val for val in l1 if val in l2]
    temp = list(l1)
    for i in intersection:
        temp.remove(i)
    return temp


def getMarkers(states,
               neighborhood,
               number_of_markers,
               p,
               q):
    """
    Implementation of the Clarridge and Salomaa algorithm.

    :param states: state set.
    :type states: list
    :param neighborhood: list with neighbors coordinates.
    :type neighborhood: list of tuples
    :param number_of_markers: size of the set of markers.
    :type number_of_markers: int
    :param p: probability constant [0,1].
    :type p: float
    :param q: probability constant [0,1].
    :type q: float
    :returns: list of CSMarkers

    """

    markers = []
    T = []

    all_possible_neighborhoods = list(itertools.product(states,repeat=
                                len(neighborhood)))

    T.append(getRandomElement(states))
    T.append(getRandomElement(getIntersectionComplement(states,T)))

    for i in range(number_of_markers):
        m = CSMarker()
        m.neighborhood = neighborhood


        # permutation
        if random.random() < p and not set(T) == set(states):
            T.append(getRandomElement(
                getIntersectionComplement(states,T)))
        available_permuts = list(itertools.combinations(T,2))
        permutation = getRandomElement(available_permuts)
        m.permutation = permutation


        # acting set
        random_number = random.getrandbits(len(neighborhood))
        binary_string = ('{0:0' + str(len(neighborhood))
            + 'b}').format(random_number)

        for nb in all_possible_neighborhoods:
            no_nb_change = True

            for j in range(len(neighborhood)):
                if nb[j] in T:
                    no_nb_change = False
                    if binary_string[j] == '1':
                        m.actingset.append(nb)
                        break

            if no_nb_change and random.random() < q:
                m.actingset.append(nb)

        m.actingset = sorted(m.actingset)
        markers.append(m)
    return markers
```

A.5 Description

The usage of the code is straightforward. If, for instance, we want to generate a set of two markers with a two-dimensional neighborhood (top and right cells) and a state set $S = \{a, b, c\}$ we have:

```
M = getMarkers(['a','b','c'],[(1,0),(0,1)],2,0.5,0.5)
```

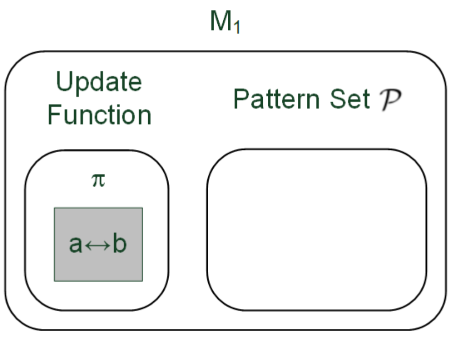
where both probabilities constants p and q have the same value 0.5, returning a list containing the two desired markers (CSMarkers).

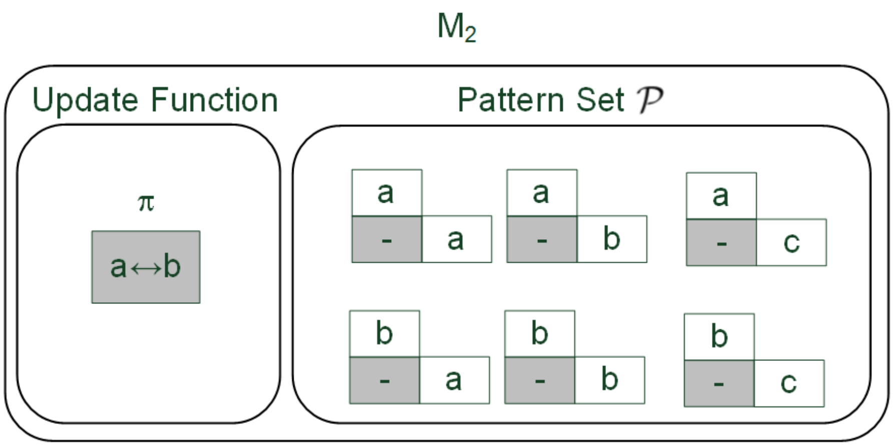
The generated markers from the above function could be the following:

