ROYAUME DU MAROC

Université Mohammed V - RABAT - جامعة محمد الخامس - الرباط -

Faculté des sciences

كلية العلوم

CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

N° d'ordre : 3398

THESE

En vue de l'obtention du : ***DOCTORAT***

Structure de Recherche : Intelligent Processing Systems & Security (IPSS)
Discipline : Informatique
Spécialité : Sécurité Informatique

Présentée et soutenue le 19/12/2020 par :

Fatima Ezzahra ZIANI

Design of Secure Hybrid Cellular Automata-Based Symmetric Cryptographic Systems

JURY

Pr. Youssef EL BENNANI	PES,	Faculté des Sciences, Université Mohammed V, Rabat	Président
Pr. Fouzia OMARY	PES,	Faculté des Sciences, Université Mohammed V, Rabat	Directrice de Thèse
Pr. Soumia ZITI	PH,	Faculté des Sciences, Université Mohammed V, Rabat	Rapporteur/Examinateur
Pr. Abdelaziz MOULOUDI	PES,	Faculté des Sciences, Université Ibn Tofail, Kénitra	Rapporteur/Examinateur
Pr. Zine El Abidine GUENNOUN	PES,	Faculté des Sciences, Université Mohammed V, Rabat	Rapporteur/Examinateur

Année Universitaire : 2019/2020

"I really think that if we change our own approach and thinking about what we have available to us, that is what will unlock our ability to truly excel in security. It's a perspectives exercise. What would it look like if abundance were the reality and not resource constraint?" – Greg York

Dedications

I dedicate this thesis to :

- To all my Big Family members! I can't cite all of them, the list is very long.
- To my Parents for all the sacrifices, patience and effort during the years of my academic career.
- To my sisters Hajar, Zoubida and Rafika who were always there, encouraged and supported me during my cursus and shared together wonderful moments. Thank you for all!
- To my best friends Bethaina, Narjiss and Ilham for being part of my life.
- To the IPSS research team, especially Hicham, Charifa, Bouchra and Anas for their help and collaborations.
- To all my friends and beloved ones.

Aknowledgement

First of all, I thank God for giving me the opportunity, the strength, motivation, and patience to prepare this thesis in the field of information security.

It was a pleasure for me to carry out the research work of this thesis under the direction and supervision of Pr Fouzia OMARY within the research team Intelligent Processing & Security of Systems (IPSS). The computing department of the Faculty of Sciences - University Mohammed V of Rabat (FSR-UMV). I wish to express my gratitude to Pr Fouzia OMARY for her valuable guidance and support throughout the thesis preparation period. I am grateful that I have gone through this experience.

My thanks and respects also go to Pr. Youssef EL BENNANI, PES at Mohammed V University in Rabat, Faculty of Sciences, for having accepted to be the Jury President of this thesis and to examine this work.

I am also grateful to Pr. Soumia ZITI, PH at Mohammed V University in Rabat, Faculty of Sciences, for having accepted to be one of the jury members, reporter and examiners of this thesis.

Further thanks to Pr. Zine El Abidine GUENNOUN, PES at Mohammed V University in Rabat, Faculty of Sciences, for accepting to examine this work and to be part of the jury members as a reporter and examiner.

I wish to express my gratitude to Pr. AbdelAziz MOULOUDI, PES at the University Ibn Tofail, Faculty of Sciences of Kenitra, for his interest in this work by accepting to be an examiner and jury member of this thesis.

I would also like to thank my dear family and friends, my colleagues and

anyone who contributed from near or far to achieve this work, my success is the result of all of us.

Abstract

The use of cryptographic systems and protocols became indispensable to guarantee the information security goals, which fall in the purpose of this thesis. New symmetric cryptographic systems are designed to ensure data confidentiality and integrity using cellular automata, which provide complex behavior and good properties from fast, simple, and parallel computations. At first a symmetric partition problem inspired encryption scheme , Partition Ciphering System (PCS), is proposed to make frequency cryptanalysis and brute force attacks challenging. Then the second contribution introduces a hybrid cellular automaton producing more confusion and diffusion, as well as better security against attacks. The third contribution aims to produce sequences with high quality of randomness through a family of pseudo-random number generators, CFA, based on cellular automata and other cryptographic primitives. The fourth contribution is a stream cipher involving cellular automata, with the objective of investigating different configurations to have better security level. The last two contributions fall in the primitives providing data integrity, namely hash functions. The fifth contribution, LCAHASH1.1, is a cellular automaton-based hash functions family. The last one makes use of multiple cellular automata in addition to some features to prevent known attacks.

Keywords: Information security, Symmetric Cryptography, Cellular Automata, Confidentiality, Integrity, Avalanche effect, Confusion, Diffusion, Pseudorandom Number Generator, Stream Cipher, Hash functions

Résumé

Le recours de systèmes et de protocoles cryptographiques est devenue indispensable pour garantir les objectifs de la sécurité informatique, qui correspondent à la motivation de cette thèse. De nouveaux systèmes cryptographiques symétriques sont conçus pour assurer la confidentialité et l'intégrité des données en utilisant les automates cellulaires, qui fournissent un comportement complexe et des propriétés satisfaisantes à partir de calculs rapides, simples et en parallèle. Dans un premier temps, un système de chiffrement symétrique inspiré par le problème de la partition, chiffrement par partition (PCS), est proposé pour rendre la cryptanalyse des fréquences et les attaques par force brute plus complexes. Ensuite, la deuxième contribution, CA-PCS utilise un automate cellulaire non-uniform produisant plus de confusion et de diffusion, ainsi qu'une meilleure sécurité contre les attaques. La troisième contribution vise à produire des séquences de haute qualité d'aléatoire grâce à une famille de générateurs de nombres pseudo-aléatoires, CFA, basée sur des automates cellulaires et d'autres primitives cryptographiques. La quatrième contribution est un chiffrement par flux impliquant des automates cellulaires, avec l'objectif d'étudier plusieurs configurations pour atteindre un meilleur niveau de sécurité. Les deux dernières contributions concernent les primitives assurant l'intégrité des données, à savoir les fonctions de hachage. La cinquième contribution, LCAHASH1.1, est une famille de fonctions de hachage basées sur des automates cellulaires. La dernière fait appel à plusieurs automates cellulaires en plus de certaines mécanismes pour prévenir les attaques connues.
Mots-clefs : Sécurité Informatique, Cryptographie Symètrique, Automate Cellulaire, Confidentialité, Integrité, Effet d'Avalanche, Confusion, Diffusion, Générateur Pseudo-Aléatoire, Chiffrement par flux, Fonction de Hachage.

Résumé détaillé

Le but de ce travail est de construire des cryptosystèmes basés sur les automates cellulaires pour assurer la confidentialité et l'integrité des données, qui font partie des objectifs principals de la sécurité de l'information. L'objectif de l'utilisation d'automates cellulaires dans la conception des différents systèmes cryptographiques est de tirer profit de leurs caractéristiques, à savoir la simplicité, les propriétés aléatoires, la facilité de l'implémentation logicielle et matérielle, le parallélisme, la vitesse de calcul, ainsi que les propriétés cryptographiques, qui changent de manière significative au fil du temps. Différents domaines de recherche (physique, biologie, chimie, cryptographie, etc.) utilisent les ACs. Dans ce mémoire, son utilisation dans la cryptographie est présentée dans les sections Related Work des chapitres 3,4,5 et 6.

Organisation du mémoire:

Cette thèse comprend une introduction générale, un chapitre sur les fondaments de la cryptographie symètrique et les automates cellulaires, et six contributions. Les contributions de 1 à 4 assurent la confidentialié, et les contributions 5 et 6 permettent de protéger l'integrité des données :

Contribution 1 : Système de chiffrement des partitions (PCS): Il s'agit d'un système de chiffrement symétrique développé pour améliorer l'extension binaire du SEC [1] contre l'analyse de fréquence, en faisant apparaître les blocs ou les caractères avec la même fréquence dans le chiffré. Le Problème de Piles égales(Equal Piles Problem EPP) a été notre source d'inspiration pour atteindre notre objectif, avec une adaptation de la définition du problème en fonction de nos besoins. Le chapitre 2 présente d'abord le système de chiffrement PCS, puis sa combinaison avec la version binaire du SEC, qui est un système de chiffrement symétrique conçu en introduisant un algorithme évolutionniste pour changer les positions des blocs dans le chiffré. La version binaire du SEC permet de permuter les listes des positions des blocs sans changer les listes des positions. Alors, pour que tous les blocs apparaissent avec la même fréquence, la proposition de PCS permet de définir en premier lieu le nombre d'apparitions par le calcul de la cardinalité idéale. Puis selon sa valeur, les blocs qui apparaissent moins sont ajoutés, et les blocs qui apparaissent plus sont enlevés d'une position aléatoirement. Ces opérations changent les éléments des listes de positions. Vu la simplicité et la présence du concept de l'aléatoire, le système développe des propriétés statistiques satisfaisantes.

Contribution 2 : Système de chiffrement de partition basé sur les automates cellulaire (CA-PCS): Il s'agit d'une nouvelle version de PCS qui comprend des opérations supplémentaires. Dans cette version, CA-PCS, comme son nom l'indique, utilise un automate cellulaire unidimensionnel ainsi qu'une permutation aléatoire afin de fournir de meilleures propriétés de confusion, de diffusion, et des propriétés statistiques. En premier lieu, l'automate cellulaire unidimensionnel et non-uniforme évolue 64 itérations en utilisant des fonctions locales qui dépendent de la cellule et ses voisins (droit et gauche), ce sont des fonctions booléennes à 3-variables. Ces dernières, après 64 itérations forme une fonction de transition globale qui dépend de 128 cellules, autrement dit, une fonction booléenne à 128-variables. Ce qui rend les propriétés cryptographiques dans un niveau important, (La non-linéarité, le degré algébrique, ...). Le calcul de la cardinalité idéale change ici pour pouvoir éliminer l'opération de suppression des blocs. Il y aura seulement des ajouts de blocs dont le nombre d'apparitions est inférieur à la cardinalité idéale. Puis, une permutation aléatoire est appliqué au résultat de l'étape d'insertion des blocs. L'étude des propriétés cryptographiques et des caractéristiques statistique implique que les attaques linéaires et différentielles sont difficile à atteindre. Cette contribution est détaillée dans le chapitre 3.

Contribution 3 : CFA : A New Family of Hybrid CA-Based PRNGs : Il s'agit d'une famille de générateurs de nombres pseudo-aléatoires sécurisés basés sur des automates cellulaires non uniformes. ce travail est actuellement soumis dans le Journal Security and Communication Networks en cours d'évaluation. Sa conception générale comprend trois primitives fondamentaux : une fonction de hachage, un chiffrement par blocs et un automate cellulaire. Une implémentation spécifique utilise Keccack(256), le gagnant de la compétition de SHA3, AES-256, et un CA non uniforme évoluant avec des règles soigneusement choisies avec des conditions périodiques aux limites. Pour concevoir l'AC, nous avons suivi les recommandations présentées dans [2] afin qu'il fournisse une grande qualité d'aléatoire avec une périodicité élevée, pour l'utiliser pour la génération de graines, de clés, des IV, de nonces, ou de sels à usage cryptographique. Cette construction est détaillée dans le chapitre 4.

Contribution 4 : CASC 3N vs. 4N : Effet de l'augmentation de la taille du voisinage des automates cellulaires sur sécurité du système : Une conception de chiffrement par flux inspiré par la structure du chiffrement Grain, en utilisant des automates cellulaires au lieu de LFSR, NFSR et la fonction de mélange introduite dans le système Grain[3], qui est l'un des finalistes du projet eStream. L'objectif principal de cette contribution est l'étude du passage des règles à 3 voisins (fonctions à 3-variables) à des règles de 4 voisins, (fonctions à 4-variables)et leurs caractéristiques et l'impact de cette transition sur les propriétés cryptographiques et le niveau de sécurité. En premier lieu, la version à 3 voisins est présentée dans le chapitre 5, et à partir de cette version (les règles des automates cellulaires utilisées servent pour retouver des règles à 4 variables, en ajoutant une variable de plus au lieux de parcourir toutes les fonctions boolèenes à 4-variable =65536). Un exemple est présenté dans le chapitre 5 ainsi que les détails de l'étude. Les deux versions ont des avantages qu'on peut par la suite combiner pour une meilleure solution pour différents besoins.

Contribution 5 : LCAHASH 1.1 : Une nouvelle version de la famille de fonctions de hachage LCAHASH: Il s'agit d'une version améliorée de la conception LCAHASH[4], qui est une fonction de hachage basée sur un automate cellulaire comprenant une seule itération d'une règle à 7 ou 8 variables. Dans LCAHASH 1.1, nous avons proposé de modifier la fonction de transition globale en utilisant des règles à 3 voisins(variables) pour n évolutions, ce qui fait de la transformation globale une fonction booléenne à 128 ou 256-variables. Par conséquent, elle offre une meilleure confusion et une meilleure diffusion, ainsi que des propriétés cryptographiques d'ordre plus important. Cette contribution est détaillée dans le chapitre 6.

Contribution 6 : HCAHF : A New Family of CA-Based Hash Functions : Il s'agit d'une construction inspirée par "wide-pipe construction" qui utilise des automates cellulaires. Elle comprend trois étapes : une étape de prétraitement qui divise et appliquele padding ainsi que l'introduction du sel "salt", une étape de compression comprenant un AC uniforme, en utilisant les règles des classes 3 et 4 ayants des propriétés satisfaisantes, et le XOR de leurs résultats, et à la fin, une étape de transformation comprenant un AC non uniforme fournissant des caractéristiques intéressantes au système générale. Une version spécifique produisant des digests de 256 bits est présentée et étudiée aussi dans le chapitre 6.

À la fin, une conclusion générale résume les travaux de cette thèse et présente les prochains travaux.

Content

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

3CASC	3-neighborhood Cellular Automata-based Stream Cipher
4CASC	4- neighborhood Cellular Automata-based Stream Cipher
AES	Advanced Encryption Standard
CA	Cellular Automaton
CAs	Cellular Automata
CA-PCS	Cellular Automaton-based Partition Ciphering System
CASC	Cellular Automata-based Stream Cipher
CBC	Cipher Block Chaining
CTR	Counter
DES	Data Encryption Standard
ECB	Electronic Codebook
EPP	Equal Piles Problem
HCAHF	Hybrid Cellular Automata-based Hash Function
LCAHASH	Lightweight Cellular Automata based Hash function
LFSR	Linear Feedback Shift Register
MAC	Message Authentication Code
NFSR	Nonlinear Feedback Shift Register
NIST	National Institute of Standards and Technology
OFB	Output Feedback
OTP	One Time Pad
PCS	Partition Ciphering System
PRNG	Pseudo Random Number Generator
RC4	Rivest Cipher 4
SEC	Symmetrical Evolutionist-based Ciphering
SECEX	SEC Extension to binary blocks

SHA3	Secure Hash Algorithm 3
SPN	Substitution Permutation Network
TDES	Triple-Data Encryption Standard
TRNG	True Random Number Generator
XOR	Exclusive OR

Introduction

Nowadays, the use of cryptosystems and cryptographic protocols became indispensable to achieve the information security goals in almost all information systems of different parties, such as companies, government, military, hospitals, and many others. For example, a company should keep the clients and the departments' sensitive data secret from unauthorized access, alteration, or destruction, whether sent over an unprotected communication channel or stored in a local database. They should employ robust encryption schemes so that unauthorized parties could not decrypt encrypted data or even destroy it. Cryptographic systems and protocols are fundamental to provide a satisfying security level depending on the types of applications. It is a subfield of cryptology, which includes cryptanalysis aiming to find cryptographic systems flaws. There are two basic types of cryptographic algorithms: symmetric cryptosystems involving the same key for both encryption and decryption, and asymmetric ones that make use of a specific key for each operation (encryption and decryption). An additional class of algorithms, called hybrid schemes, is defined using the combination of both types to benefit from their advantages. In this thesis, the focus goes to symmetric cryptographic primitives to ensure confidentiality and integrity using cellular automata as a building block of almost all contributions. Cellular automata are dynamic systems involving a network of cells. The purpose behind the use of cellular automata in the design of the different cryptosystems is to benefit from their characteristics, namely the simplicity, randomness properties, ease of software and hardware implementation, parallel computation, speed, as well as the cryptographic properties, which changes significantly through the time. Different research areas (physics, biology, chemistry, cryptography, etc.) use CAs. In cryptography, CAs were at first basic building blocks in the design of pseudo-random number generators, such as the wolfram rule 30 generator[1], and many others like [2], [3], [4], and [5]. They were used also to design hash functions like the Damgard design [6], Mihaljevic construction [7], [8], [9] and [10], block ciphers such as BC-CaACO [11],SPF-CA [12],and KAMAR [13], stream ciphers like hiji-bij-bij [14] and CAvium [15], and message authentication codes such as CAA [16] and LCAHASH-MAC [17].

Contributions

The fundamental purpose of this thesis is to provide data confidentiality and integrity using the cellular automata as building blocks of the proposed designs. This thesis comprises six contributions:
Contribution 1: Partition Ciphering System (PCS): It is a symmetric encryption scheme developed to make the SEC extension to binary blocks [18] more secure against frequency analysis by making the blocks or characters appear with the same frequency in the output. The partition problem was our inspiration source to achieve the specified goal with an updated definition according to our needs. Chapter 2 presents PCS at first as a standalone system followed by its combination with SEC extension to binary blocks, a previously designed symmetric encryption scheme involving an evolutionist algorithm to substitute the blocks positions.
Contribution 2: Cellular Automaton-based Partition Ciphering System (CA-PCS): It is an extended version of PCS that includes additional features. Its design involves a hybrid cellular automaton as well as a random permutation to provide better confusion and diffusion properties and statistical characteristics.
Contribution 3: CFA: A New Family of Hybrid CA-Based PRNGs: It is a secure pseudo-random Number Generators family based on non-uniform cellular automata. It is currently under review in the Hindawi's journal Security and Communication Networks. Its general design comprises three fundamental building blocks: a hash function, a block cipher, and a cellular automaton. A specific implementation uses Keccack(256), the winner of the SHA3 competition, AES-256, and a non-uniform CA evolving with carefully chosen rules with periodic boundary conditions. To design the CFA, we followed the recommendations found in [19] so that it provides a high quality of randomness with a high periodicity, to use it for the generation of seeds, keys, IVs, nonces, or salts for cryptographic use.

Contribution 4: CASC 3N vs. 4N: Effect of Increasing Cellular Automata Neighborhood Size on Cryptographic Strength: A stream cipher design having grain-like structure using Cellular automata instead of LFSR, NFSR and mixing function of grain design [20], one of the eStream project finalists. The principal purpose of this contribution is the study of the 3-neighborhood rules and their augmented 4-neighborhood rules characteristics and how this transition impacts the cryptographic properties and the security level.
Contribution 5: LCAHASH 1.1: A New version of LCAHASH hash functions family: It is an improved version of LCAHASH design [10], which is a cellular automaton-based hash function including only one iteration of a 7-variable or 8-variable rule during evolution. In LCAHASH 1.1, we proposed to change the global transition function using 3-neighborhood rules for n evolutions, which makes the global transformation a 128-variable/256-variable boolean function. Consequently, it provides better confusion and diffusion, as well as cryptographic properties.
Contribution 6: HCAHF: A New Family of CA-Based Hash Functions: It is a wide-pipe Merkle Damgard inspired construction that uses cellular automata. It comprises three steps: a preprocessing step, a compression step including uniform CA involving the class 3 and 4 rules with satisfying properties and the XOR of their results, and a transformation step including a non-uniform CA providing good characteristics to the general design. A specific version producing 256-bit digests is presented and studied.

Manuscript structure

This dissertation follows this structure:

- Chapter 1 gives a short insight into the background of symmetric cryptography and cellular automata.
- Chapter 2 provides the first contribution that aims to guarantee confidentiality PCS encryption scheme and its combination with the SEC extension to binary blocks.
- Chapter 3 details the second contribution providing confidentiality, namely CA-PCS encryption, which is an extended version of PCS.

- Chapter 4 presents the third contribution concerning confidentiality, the general design of the PRNGs family provided by CFA, and a specific implementation CFA-256.

- Chapter 5 describes the fourth contribution ensuring confidentiality, the stream cipher NCASC with two values of N (3 and 4) referring to the cells neighbor size and investigates the impact of switching these values to the cryptographic properties and security.

- Chapter 6 presents the contributions providing data integrity, namely the LCAHASH 1.1 and HCAHF designs.

This manuscript ends up with a general conclusion summarizing these contributions and future works.

Chapter 1

Cryptography and Cellular Automata

1.1 Information Security

Information security is the field of study that aims to keep sensitive data safe from illegal acts of unauthorized parties in any system. This research area got more attention through the years and became a necessity to protect information systems and confidential data of governments, military, hospitals, and many others. Cryptography is one of the fundamental tools ensuring privacy, including several security goals, namely confidentiality, integrity, authentication, availability as well as non-repudiation.

1.1.1 Confidentiality

If sensitive data are transmitted between concerned parties (sender and receiver), a third party trying to intercept sent data could not get the original (plaintext) form of encrypted data (ciphertext) without knowledge of the secret key K [21].

1.1.2 Integrity

If data do not go through any modification during their cycle life, then the data integrity is verified [21].

1.1.3 Authentication

If two parties decide to communicate, they should be able to identify each other using a signature. Also, the sent data should be authenticated by the attribution of a specific ID using Message Authenticated Codes, for example [21].

1.1.4 Non-repudiation

If two parties send/receive some data over a communication channel, they should not be able to deny sending or receiving data [21].

All these goals could be ensured by a combination of multiple cryptographic primitives to design a more integrated system. These primitives could be either symmetric or asymmetric. In symmetric cryptosystems, the sender and receiver use the same key for both encryption and decryption, as illustrated in figure 1.1. On the other hand, in asymmetric cryptosystems, the sender uses its secret key SK to produce the ciphertext, which is decrypted using its public key pk by the receiver as displayed in figure 1.2.