```
M1.neighborhood = [(0,1),(1,0)]
M1.actingset = []
M1.permutation = ('a','b')

M2.neighborhood = [(0,1),(1,0)]
M2.actingset = [('a', 'a'), ('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'b'),
                ('b', 'c')]
M2.permutation = ('a','b')
```

With their corresponding visual representation as:




M2
Update Function
π
a↔b
Pattern Set 𝒫
a
-
a
a
-
b
a
-
c
b
-
a
b
-
b
b
-
c

Bibliography

[AEdR08] G. Alvarez, L.H. Encinas, and A.M. del Rey. A multisecret sharing scheme for color images based on cellular automata. *Information Sciences*, 178(22):4382–4395, November 2008.

[AIB09] P. Anghelescu, S. Ionita, and I. Bostan. Design of programmable cellular automata based cipher scheme. In *Proceedings of NaBIC 2009*, pages 187–192, 2009.

[AIS07] P. Anghelescu, S. Ionita, and E. Sofron. Block encryption using hybrid additive cellular automata. In *Proceedings of HIS 2007*, pages 132–137, Washington, DC, USA, 2007. IEEE Computer Society.

[ALI$^+$13] A.A. Abdo, S. Lian, I.A. Ismail, M. Amin, and H. Diab. A cryptosystem based on elementary cellular automata. *Communications in Nonlinear Science and Numerical Simulation*, 18(1):136–147, 2013.

[Ang11] P. Anghelescu. Encryption algorithm using programmable cellular automata. In *Proceedings of WorldCIS 2011*, pages 233–239, 2011.

[AP72] S. Amoroso and Y.N. Patt. Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. *Journal of Computer and System Sciences*, 6(5):448–464, October 1972.

[Atr65] A.J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers*, EC-14(3):394–399, June 1965.

[Bao03] F. Bao. Cryptanalysis of a new cellular automata cryptosystem. In Rei Safavi-Naini and Jennifer Seberry, editors, *Proceedings of ACISP 2003*, volume 2727 of *LNCS*, pages 416–427. Springer, Berlin, Heidelberg, 2003.

[Bat07] M. Batty. *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals.* The MIT Press, 2007.

[BBM97] M. Blackburn, S.R. Blackburn, and S. Murphy. Comments on "Theory and applications of cellular automata in cryptography", 1997.

[Bla79] G.R. Blakley. Safeguarding cryptographic keys. In *Proceedings of AFIPS 1979*, volume 48, pages 313–317, Jun 1979.

[BMS01] S. Bandini, G. Mauri, and R. Serra. Cellular automata: from a theoretical parallel computational model to its application to complex systems. *Parallel Computing*, 27(5):539–553, 2001. Cellular automata: from modeling to applications.

[BS91] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In A.J. Menezes and S.A. Vanstone, editors, *Proceedings of CRYPT0 1990*, volume 537 of *LNCS*, pages 2–21. Springer Berlin Heidelberg, 1991.

[CDL97] B. Chopard, A. Dupuis, and P. Luthi. A cellular automata model for urban traffic and its application to the city of Geneva. In *Proceedings of Traffic and Granular*, pages 154–168. Springer, 1997.

[CDM] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Proceedings of EUROCRYPT 2000.*

[CJS05] J.A. Clark, J.L. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. *New Generation Computing*, 23(3):219–231, 2005.

[Cla09] A. Clarridge. Cellular Automata: Algorithms and Applications. Master's thesis, School of Computing - Queen's University, Kingston, Ontario, Canada, 2009.

[CP91] P. Camion and J. Patarin. The Knapsack hash function proposed at Crypto89 can be broken. In DonaldW. Davies, editor, *Proceedings of EUROCRYPT 1991*, volume 547 of *LNCS*, pages 39–53. Springer Berlin Heidelberg, 1991.

[CS92] F. Celada and P.E. Seiden. A computer model of cellular interactions in the immune system. *Immunology Today*, 13(2):56–62, 1992.

[CS09] A. Clarridge and K. Salomaa. A cryptosystem based on the composition of reversible cellular automata. In A. Dediu, A. Ionescu, and C. Martín-Vide, editors, *Language and Automata Theory and Applications*, volume 5457 of *LNCS*, pages 314–325. Springer Berlin Heidelberg, 2009.

[CSS$^+$05] J.C.H. Castro, J.M. Sierra, A. Seznec, A. Izquierdo, and A. Ribagorda. The strict avalanche criterion randomness test. *Mathematics and Computers in Simulation*, 68(1):1–7, 2005.

[Dam89] I.B. Damgård. A design principle for hash functions. In *Proceedings of CRYPTO 1989*, pages 416–427, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[DDRR04] A. Doeschl, M. Davison, H. Rasmussen, and G. Reid. Assessing cellular automata based models using partial differential equations. *Mathematical and Computer Modelling*, 40(9-10):977–994, November 2004.

[DF92] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In Joan Feigenbaum, editor, *Proceedings of CRYPTO 1991*, volume 576 of *LNCS*, pages 457–469. Springer Berlin Heidelberg, 1992.

[DGV93] J. Daemen, R. Govaerts, and J. Vandewalle. A framework for the design of one-way hash functions including cryptanalysis of Damgard's one-way function based on a cellular automaton. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Proceedings of ASIACRYPT 1991*, volume 739 of *LNCS*, pages 82–96. Springer Berlin Heidelberg, 1993.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976.

[DSS09] S. Das, M. Saha, and B.K. Sikdar. A cellular automata based model for traffic in congested city. In *Proceedings of SMC 2009*, pages 2397–2402, Oct 2009.

[EA10] Z. Eslami and J.Z. Ahmadabadi. A verifiable multi-secret sharing scheme based on cellular automata. *Information Sciences*, 180(15):2889–2894, 2010.

[Elg85] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, Jul 1985.

[ERA10] Z. Eslami, S.H. Razzaghi, and J.Z. Ahmadabadi. Secret image sharing based on cellular automata and steganography. *Pattern Recognition*, 43(1):397–404, 2010.

[ES97] J. Esser and M. Schreckenberg. Microscopic simulation of urban traffic based on cellular automata. *International Journal of Modern Physics C*, 08(05):1025–1036, 1997.

[Fis65] P.C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12(3):388–394, July 1965.

[For] R. Forré. The strict avalanche criterion: Spectral properties of boolean functions and an extended definition. In *Proceedings of CRYPTO 1988*.

[Gar70] M. Gardner. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, pages 120–123, Oct 1970.

[Gir88] M. Girault. Hash-functions using modulo-N operations. In David Chaum and WynL. Price, editors, *Proceedings of EUROCRYPT 1987*, volume 304 of *LNCS*, pages 217–226. Springer Berlin Heidelberg, 1988.

[GPSW] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of CCS 2006*.

[Gua87] P. Guan. Cellular automaton public-key cryptosystem. In *Complex Systems*, volume 1, pages 51–57, 1987.

[Gut98] Peter Gutmann. Software Generation of Practically Strong Random Numbers. In *Proceedings of SSYM 1998*, volume 7, pages 19–19, Berkeley, CA, USA, 1998. USENIX Association.

[GZ03] S.i Guan and S. Zhang. An evolutionary approach to the design of controllable cellular automata structure for random number generation. *IEEE Transactions on Evolutionary Computation*, 7(1):23–36, 2003.

[Hed69] G.A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Mathematical systems theory*, 3(4):320–375, 1969.

[HMC89] P.D. Hortensius, R.D. McLeod, and H.C. Card. Parallel random number generation for VLSI systems using cellular automata. *IEEE Transactions on Computers*, 38(10):1466–1473, 1989.

[HMP$^+$89] P.D. Hortensius, R.D. McLeod, W. Pries, D.M. Miller, and H.C. Card. Cellular automata-based pseudorandom number generators for built-in self-test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):842–859, 1989.