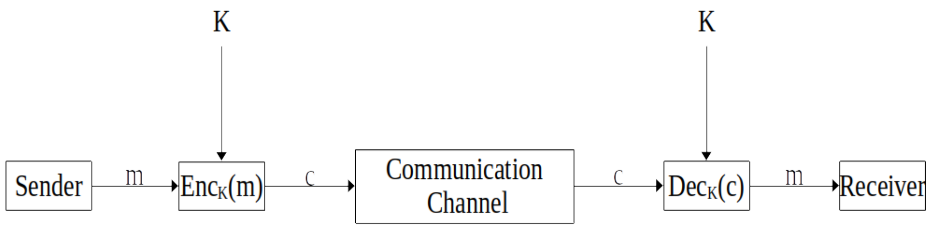


Figure 1.1: Symmetric Cryptosystems

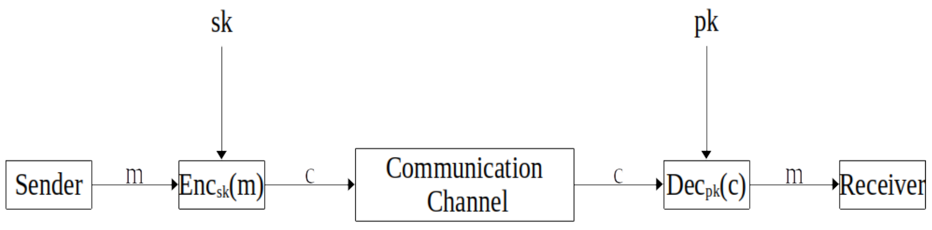


Figure 1.2: Asymmetric Cryptosystems

The rest of this section is concerned with the symmetric primitives, including Pseudo-Random Number Generators PRNGs, stream ciphers, block ciphers, hash functions, and Message Authentication Codes (MACs).

1.2 Symmetric Cryptography

1.2.1 PRNGs

The generation of random sequences is an essential cryptographic feature. Almost all cryptographic applications need random bitstreams such as session keys for encryption, initial vectors, salts, nonces, and many others. Fundamentally, one of two methods that generate these bitstreams. A pseudo-random number generator(PRNG) is a mostly used technique in cryptographic applications that involves deterministic algorithms. The second method makes use of non-deterministic physical sources of randomness, termed true random number generator (TRNG). For cryptographic use, bitstreams should have high-quality of randomness and unpredictability. To consider the bitstream random, it needs to satisfy two fundamental conditions. The first one is the uniform distribution of ones and zeros in the sequence, which means that the bits should occur approximately at the same rate. The second condition is the independence of the bits or subsequences that form the bitstream, which means that there is no correlation between them. Still, for several applications like the generation of session keys and stream ciphers, there is a further condition in addition to the mentioned constraints, which is the unpredictability. If each bit /number of a sequence generated by a TRNG is statistically independent of each other, then the unpredictability is satisfied. However, TRNGs are not suited for all applications because of their limitations (e.g., efficiency problems). Accordingly, PRNGs producing pseudo-random numbers that look like random ones are generally adequate for many situations. Thus, designers should keep in mind the unpredictability condition [22]. Consequently, an adversary having access to the current number would never predict the next number or even the previous one. More details are depicted in the rest of this part.

Deterministic algorithms are adopted to generate random sequences for cryptographical use. It means that the resulting bitstreams are not statistically independent, and consequently, they are not random. Still, if the algorithm design is well, it will provide sequences satisfying most of the randomness tests. These sequences are considered pseudo-random numbers/

sequences. If some conditions are satisfied, pseudo-random numbers are used instead of random numbers and maintain similar results for a specific use. Therefore, PRNGs are broadly adopted by many other applications besides cryptography, such as physics, statistics, and many others [22]. Figure

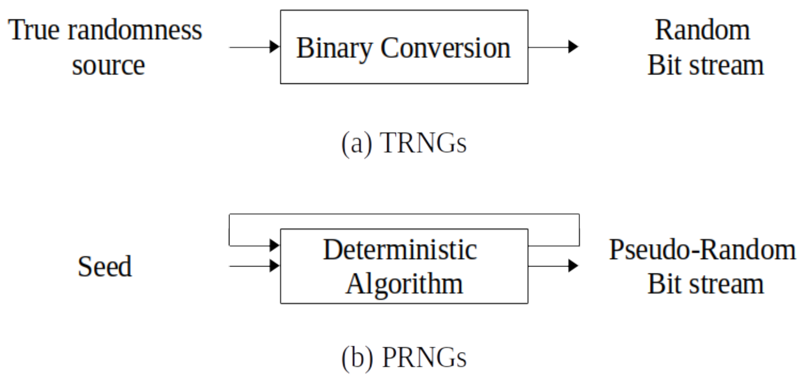


Figure 1.3: PRNGs and TRNGs

1.3 displays a global vision of TRNGs and PRNGs. Commonly, a TRNG accepts an entropy source as input, which is random and could be formed by a single or several physical sources. These sources could be audio or video input, hard disk activity, system clock, or mouse movements. When the TRNG algorithm takes these inputs, it will do a conversion to get a binary output, or it will apply further transformations in the case of biased sources. On the other hand, a pre-generated seed, which must be secret, is an input to the PRNG. It will go through using the deterministic transformations provided by the algorithm to produce pseudo-random sequences. Depending on the demanded number of bits, the algorithm requires each output in the next number generated. In general, it is better to use a TRNG to provide the seed [22].

1.2.1.1 The PRNGs requirements

a) Randomness

The conditions of a PRNG concerning the randomness properties rely on the fact that the resulting sequence should look like a random one, although it is deterministic. There are two popular statistical test tools, the NIST statistical test suite and the dieharder battery of tests designed to examine the quality of the PRNGs outputs. According to the percentage of successful tests, the randomness quality of the PRNG decided. For example, according to the NIST SP 800-22 [23] , three characteristics are concerned with tests:

- **Uniformity:** For each random or pseudo-random bitstream, 0s and 1s should appear with the same probability, which should be 0.5. And if n is the bitstream size, then zeros and ones should appear exactly n/2 times [23].
- **Scalability:** If a test was successful on a bitstream, it should be valid also for sub-bitstreams taken randomly from the latter. That is to say, if a sequence is random, then a subsequence taken from this sequence should also be random[23].
- **Consistency:** Multiple seeds should be used in the generation of the bitstream to test the consistency of the PRNG's behavior[23].

b) Unpredictability

Two kinds of unpredictability should be satisfied by the outputs of a PRNG:

- **Forward unpredictability**: If the seed is secret, even if some previous bits are known, the next bit is supposed to be unpredictable.
- **Backward unpredictability**: It should be impossible to find the seed from a known sequence. That is to say, all the sub-sequences generated using the same seed should not display any correlation.

These properties could be verified using the same tests provided by NIST and Dieharder battery[24]. If the results show that the sequence tested is random, then it is unachievable to predict the following bit based on the known previous bits. Likewise, if the bitstream looks like a random sequence, then it is impossible to determine the seed based on this later if known since no correlation is supposed to figure between the seed and the bitstream [22].

c) Periodicity

The period of a PRNG is a significant characteristic, which is defined to be the period of bitstream reproduction, using the same seed. This parameter should be large enough to say that the PRNG is good. If the period is short, then the unpredictability of the bitstream generated turns out to be unsatisfied [22].

1.2.1.2 Seed Requirements

When the PRNG is particular for cryptographic use, then the corresponding seed should be secure. In other terms, it should be unpredictable so that an opponent is unable to get it, and therefore, the output of the PRNG since the latter is deterministic. Consequently, the seed should be random or look like a random sequence. In general, it should be produced by a TRNG as recommended by the NIST report SP 800-90A. However, not all the times a TRNG is accessible or practical (e.g., for stream ciphers, TRNGs are not convenient). For this reason, a secure PRNG is adequate to generate the seed [22].

1.2.1.3 The Algorithm Design

The development of cryptographically secure PRNGs is a challenging task. Before getting involved in the design, one should decide which type of PRNGs he is developing:
a) Specially developed algorithms
These algorithms are specific to the generation of pseudo-random sequences. Certain algorithms of this type are adequate for different applications of PRNGs like Linear Congruential Generators and Blum Blum Shub Generator. While other ones are conceived particularly for stream ciphers like RC4.
b) Cryptographic primitives-based algorithms
These algorithms rely on the random behavior of pre-developed cryptographic primitives, which should not display a specific pattern to make the cryptanalysis challenging. Hence, these primitives are adequate to produce a cryptographically secure PRNG since the security of this latter is based on these primitives' security. In general, these primitives could be block ciphers, asymmetric ciphers, hash functions, or Message Authentication Codes.

1.2.1.4 Crytographic Statistical Test Suites

This subsection presents a short description of the most popular test suites used in cryptography, the NIST Statistical Test Suite and the Dieharder battery.

1.2.1.4.1 NIST STS

The National Institute of Standards and Technology designed the NIST Statistical Test Suite that comprises 15 tests to evaluate the PRNGs' behavior and their statistical properties. For each sequence, the test algorithms compute the p-value, which refers to the probability that a TRNG was used to generate it. More specifically, the p-values estimate the distance between the corresponding bitstream and random ones. The results of the tested algorithms are successful if the p-values are in the range $[\alpha, 1-\alpha]$, where alpha is the significance level, $\alpha = 0.01$. The NIST report [23] provides more details about this test suite.

1.2.1.4.2 Dieharder

Dieharder is a battery of statistical tests, comprising 32 tests, designed by Brown to check the randomness and statistical properties of pseudo-random number generators and cryptographic primitives such as encryption systems, hash functions, and MACs [24]. The battery algorithms compute the p-values, which are the probabilities that the bitstream tested is random. Accordingly, the sequence is said to be random or not. If the p-values are between $0+\alpha$ and $1-\alpha$, then the bitstream is indistinguishable from a random bitstream, where alpha is the significance level such that $\alpha = 0.005$.

1.2.1.4.3 Statistical Tests

This subsection presents a short description of some statistical tests provided by the NIST statistical test suite and Dieharder.

a) **Frequency Monobit Test :** This test aims to evaluate the percentage of the ones and zeros in the bitstream, which is supposed to be random if the ones and zeros appear nearly at the same rate. Other statistical tests results rely on the success of this test.

b) **Frequency Test within a Block :** This test is a general form of the Frequency Monobit Test. It measures the proportion of ones in M-bit subsequences from the whole bitstream. If the ones appear in almost 50% of positions in the block, then this test is successful, and the bitstream has a high randomness quality. In the case of $M = 1$, it leads to an instance of the frequency test.

c) **Runs Test :** This test's objective is to check if the number of subsequences formed by consecutive 1s or 0s in the bitstream is as supposed to be in a random sequence or not. It evaluates how the variation from 1s to 0s or from 0s to 1s is in the bitstream.

d) **The Longest Runs Test :** This test aims to check if the maximum size of subsequences formed by 1s, namely runs, in the bitstream evaluated is close to what is supposed to be in a random sequence. The 1s runs evaluation indicates if the 0s longest runs are regular, that is why this test concerns 1s runs only.

e) **Rank Matrix Test :** This test tries to verify if there is a linear dependency between substrings. The focus of the test is the rank of disjoint sub-matrices of the entire sequence. The purpose of this test is to check for linear dependence among fixed-length substrings of the original bitstream. Note that this test also appears in the DIEHARD battery of tests.

f) **Approximate Entropy Test :** This test evaluates the bitstream by computing the frequency of all possible k-bits and (k+1)-bits blocks in it. Then it is compared with the values that should be in a random sequence.

More detailed descriptions of these tests and the other ones could be found in the NIST report [23] and [24].

1.2.2 Stream Ciphers

Stream Ciphers form a class of symmetric encryption schemes that are built based on PRNGs designs to produce the keystream, which goes through the XOR operation with the plaintext to get the ciphertext. Many investigations over the years lead to the development of several systems. Stream ciphers look like the One Time Pad system [25], except that PRNGs replace TRNGs involved in OTP. Following the recommendations found in [22], stream ciphers should include PRNGs that produce sequences with a high period. In general, the functions employed in PRNGs generate deterministic bitstreams. Thus, it ends up reproducing similar subsequences. The higher periodicity makes the cryptanalysis challenging. Also, PRNGs should provide keystreams with high randomness quality, as stated in the previous section (PRNG). This feature transfers to the ciphertext. Therefore, cryptanalysis is more challenging. Moreover, the key or seed input to the PRNG should be long enough to make the system robust against brute force attacks. It should be at least a 128-bit key [22]. There are two kinds of stream ciphers:

1. **Synchronous Stream Ciphers :** In this type of stream ciphers, the keystream generated has no correlation with the plaintext or the ciphertext. A cryptographic PRNG is involved, such that the Key and IV are the only inputs [26].

2. **Self-Synchronous Stream Ciphers :** In this type of stream cipher, in addition to the key and IV, the PRNG involved takes the previous bits of the ciphertext as an input to produce the keystream [27].

Figures 1.4 and 1.5 display the general structure of synchronous and self-synchronous stream ciphers, respectively.

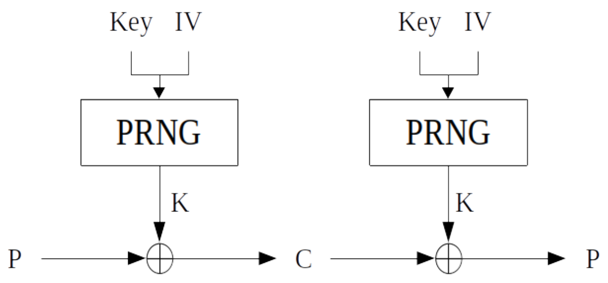


Figure 1.4: Synchronous Stream Ciphers

The use of linear feedback shift register LFSR together with nonlinear feedback shift register NFSR in the same system or nonlinear combination of multiples LFSRs are among the most popular designs adopted. However, these designs are still vulnerable to attacks like fault attack and correlation attacks, even if they provide a high randomness quality. In this thesis, a stream cipher involving cellular automata as the fundamental building blocks instead of LFSRs and NFSRs is provided by Chapter 5.

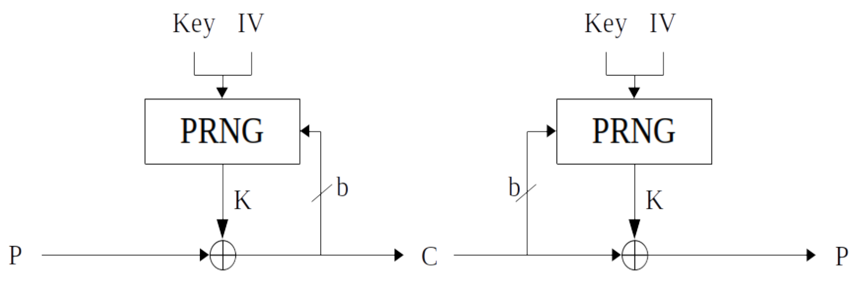


Figure 1.5: Self-Synchronous Stream Ciphers

1.2.3 Block Ciphers

A block cipher [22] is a second class of symmetric encryption systems that process over a plaintext of arbitrary length by splitting it into n-bit blocks. Next, a pre-defined encryption function f_E is applied to each block M_i to provide C_is, $i \in \{1, ..., m\}$, $C_i = f_E(M_i, K)$ using a specific mode (ECB, CBC, OFB, or CTR). In the case of M not being a multiple of n, it is padded by 1||0*.

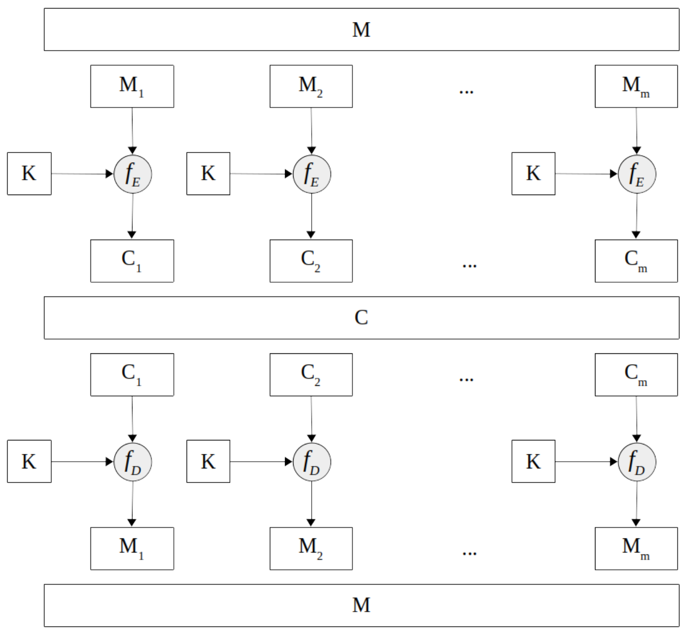


Figure 1.6: A block cipher general structure in ECB mode

Figure 1.6 presents a general structure of a block cipher that uses ECB mode. The encryption/decryption function should follow a specific structure: either a substitution-permutation network, a Feistel scheme, or a combination of both. These structures provide two main cryptographic properties, namely confusion, and diffusion, which are fundamental for the block ciphers, as well as other symmetric primitives, security purposes. The confusion property illustrates the statistical relationship between the key and the ciphertext, while the diffusion represents the statistical relation of the plaintext and the ciphertext [22].

Substitution-Permutation Network

According to the proposition of Shannon, a robust SPN-based block cipher follows a complex combination of two transformations, a substitution as well as a permutation, using the key's bits and multiple layers to fully mix-up the plaintext bits and to remove any existing potential pattern. The substitution is performed using various nonlinear S-Boxes replacing sub-blocks bits in each round. This feature should provide satisfying confusion. For this reason, S-Boxes should be carefully designed and studied. On the other hand, the permutation involves a well designed linear P-box to swap the bits of the block to produce high diffusion to the whole system [28].

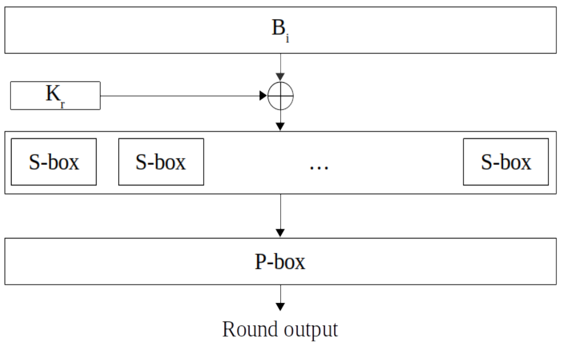


Figure 1.7: One round of Substitution-Permutation Network

Figure 1.7 illustrates one layer of the general SPN- designs for one block. At first, a round key is XORed with the bits of the blocks. Next, the result is fed to S-boxes. The resulting bits are then permuted using P-box. SPN-based block ciphers include AES [29], PRESENT [30], SHARK [31], and many others.

Feistel Network

A Feistel cipher [32] is one of the commonly used structures in the design of block ciphers. The general process starts with the split of the n-bit blocks into n/2-bits sub-blocks L_0 and R_0, where L_0 is the left sub-block and R_0 is the right sub-block. Then, the sub-blocks are processed r rounds and combined to produce a ciphertext block. In one round (round 1 in Figure 1.8), a round key (K_1) is involved to compute $F(R_0, K_1)$ and the Exclusive-OR of the output with L_0. Then the result is fed to the right sub-block of the next round input $R_1 = L_0 \oplus F(R_0, K_1)$. The left sub-block L_1 is supplied by the previous right sub-block R_0. Figure 1.8 illustrates the encryption process using one round of Feistel structure. After r rounds, L_r and R_r are swapped/switched to provide the ciphertext block $C_j = R_r || L_r$.

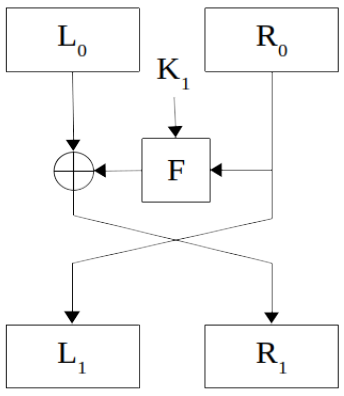


Figure 1.8: One round of Feistel structure

The function involved in the encryption process serves to decrypt a ciphertext block, and the order of round keys is reversed. Different parameters are involved in the design of a secure Feistel-based block cipher, such as the block length, the key-length, the number of rounds, the key rounds generator, as well as the round function F. Also, the block cipher should be fast and easy to analyze. Feistel based designs include DES [33], TDES[34], Blowfish [35], and many others.

1.2.4 Hash Functions

A cryptographic hash function is a primitive ensuring data integrity by means of a one-way function. It proceeds over an arbitrary-length message to produce an n-bit message digest $digest = H(M)$. Formally, a hash function $H : \{0,1\}* \rightarrow \{0,1\}^n$ s a function mapping an arbitrary length message $M \in \{0,1\}*$ to a fixed length output $H(M) \in \{0,1\}^n$, $n > 0$. There are two kinds of hash functions: unkeyed and keyed hash functions. Their most common applications include digital signature, public-key encryption, message authentications, virus detection, and intrusion detection. From the above definition, it is clear that hash functions are many to one since the size of the input is not fixed, which means that collisions are not evitable. However, additional conditions are required to say that a hash function is cryptographically secure [**?**].

Collision resistance

It should be hard to find two different messages m_1 and m_2 that hash to the same value $H(m_1) = H(m_2)$.

Pre-image resistance

It should be infeasible to find a message m corresponding to a given hash value H(m).

Second-preimage resistance

Given a message m_1 and its hash value $H(m_1)$, it should be hard to find a second message m_2 that hashes to the same value $H(m_1) = H(m_2)$.