[HR03] R. Hu and X. Ruan. Differential equation and cellular automata model. In *Proceedings of 2003 IEEE International Conference on Robotics, Intelligent Systems, and Signal Processing*, volume 2, pages 1047–1051, Oct 2003.

[IY88] K. Culik II and S. Yu. Undecidability of CA classification schemes. *Complex Systems*, 2(2):177–190, Apr 1988.

[Jeo13] J.C. Jeon. One-way hash function based on cellular automata. In Kuinam J. Kim and Kyung-Yong Chung, editors, *IT Convergence and Security 2012*, volume 215 of *Lecture Notes in Electrical Engineering*, pages 21–28. Springer Netherlands, 2013.

[JMC06] P. Joshi, D. Mukhopadhyay, and R.D. Chowdhury. Design and analysis of a robust and efficient block cipher using cellular automata. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 2, pages 67–71, April 2006.

[JNKS07] B. Jafarpour, A. Nematzadeh, V. Kazempour, and B. Sadeghian. A cheating model for cellular automata-based secret sharing schemes. *Proceedings of WASET*, 25:306–310, 2007.

[JSHJ98] M. Jakobsson, E. Shriver, B.K. Hillyer, and A. Juels. A Practical Secure Physical Random Bit Generator. In *Proceedings of CCS 1998*, pages 103–111, New York, NY, USA, 1998. ACM.

[KA97] C.K. Koc and A.M. Apohan. Inversion of cellular automata iterations. *IEE Proceedings of Computers and Digital Techniques*, 144(5):279–284, 1997.

[Kah96] D. Kahn. *The codebreakers : the story of secret writing.* Scribner, New York, 1996.

[Kar90] J. Kari. Reversibility of 2D cellular automata is undecidable. *Physica D: Nonlinear Phenomena*, 45(1-3):379–385, 1990.

[Kar92] J. Kari. Cryptosystems based on reversible cellular automata. Technical report, 1992.

[Kar94] J. Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1):149–182, 1994.

[KMI91] K. Kim, T. Matsumoto, and H. Imai. A recursive construction method of s-boxes satisfying strict avalanche criterion. In AlfredJ. Menezes and ScottA. Vanstone, editors, *Proceedings of CRYPTO 1990*, volume 537 of *LNCS*, pages 565–574. Springer Berlin Heidelberg, 1991.

[Knu97] R.E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3st edition, 1997.

[K9̊7] P. Kůrka. Languages, equicontinuity and attractors in cellular automata. *Ergodic Theory and Dynamical Systems*, 17:417–433, 4 1997.

[K0̊3] P. Kůrka. *Topological and symbolic dynamics.* Collection SMF. Société Mathématique de France, 2003.

[KY04] H.S. Kim and K.Y. Yoo. Cellular automata based multiplier for public-key cryptosystem. In Dieter Hutter, Gnter Mller, Werner Stephan, and Markus Ullmann, editors, *Security in Pervasive Computing*, volume 2802 of *LNCS*, pages 227–236. Springer Berlin Heidelberg, 2004.

[Lai92] X. Lai. On the design and security of block ciphers. 1992.

[Lan05] T. Lange. Koblitz curve cryptosystems. *Finite fields and their applications*, 11(2):200–229, 2005.

[LP90] W. Li and N. Packard. The structure of the elementary cellular automata rule space. *Complex Systems*, 4:281–297, 1990.

[LP03] Y. Liu and S.R. Phinn. Modelling urban development with cellular automata incorporating fuzzy-set approaches. *Computers, Environment and Urban Systems*, 27(6):637–658, 2003. Part Special Issue: Urban Data Management Society.

[LPL90] W. Li, N.H. Packard, and C.G. Langton. Transition phenomena in cellular automata rule space. *Physica D: Nonlinear Phenomena*, 45(1-3):77–94, 1990.

[LZ02] H. Li and C.N. Zhang. Efficient cellular automata based versatile multiplier for GF(2m). *Journal of Information Science and Engineering*, 18:18–479, 2002.

[Mat94] M. Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Proceedings of EUROCRYPT 1993*, volume 765 of *LNCS*, pages 386–397. Springer Berlin Heidelberg, 1994.

[MBC$^+$99] W. Millan, L. Burnett, G. Carter, A. Clark, and E. Dawson. Evolutionary heuristics for finding cryptographically strong S-boxes. In Vijay Varadharajan and Yi Mu, editors, *Proceedings of ICICS 1999*, volume 1726 of *LNCS*, pages 263–274. Springer Berlin Heidelberg, 1999.

[MEdR03] G.A. Marañón, L.H. Encinas, and A.M. del Rey. Sharing secret color images using cellular automata with memory. *CoRR*, cs.CR/0312034, 2003.

[MGS$^+$02] P. Maji, N. Ganguly, S. Saha, A. Roy, and P.P Chaudhuri. Cellular automata machine for pattern recognition. In Stefania Bandini, Bastien Chopard, and Marco Tomassini, editors, *Proceedings of ACRI 2002*, volume 2493 of *LNCS*, pages 270–281. Springer Berlin Heidelberg, 2002.

[MH78] R. Merkle and M.E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, Sep 1978.

[MH89] K. Morita and M. Harao. Computation universality of one-dimensional reversible (injective) cellular automata. In *IEICE Transactions Japan E-72*, volume 8, pages 758–762, 1989.

[Mih] M.J. Mihaljević. Security examination of a cellular automata based pseudorandom bit generator using an algebraic replica approach. In *Proceedings of AAECC 2012*.

[MS91] W. Meieri and O. Staffelbach. Analysis of pseudo random sequences generated by cellular automata. In Donald W. Davies, editor, *Proceedings of EUROCRYPT 1991*, volume 547 of *LNCS*, pages 186–199. Springer Berlin Heidelberg, 1991.

[MSC01] W.A. Maniatty, B.K. Szymanski, and T. Caraco. Progress in computer research. chapter Parallel Computing with Generalized Cellular Automata, pages 51–75. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[MVO96] A. Menezes, A. Vanstone, and P. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[MZI98] M. Mihaljević, Y. Zheng, and H. Imai. A cellular automaton based fast one-way hash function suitable for hardware implementation. In Hideki Imai and Yuliang Zheng, editors, *Proceedings of PKC 1998*, volume 1431 of *LNCS*, pages 217–233. Springer Berlin Heidelberg, 1998.

[MZI99] M. Mihaljević, Y. Zheng, and H. Imai. A family of fast dedicated one-way hash functions based on linear cellular automata over GF(q), 1999.

[NBoS77] U.S. Department of Commerce National Bureau of Standards. *Data Encryption Standard*, FIPS-Pub.46., January 1977.

[NKC94] S. Nandi, B.K. Kar, and P.P. Chaudhuri. Theory and applications of cellular automata in cryptography. *IEEE Transactions on Computers*, 43(12):1346–1357, 1994.

[NM07] N. Nedjah and L.M. Mourelle. Designing substitution boxes for secure ciphers. *International Journal of Innovative Computing and Applications*, 1(1):86–91, April 2007.

[NW98] M. Naor and A. Wool. Access control and signatures via quorum secret sharing. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):909–922, Sep 1998.

[PDL$^+$79] K. Preston, M.J.B. Duff, S. Levialdi, P.E. Norgren, and J. Toriwaki. Basics of cellular logic with some applications in medical image processing. *Proceedings of IEEE*, 67(5):826–856, May 1979.

[PT01] Federal Information Processing and Announcing The. Announcing the Advanced Encryption Standard (AES), 2001.

[Rab] M.O. Rabin. Randomized byzantine generals. In *Proceedings of SFCS 1983*.

[REW$^+$04] C.F. Rubio, L.H. Encinas, S.H. White, A.M. del Rey, and G.R. Sánchez. The use of linear hybrid cellular automata as pseudo random bit generators in cryptography. *Neural, Parallel & Scientific Computations*, 12(2):175–192, 2004.

[Ric72] D. Richardson. Tessellations with local transformations. *Journal of Computer and System Sciences*, 6(5):373–388, 1972.