The complexity to find a collision, a preimage, or a second-preimage is related to the size of the hash functions output. Finding collisions requires $2^{\frac{n}{2}}$ operations, due to the introduction of the birthday attack [36]. On the other hand, it requires 2^n to find a preimage or a second preimage [**?**].

Moreover, a fundamental condition should be verified. A secure hash function should conduct like a random oracle. For this purpose, the output should be unpredictable. If two messages are different by only a few bits, their hash values should be different. This characteristic strengthens the hash function system. Also, it should be efficient enough to make the software and hardware implementations easy.

There are two forms of hash functions:

1) **Merkle Damgard Construction:** This type of iterative constructions forms a hash function that makes use of a compression function

CF. It starts by the split of the message to hash into m-bit blocks M_1,..., M_k. Next, the last block is padded if necessary. Then, CF is applied iteratively over blocks together with k n-bit chaining variable h_i, such that $h_0 = IV$[37]. Figure 1.9 displays the general design of a Merkle Damgard construction.

2) **Sponge Construction:** This construction is a second class of iterative constructions that is completely different from the Merkle Damgard constructions. It involves permutations instead of compression functions and outputs variable-length outputs instead of a fixed size output. Also, no IV is involved. A sponge hash function [38] comprises two fundamental steps:

   – **Absorbing Step:** At this level, the message is firstly padded. Next, it is split into r-bit blocks M_1, ..., M_k. Then, a b-bit state goes through the permutation f iteratively together with the blocks. Initially, the state is set to $\{0\}^b$, such that $b = r + c$ where r is termed the bitrate and c the capacity. The first r bits of the state are XORed each time with the i^{th} block M_i. Afterward, it is fed to f together with the rest c bits of the state, $1 \leq i \leq k$.

   – **Squeezing Step:** This step outputs r bits of the state and applies f to get more outputs as long as more bits are desired to fit the size specified by the user.

   Figure 1.10 summarizes the general design of a sponge hash function. Examples of Sponge constructions include Keccack [39], which the winner in the SHA3 competition, and Spritz [40], a hash function inspired by the RC4 design.

1.2.5 Message Authentication Code (MACs)

Message Authentication Codes (MACs) are symmetric primitives employed to ensure data integrity and authentication. Three building blocks form a MAC algorithm [41]:

- A Key Generation Mechanism: The k-bit secret key K should be generated using a cryptographic PRNG [41].

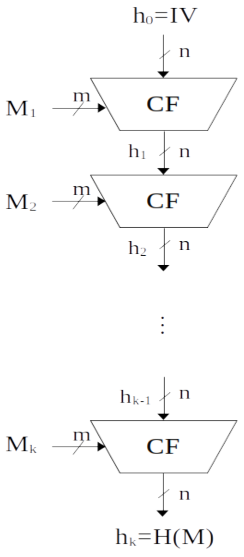


Figure 1.9: Merkle Damgard Construction

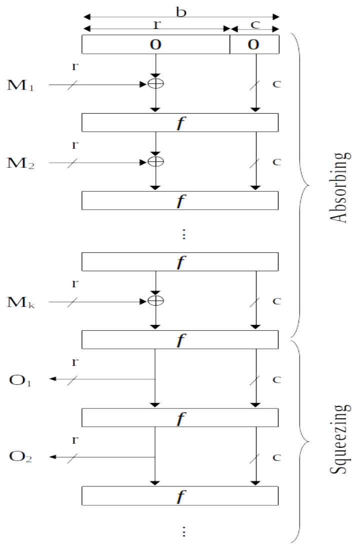


Figure 1.10: Sponge Construction

- A MAC generation mechanism: Given a message M and the secret key K, the process of this mechanism produces a MAC value $MAC_K(M)$, which is also termed tag, of a fixed-length n [41].
- A MAC Verification mechanism: This mechanism serves to check the message authenticity given the secret key K and the MAC value $MAC_K(M)$

Figure 1.11 illustrates the MAC generation and verification processes. Most of the MAC designs are based on either a block cipher like AES-CMAC [42], a hash function such as HMAC-MD5 [43], or a dedicated design like MAA [44].

A secure MAC should be robust against forgery attacks, where an opponent tries to reproduce the MAC of a message without knowledge of the secret key, and known-message attacks, in which attackers seek to get a set of messages and their MACs. In addition to chosen-attacks that aim to find the MACs of specific messages [45].

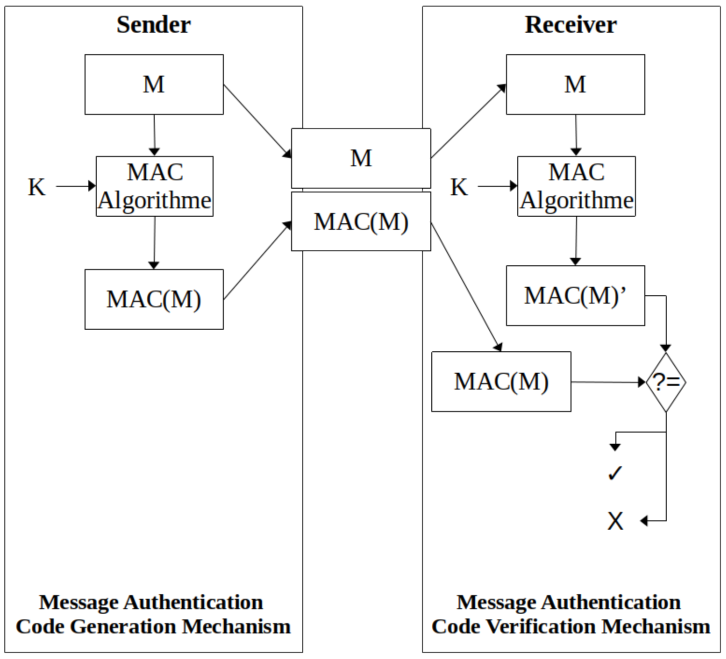

Sender
M
K
MAC Algorithme
MAC(M)
Message Authentication Code Generation Mechanism
M
MAC(M)
Receiver
M
K
MAC Algorithme
MAC(M)'
MAC(M)
?=
✓
X
Message Authentication Code Verification Mechanism

Figure 1.11: Message Authentication Codes General Design

1.3 Cellular Automata

1.3.1 Introduction to Cellular Automata

Cellular automata, which were proposed initially by John Von Neumann in the 1950s, form a class of discrete dynamic systems. They were widespread in the 1980s through Stephen Wolfram's purposes [46]. At first, Von Neumann studied their relevance in the biological self-reproduction modeling through the suggestions of Stanislas Ulam [47]. Later, several areas, namely biology, chemistry, physics, mathematics, and so on, employed cellular automata in modeling and solving natural, physical, and real-life problems. The CAs are getting the researchers' attention due to their complicated global conduct arising from the simple cellular transformations, in addition to their randomness and computation universality features [48]. A cellular automaton is a discrete system consisting of n cells, which have a finite number of states, arranged as an array. Each cell state is updated in a discrete-time using the neighbor cells state. Formally, a cellular automaton is a quintuple (L, S, N, f, R), such that:

- L refers to the space of the d-dimensional cellular.
- S is the set of finite-state.
- N is the vector indicating the cells' neighborhood, which is denoted by a radius r that designates how many successive cells are involved in each cell evolution.
- F is a local function, also called a local rule, that produces each cell's future state.
- R denotes the set of rules involved to update the CA cells.

The variation of L, S, and N results in several kinds of cellular automata. For instance, 2-dimensional cellular automata with 5-neighborhood and 29-state were proposed and studied by Jon von Neumann. If the local transformation f involves only the XOR operator, then the CA is linear. If additional operations are included, such as OR or AND, then the CA is nonlinear. The ruleset R defines if the CA is uniform or not. A uniform CA uses a single rule to update all the cells $|R| = 1$. A non-uniform CA involves more than one rule to evolve the CA cells $|R| > 1$. Multidimensional cellular automata allow a more complex behavior, but their study is more challenging compared to 1-dimensional CAs. Consequently, most of the cryptographic applications involve 1-dimensional CAs, especially Stephen Wolfram's elementary CAs [49], which are 2-state 3-neighborhood CAs. These CAs have $2^3 = 8$ neighbor configurations and $2^{2^3} = 256$ rules. Table 1.1 provides examples of linear and nonlinear Elementary CAs rules.

Table 1.1: Elementary Cellular Automata Example

Neighborhood configuration	111	110	101	100	011	010	001	000
NL Rule 86	0	1	0	1	0	1	1	0
L Rule 60	0	0	1	1	1	1	0	0

Such that x_{i-1}, x_i, and x_{i+1} are the left neighbor, the cell, and the right neighbor, respectively. According to Wolfram [121], the name of each binary rule is its decimal value (for example, $(01010110)_2 = (86)_{10}$ is denoted rule 86). In addition to 3-neighborhood CAs, this thesis includes

4-neighborhood CAs in chapter 5 in the design of a stream cipher. By analogy to 3-neighborhood 1-dimensional CAs, 4-neighborhood 1-dimensional 2-state CAs have $2^4 = 16$ neighborhood configurations and $2^{2^4} = 65535$ rules. Examples of linear and non-linear 4-neighborhood cells are in table 1.2.

Table 1.2: 4 Neighborhood Cellular Automata Example

Neighborhood configuration	NL Rule 42390	L Rule 42330
1111	1	1
1110	0	0
1101	1	1
1100	0	0
1011	0	0
1010	1	1
1001	0	0
1000	1	1
0111	1	0
0110	0	1
0101	0	0
0100	1	1
0011	0	1
0010	1	0
0001	1	1
0000	0	0

To see how a CA evolve during the time, space/time diagram is used, where the CA space is provided by the x-axis, and the time is illustrated by the y-axis, such that the different states are represented by colors, (e.g. black and white in the case of two-state CA). Space/time diagram is useful to display and analyze the overall behavior of a cellular automaton according to its ruleset R. Examples of space/time diagram are represented in the following table (86, 60, 24390, 24330).

1.3.2 Boundary Conditions

In cryptographic applications, finite-state CAs are used to explore a particular framework. The boundary conditions should be defined to do so. The most used ones are:

- null boundary where the leftmost and the rightmost cells neighbor cell are set to '0'.
- Periodic boundary where the leftmost and the rightmost cells are neighbors.
- Adiabatic boundaries where the leftmost and the rightmost cells are their own neighbor.
- Reflexive boundaries where the neighbor cells are the same (left neighbor = right neighbor)

This thesis is concerned by the periodic boundary conditions due to the properties provided in the leftmost and the rightmost cells, which spread through evolutions

1.3.3 Boolean functions properties

This section presents short descriptions of boolean functions' properties that help to measure their adequacy with cryptographic systems. More details could be found in [50].

Hamming weight

Given an n-variable boolean function f, its hamming weight wt(f) is defined to be the number of ones in its binary representation. If $wt(f) = 2^{n-1}$ then f has a satisfying hamming weight.

Hamming distance

Given two n-variable boolean functions f and g, the hamming distance between f and g is denoted by the hamming weight of the XOR of f with g, $d_H(f, g) = wt(f \oplus g)$.