[Riv94] R.L. Rivest. The RC5 Encryption Algorithm. In *Proceedings of FSE 1994*, volume 1008 of *LNCS*, pages 86–96. Springer, 1994.

[Roc05] A. Rock. Pseudorandom number generators for cryptographic applications. Master's thesis, Naturwissenschachtlichen Fakultt der Paris-Lodron-Universitt Salzburg, 2005.

[RRSY] R.L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. The RC6 block cipher.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A Method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[SB04a] M. Seredynski and P. Bouvry. Block cipher based on reversible cellular automata. In *Proceedings of CEC 2004*, volume 2, pages 2138–2143, 2004.

[SB04b] M. Seredynski and P. Bouvry. Block encryption using reversible cellular automata. In PeterM.A. Sloot, Bastien Chopard, and AlfonsG. Hoekstra, editors, *Cellular Automata*, volume 3305 of *LNCS*, pages 785–792. Springer Berlin Heidelberg, 2004.

[SBZ04] F. Seredynski, P. Bouvry, and A.Y. Zomaya. Cellular automata computations and secret key cryptography. *Parallel Computing*, 30(5-6):753–766, 2004. Parallel and nature-inspired computational paradigms and applications.

[Sch94] B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In Ross Anderson, editor, *Fast Software Encryption*, volume 809 of *LNCS*, pages 191–204. Springer Berlin Heidelberg, 1994.

[Sch95] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 2st edition, 1995.

[Sch99] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In Michael Wiener, editor, *Proceedings of CRYPTO 1999*, volume 1666 of *LNCS*, pages 148–164. Springer Berlin Heidelberg, 1999.

[Sea71] M. Sears. The automorphisms of the shift dynamical system are relatively sparse. *Mathematical systems theory*, 5(3):228–231, 1971.

[SH97] N.J. Savill and P. Hogeweg. Modelling morphogenesis: from single cells to cawling slugs. *Journal of Theoretical Biology*, 184(3):229–235, 1997.

[Sha49] C.E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[SHH07] J. Sung, D. Hong, and S. Hong. Cryptanalysis of an involutional block cipher using cellular automata. *Information Processing Letters*, 104(5):183–185, 2007.

[Sin00] S. Singh. *The code book: the science of secrecy from ancient Egypt to quantum cryptography.* Anchor, reprint edition, Aug 2000.

[SS08] M. Szaban and F. Seredynski. Application of cellular automata to create S-box functions. In *Proceedings of IPDPS 2008*, pages 1–7, 2008.

[SS09] M. Szaban and F. Seredynski. Cellular automata-based S-boxes vs. DES S-boxes. In Victor Malyshkin, editor, *Parallel Computing Technologies*, volume 5698 of *LNCS*, pages 269–283. Springer Berlin Heidelberg, 2009.

[SS13] S. Sujata and S.S. Shefali. Cellular automata for crypt-stegonography. *International Journal of Advanced Technology & Engineering Research*, 3, 2013.

[SSB06] M. Szaban, F. Seredynsk, and P. Bouvry. Collective behavior of rules for cellular automata-based stream ciphers. In *Proceedings of CEC 2006*, pages 179–183, 2006.

[SSC$^+$02] S. Sen, C. Shaw, D. Chowdhuri, N. Ganguly, and P.P. Chaudhuri. Cellular cutomata based cryptosystem (CAC). In Robert Deng, Feng Bao, Jianying Zhou, and Sihan Qing, editors, *Proceedings of ICICS 2002*, volume 2513 of *LNCS*, pages 303–314. Springer Berlin Heidelberg, 2002.

[ST96] M. Sipper and M. Tomassini. Generating parallel random number generators by cellular programming. *International Journal of Modern Physics C*, 7:181–190, 1996.

[Sta11] W. Stallings. *Cryptography and Network Security: Principles and Practice.* Prentice Hall, New York, NY, USA, 5 edition, 2011.