Nonlinearity

Given an n-variable boolean function f, its nonlinearity is deduced from the following formula

$$NL(f) = min\{d_H(f, g)|g \in AF\}$$

such that AF is the set of all n-variable affine boolean functions where NL(f) is bounded by $2^{n-1} - 2^{\frac{n}{2}-1}$

Algebraic degree

Given an n-variable boolean function f, the algebraic degree corresponds to the number of variables in the highest order term. It is bounded by n-1.

Balancedness

An n-variable boolean function f is said to be balanced if its hamming weight is 2^{n-1} ($wt(f) = 2^{n-1}$).

Correlation immunity

An n-variable boolean function is said to be mth order correlation immune if its outputs are independent of at most m variables of its input ($m < n$).

Resiliency

An n-variable boolean function is m-resilient if it is a balanced mth order correlation immune function.

1.3.4 Cryptographic Application of CAs

Cryptography is one of the fields making use of cellular automata in almost all primitives. Chapter 3 gives a summarized view of their use in encryption in section 3.2. Next, Chapter 4 provides how CAs were good candidates in the design of secure PRNGs in section 4.2. Also, chapter 5 gives a summary of their use in the development of stream ciphers and how they replaced the preceding features (LFSRs and NFSRs) in section 5.2. Afterward, Chapter 6 presents the different applications in the hash functions schemes in section 6.2.

Part I

Confidentiality

Chapter 2

PCS: Partition Ciphering System

2.1 Introduction

This chapter describes the Partition Ciphering System (PCS). It is a symmetric encryption algorithm that got inspiration from the study of the partition problem and the Symmetrical Evolutionist-based Ciphering (SEC) scheme [51]. Concisely, this latter is an encryption scheme based on an evolutionary algorithm that aims to substitute all the plaintext characters to result in a change in the appearance frequency [51]. At first, this contribution was designed mainly to be combined with the SEC algorithm, to increase its resistance against frequency analysis, referring to the constraint of equality of appearance frequency. Nevertheless, in this section, PCS is first presented and studied as a standalone system. Then, it is joined with the SEC extension to binary blocks to produce a multiple-encryption scheme. The main objective behind this construction is to achieve high entropy so that frequency analysis could not reveal any information. Consequently, it will display good results for statistical tests. According to the SEC design, the partition problem is the most practical problem to get inspiration, especially the Equal Piles Problem, which was defined by Jones and Beltramo in [52]. It is one of the grouping problems aiming to group or partition a set of elements into disjoint subsets(piles) referring to a specific constraint, which is the subsets sums equality[53]. However, to reach the goal specified before, an adaptation of the EPP to the Card-Partition Problem was suggested.

The idea is to partition a set into subsets(piles) such that the subsets cardinals are equal. Accordingly, the PSC encryption algorithm processes using simple operations to find a balanced ciphertext. Afterward, the statistical tests of Dieharder were applied to analyze the behavior of PCS. Also, the confusion and diffusion tests were verified using the avalanche effect to determine if PCS respects the requirements of a secure symmetric scheme, as stated by Claude Shannon. Moreover, a comparison was performed regarding the statistical tests, the time of encryption and decryption, in addition to attacks. Finally, the combination of PCS with SEC is detailed. The rest of this chapter is organized as follow: Section 2.2 details the related work. Afterward, the PCS encryption and decryption algorithms are presented, respectively, in section 2.3 and 2.4. Next, section 2.5 presents the combination of PCS with SEC extension to binary blocks (SECEX). Finally, section 2.6 provides the results and security analysis.

2.2 Related Work

In 1991, Jones and Beltramo defined the partition problem to be the Equal Piles Problem, which is also known as the Load Balancing Problem, in their article. They used nine genetic algorithms to solve a challenging instance of this problem. This latter comprises a set of thirty-four numbers to be partitioned into ten subsets(piles), which sum to the same value [52]. However, the algorithms tested could not find an optimal solution. Later, this instance was studied by Falkenauer [53] and William [54] in their articles. In 1995, Falkenauer adopted an updated version of the grouping genetic algorithm. He defines a different encoding where the genes are the piles(subsets), he uses revised GA crossover and mutation operators to find an optimal solution to the Equal Piles Problem. Falkenauer proposed this algorithm previously to solve the Bin Packing Problem, where items of different sizes are packed to some limited capacity bins[55]. As the Equal Piles Problem and the Bin Packing Problem are part of the grouping problems[55], and the main difference between these problems is the number of subsets (piles). It is easy to adapt the algorithm to find better solutions compared to the solutions found by Jones and Beltramo. Afterward, in 2000, William proposed a genetic algorithm, called Eager Breeder, producing better results compared to Falkenauer's and Jones and Beltramo's solutions[54]. He developed the

Eager Breeder algorithm involving a more expensive representation of genes and differs from the grouping genetic algorithm in the crossover and mutation operators. The crossover aims to choose only the best genes from parents to conceive the genes of a child. Moreover, the mutation operator tries to remove a random element from a subset and insert it into a random subset[54]. Afterward, the cryptographic algorithms included the benefits of genetic and evolutionary algorithms to ensure better security. For instance, Omary and al. proposed the standard version of the SEC in [56], which performs the encryption process on characters. Next, they extended SEC with two versions. The binary extension of SEC [18] [57], which applies the encryption to binary data split into binary blocks instead of characters, and the fusion extension [57], which includes a pre-processing step to increase resistance to frequency analysis. The contribution of this chapter is related, but not limited, to the binary extension of the SEC. It is also compatible with the standard version of the SEC. Later, various approaches were introduced by Trichni et al. [58], Bougrine et al. [59], Kaddouri et al. [60][61], and Mouloudi [62] to strengthen the SEC against frequency cryptanalysis and brute force attacks. In 2011, Trichni et al. proposed an enhanced SEC version [58] that makes use of a new partition problem-based mutation operator. Later, in 2012, Bougrine et al. suggested another technique[59] producing a new SEC version. Afterward, in 2013, Kaddouri et al. proposed two SEC versions, a balancing process was applied to the standard SEC version to produce the first version[60], and a second version that involves a binary fusion operation in the binary block extension of SEC[61]. In 2015, Mouloudi adopted a new technique called fragmentation[62] that he combined with the standard SEC version to make the characters appear with nearly the same frequency. In this chapter, the proposed technique used to design the Partition Ciphering System (PCS) makes the characters or the binary blocks appear with the same frequency providing the best equilibrium compared to the previous work.

The motivation of the design of the Partition Ciphering System (PCS) was to strengthen the SEC binary extension system. The main intention was to make the blocks or characters appear with the same frequency in the ciphertext. Accordingly, the Equal Piles Problem was a suitable candidate that inspires us to reach our purpose. After the study of this problem regarding the SEC system, an adaptation of this problem was recommended.

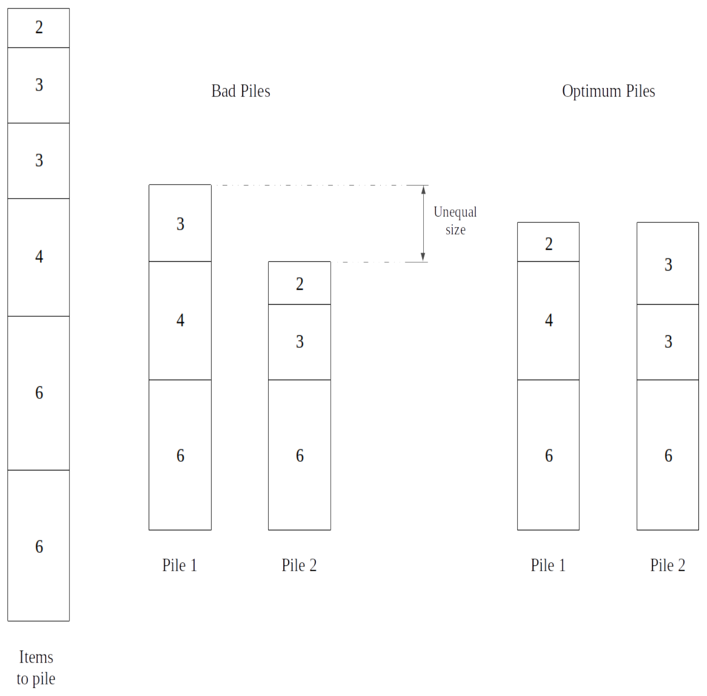


Figure 2.1: Equal Piles Problem

Consequently, the Card-Partition Problem was defined. The following figures (Figure 2.1 and Figure 2.2 display models of the equal piles problem and the card partition problem.

As the Equal Piles Problem is a grouping problem, it is then formally defined as follows:

The Equal Piles Problem

Given a set S of integer numbers and an integer t. Partition the set S into t subsets such that:

$$\sum_{el \in S_1} el = \sum_{el \in S_2} el = ... = \sum_{el \in S_t} el$$

and $S_1 \cup S_2 \cup ... \cup S_t = S$ and $S_1 \cap S_2 \cap ... \cap S_t = \emptyset$ where el are the S_j's elements, and S_j is the j^{th} subset.

The definition of the Card-Partition Problem differs from the Equal Piles Problem in the criteria that must be satisfied in the resulting partition. The subsets' cardinals replaced the subsets' sums in our case.

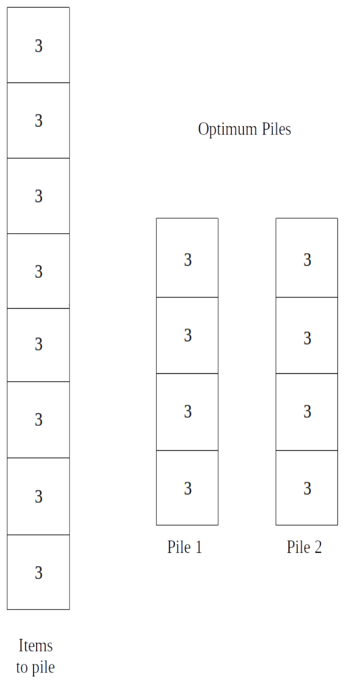


Figure 2.2: Card-Partition Problem

The Card-Partition Problem

Given a set of integers and an integer t. Partition the set S into t subsets such that:

$$cardinal(S_1) = cardinal(S_2) = ... = cardinal(S_t)$$

and $S_1 \cup S_2 \cup ... \cup S_t = S$ and $S_1 \cap S_2 \cap ... \cap S_t = \emptyset$ where $cardinal(S_j)$ is the S_j's number of elements, and S_j is the jth subset. From our perspectives, the elements of S have the same size. This latter is fixed in the pre-processing phase of PCS. The following subsections detail the steps of PCS encryption and decryption.

2.3 The Partition Ciphering System (PCS) Encryption

The aim of the proposed system, as stated before, is to construct an ideal partition corresponding to the ciphertext. In this latter, all the blocks appear at the same rate. The PCS algorithm comprises three steps to reach this goal.

2.3.1 The PCS Pre-processing Step

The objective of this step is to represent the plaintext as a partition. Since the idea of this system's design is relative to the SEC, the best representation to adopt is the same used in SEC[56]. At first, the split of the binary message into blocks of the same size s, where $2 < s \leq 16$, is performed. Next, for each block, all the positions where it figures are grouped in one list. This latter is the corresponding list of the appearance of the block in the plaintext. Figure 2.3 displays the process of this step.

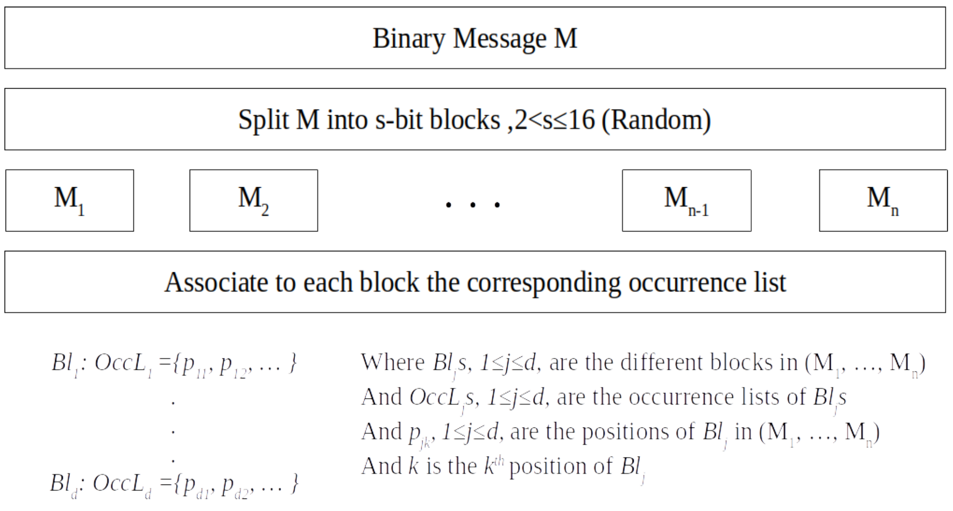

Binary Message M
Split M into s-bit blocks ,2<s≤16 (Random)
M_1
M_2
. . .
M_{n-1}
M_n
Associate to each block the corresponding occurrence list
Bl_1: $OccL_1 = \{p_{11}, p_{12}, \dots\}$
Bl_d: $OccL_d = \{p_{d1}, p_{d2}, \dots\}$
Where Bl_js, $1 \leq j \leq d$, are the different blocks in $(M_1, \dots, M_n)$
And $OccL_j$s, $1 \leq j \leq d$, are the occurrence lists of Bl_js
And p_{jk}, $1 \leq j \leq d$, are the positions of Bl_j in $(M_1, \dots, M_n)$
And k is the k^{th} position of Bl_j

Figure 2.3: The PCS pre-processing step

2.3.2 The Ideal Cardinality Computation Step

The ideal cardinality is the number of occurrences required of each block to construct the final partition. Let d be the number of distinct blocks in

M=(M_1, ..., M_n), and n be the number of blocks in M. If $\frac{n}{d} \in \mathbb{N}$ then $IC = \frac{n}{d}$ else $IC = \lceil \frac{n}{d} \rceil$.

2.3.3 Final Partition Construction Step

In the ciphertext, the number of blocks is $m \geq n$ where $m = d \times IC$. Consequently, the resulting partition found in this step is a partition of $\{1, 2, ..., m\}$ where all the blocks occurrence lists have the same cardinal.
Throughout this step, according to the actions performed, the secret key is established. It comprises four elements: the blocks' length s, the number of additional blocks $NbInsBl$, the suppression list $SuppLi$ comprising the removed blocks and the positions of removal, and a permutation π resulting from the mapping transformation between the initial partition and the resulting partition of this step. $sk = \{s, NbInsBl, SuppLi, \pi\}$.
The cardinal of each distinct block occurrence list $OccL$ is concluded and compared with IC. Therefore, two possible actions are performed depending on the situation encountered. It is determined whether to insert or remove a block.

1- The Block Insertion

This operation involves the block Bl_j for which the cardinal of the occurrence list corresponding $OccL_j < IC$ for $1 \leq j \leq d$. The algorithm, at this stage, inserts Bl_j at the last position of M. Also, it includes this position in $OccL_j$ to update the initial partition. Finally, it increases the $NbInsBl$ by one. It repeats this process until $cardinal(OccL_j) = IC$.

2- The Block Removal

The algorithm, at this level, removes Bl_j from a random position, which is selected from $OccL_j$ randomly, in M when $cardinal(OccL_j) > IC$. Also, it deletes this position from $OccL_j$ and updates the initial partition according to the resulting changes in the other lists when influenced. Finally, it inserts the index j, if it does not figure, of Bl_j in the suppression list and the position of removal. It repeats this process until $cardinal(OccL_j) = IC$.
The general algorithm updates the secret key simultaneously with the initial partition during the encryption process to create the final partition that represents the ciphertext.

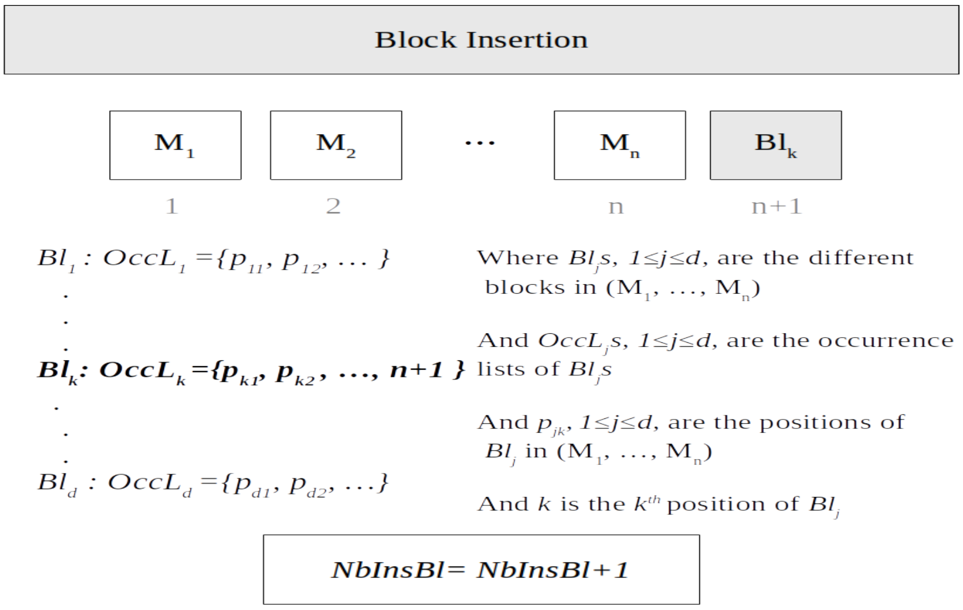

Block Insertion
M1
M2
...
Mn
Blk
1
2
n
n+1
Bl1 : OccL1 ={p11, p12, ... }
Blk: OccLk ={pk1, pk2, ..., n+1 }
Bld : OccLd ={pd1, pd2, ...}
Where Bljs, 1≤j≤d, are the different blocks in (M1, ..., Mn)
And OccLjs, 1≤j≤d, are the occurrence lists of Bljs
And pjk, 1≤j≤d, are the positions of Blj in (M1, ..., Mn)
And k is the kth position of Blj
NbInsBl= NbInsBl+1

Figure 2.4: Block Insertion

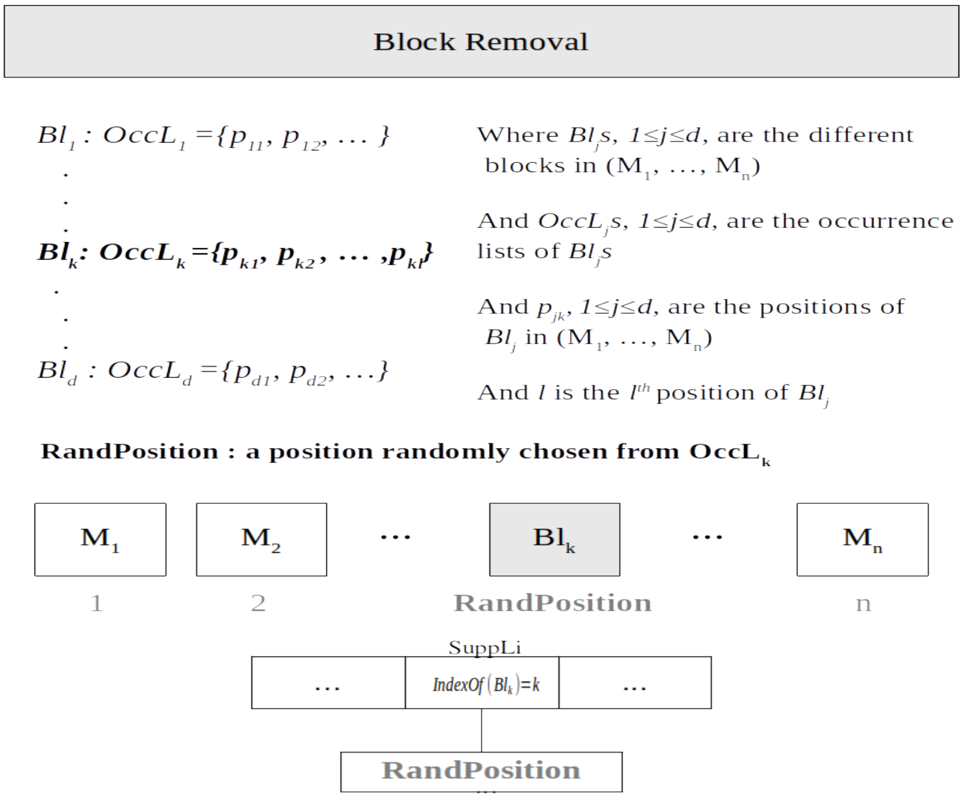

Block Removal
Bl1 : OccL1 ={p11, p12, ... }
Blk: OccLk ={pk1, pk2, ... ,pkl}
Bld : OccLd ={pd1, pd2, ...}
Where Bljs, 1≤j≤d, are the different blocks in (M1, ..., Mn)
And OccLjs, 1≤j≤d, are the occurrence lists of Bljs
And pjk, 1≤j≤d, are the positions of Blj in (M1, ..., Mn)
And l is the lth position of Blj
RandPosition : a position randomly chosen from OccLk
M1
M2
...
Blk
...
Mn
1
2
RandPosition
n
SuppLi
...
IndexOf(Blk)=k
...
RandPosition

Figure 2.5: Block Removal

The secret key sk is expressed as follows:
$sk = \{\{$ s$\}, \{$ NbInsBl$\}, \{$ IndexOf(Bl_j)$\rightarrow$ { PosOf(Bl_j)},...,IndexOf(Bl_k)$\rightarrow$ { PosOf(Bl_k)}}, π}.
Where $IndexOf(Bl_j) = j$ and $Pos(Bl_j)$ is the list of positions from where Bl_j was removed ($1 \leq j \leq d$, $1 \leq k \leq d$).