[SU67] R.G. Schrandt and S.M. Ulam. On recursively defined geometrical objects and patterns of growth. Technical Report LA-3762, Los Alamos Scientific Laboratory, Los Alamos, NM, USA, 1967.

[SV86] M. Santha and U.V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, 1986.

[SZZ] J. Seberry, X.M. Zhang, and Y. Zheng. Systematic generation of cryptographically robust S-boxes. In *Proceedings of CCS 1993.*

[TM90] T. Toffoli and N.H. Margolus. Invertible cellular automata: A review. *Physica D: Nonlinear Phenomena*, 45(1-3):229–253, 1990.

[Tof77] T. Toffoli. Computation and construction universality of reversible cellular automata. *Journal of Computer and System Sciences*, 15(2):213–231, 1977.

[Tof84] T. Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics . *Physica D: Nonlinear Phenomena*, 10(1–2):117–127, 1984.

[TP01] M. Tomassini and M. Perrenoud. Cryptography with cellular automata. *Applied Soft Computing*, 1(2):151–160, 2001.

[TSP00] M. Tomassini, M. Sipper, and M. Perrenoud. On the generation of high-quality random numbers by two-dimensional cellular automata. *IEEE Transactions on Computers*, 49(10):1146–1151, Oct 2000.

[TSZP99] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud. Generating high-quality random numbers in parallel by cellular automata. *Future Generation Computer Systems*, 16(2-3):291–305, December 1999.

[TVB09] O.K. Tonguz, W. Viriyasitavat, and F. Bai. Modeling urban traffic: A cellular automata approach. *IEEE Communications Magazine*, 47(5):142–150, May 2009.

[Tvr00] M. Tvrdý. Book Review Wolfgang Alt, Andreas Deutsch, Graham Dunn, eds.: Dynamics of cell and tissue motion. Mathematics and Bioscience in Interaction. Birkhuser, Basel 1997, xvi+336 pages, ISBN 3-7643-5781-9, price DM 118,. *Applications of Mathematics*, 45(3):239–240, 2000.

[Ula60] S.M. Ulam. *A Collection of mathematical problems*, volume 8 of *Interscience Tracts in Pure and Applied Mathematics.* Interscience Publishers, New York, 1960.

[Ula62] S.M. Ulam. On some mathematical problems connected with patterns of growth of figures. *Applied Mathematics*, 14:215–224, 1962.

[Vic84] G.Y. Vichniac. Simulating physics with cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):96–116, 1984.

[vN51] J. von Neumann. The general and logical theory of automata. *Cerebral mechanisms in behavior; the Hixon Symposium*, pages 1–41, 1951.

[Wan11] X. Wang. A novel adaptive proactive secret sharing without a trusted party. *IACR Cryptology ePrint Archive*, 2011:241, 2011.

[WL05] R. Wang and M. Li. A realistic cellular automata model to simulate traffic flow at urban roundabouts. In Vaidy S. Sunderam, Geert Dick Albada, Peter M.A. Sloot, and Jack J. Dongarra, editors, *Proceedings of ICCS 2005*, volume 3515 of *LNCS*, pages 420–427. Springer Berlin Heidelberg, 2005.

[WM07] M.E. Whitman and H.J. Mattord. *Principles of Information Security.* Course Technology Press, Boston, MA, United States, 3rd edition, 2007.

[Wol84] S. Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96(1):15–57, 1984.

[Wol85] S. Wolfram. Cryptography with cellular automata. In *Proceedings of CRYPTO 1985*, volume 218 of *LNCS*, pages 429–432. Springer, 1985.

[Wol86] S. Wolfram. Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7(2):123–169, 1986.

[Wol02] S. Wolfram. *A new kind of science.* Wolfram Media Inc., Champaign, Ilinois, US, United States, 2002.

[WT86] A.F. Webster and S.E. Tavares. On the design of S-Boxes. pages 523–534. Springer-Verlag, 1986.

[YCXJL12] X. Yan, C. Cheng-Xun, and L. Jin-Long. Modeling and simulation for urban rail traffic problem based on cellular automata. *Communications in Theoretical Physics*, 58(6):847, 2012.