Algorithm 1 displays the PCS encryption algorithm. The Suppression-List and π are initially empty.

Algorithm 1 PCS Encryption Algorithm

```
Input: The binary message M
Output: The ciphertext C and the secret key sk
Begin
$s \leftarrow random(2, 16)$                                  ▷ random integer $2 < s \leq 16$
$NbInsBl \leftarrow 0$
$sk \leftarrow \{\{s\}, \{NbInsBl\}, SuppLi, \pi\}$
$M' \leftarrow splitIntoBlocks(M, s)$                         ▷ pre-processing step
$n \leftarrow sizeOf(M')$
$d \leftarrow nbDiffBl(M')$
$PlainPartition \leftarrow toPartition(M')$
$CipherPartition \leftarrow PlainPartition$
$ListOfBlocks \leftarrow diffBlocks(M')$                      ▷ $Bl_1$, ..., $Bl_d$
$IC \leftarrow computeIdealCardinality(n, d)$  ▷ Ideal cardinality computation step
for $0 < j \leq d$ do
    while $Cardinal(OccL_j) < IC$ do                          ▷ Block Insertion
        $M' \leftarrow blockInsertion(Bl_j, M', NbInsBl, sk)$
    end while
    while $Card(OccL_j) > IC$ do                              ▷ Block Removal
        $M' \leftarrow blockRemoval(Bl_j, randPosition(OccL_j), M', sk, SuppLi)$
    end while
end for
$C \leftarrow M'$
$\pi \leftarrow generatePermutation(PlainPartition, CipherPartition, sk)$
End
```

Figure 2.6 summarize the encryption process detailed before.

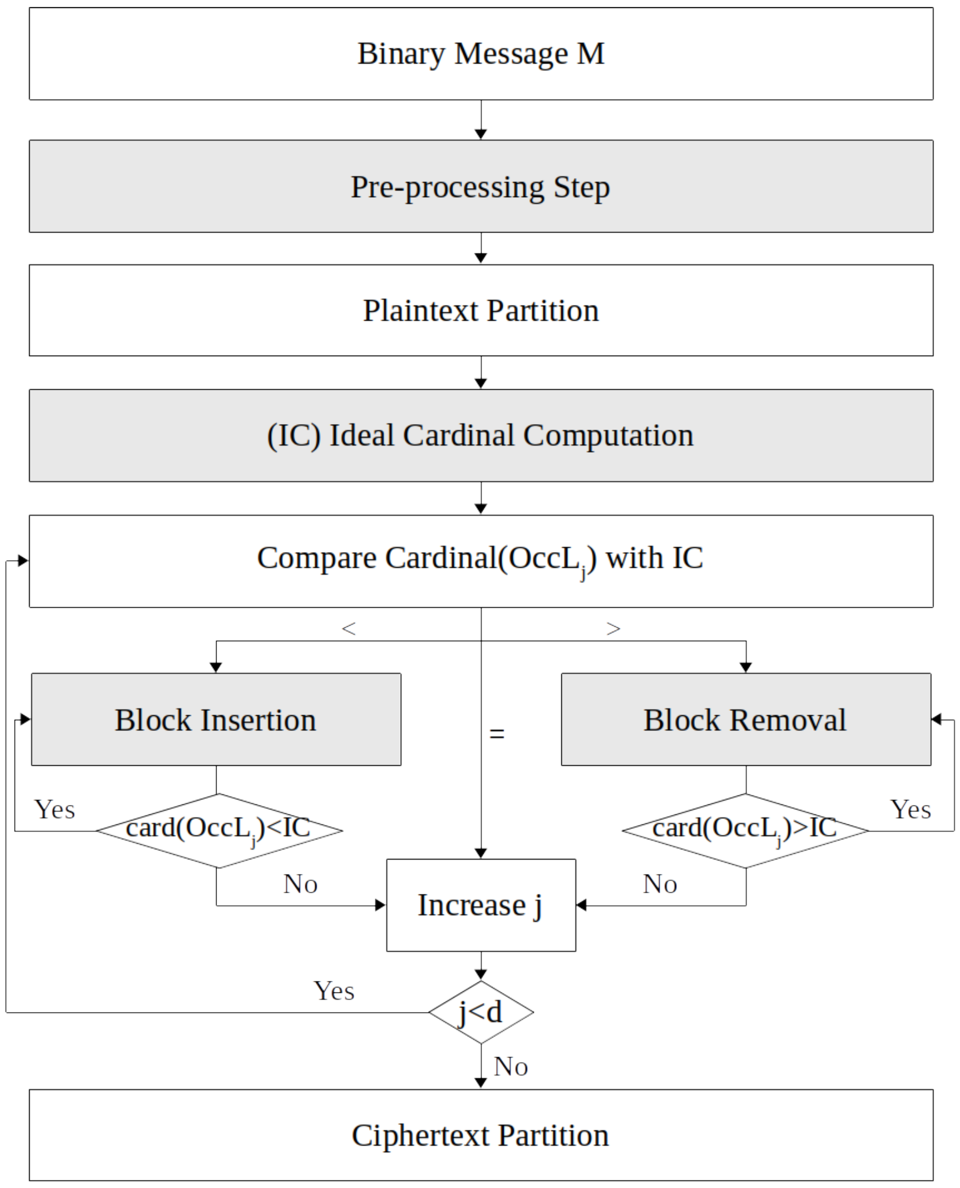


Figure 2.6: PCS Encryption

2.4 The Partition Ciphering System (PCS) Decryption

To decrypt, the PCS process through two main phases: a pre-processing one that splits the ciphertext into blocks of size s. And the second one that reverses the actions performed during the encryption algorithm.

2.4.1 The PCS Decryption Pre-processing Step

Let C be the ciphertext and $sk = \{s, NbInsBl, SuppLi, \pi\}$ the secret key. The first element of the secret key s is used by the PCS decryption algorithm to split C into s-bit blocks. Afterward, from the result of the split, the list of distinct blocks in C *LiDistBl* is produced. This list serves in the next step to decide which block is going to be inserted back or removed.

2.4.2 The PCS Decryption Step

At this level, the PCS decryption algorithm applies the reverse operations of the third step of PCS encryption operations (i.e., It inserts the removed blocks and removes the inserted ones).
At first, it uses LiDistBl, SuppLi, and π to insert each removed block in the positions of removal, where SuppLi is iterated starting from the last index. Figure 2.7 details how the insertion is processed. Finally, it removes a block from the last position NbInsBl times. Figure 2.8 displays the removal process.

Algorithm 2 describes the decryption algorithm.
Figure 2.9 Displays the PCS decryption process.

2.5 Combination of PCS with SEC extension to binary blocks

2.5.1 SEC extension to binary blocks

The SEC (Symmetrical Evolutionist-based Ciphering) is an evolutionary encryption scheme. To achieve the best security against the frequency analysis of the plaintext characters, the SEC was extended for application over binary blocks instead. In the SEC extension to binary blocks, an individual is a vector of size m, and genes are the lists of positions.

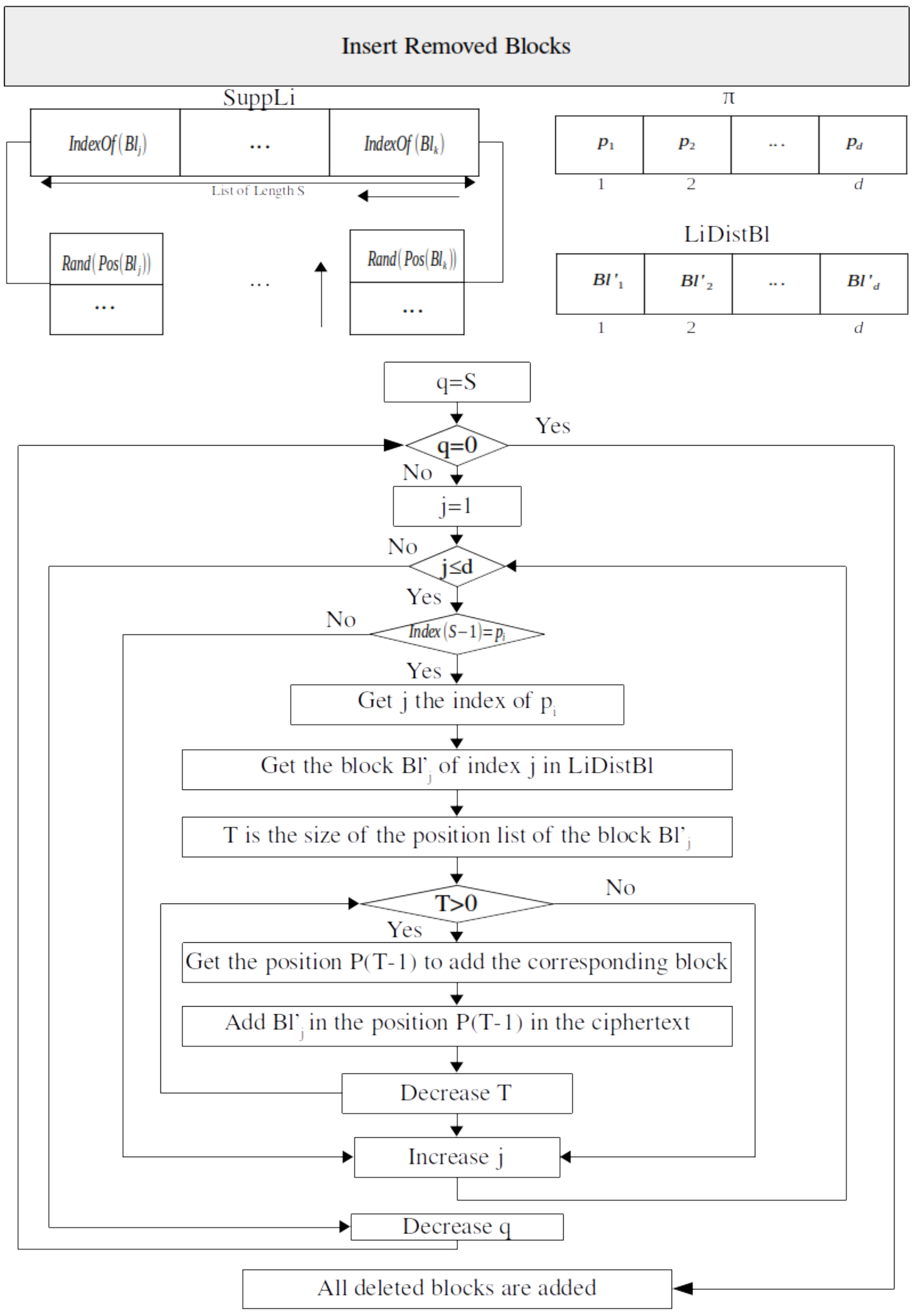

Insert Removed Blocks
SuppLi
IndexOf(Bl_j)
...
IndexOf(Bl_k)
List of Length S
Rand(Pos(Bl_j))
Rand(Pos(Bl_k))
π
P1
P2
Pd
1
2
d
LiDistBl
Bl'1
Bl'2
Bl'd
q=S
q=0
Yes
No
j=1
j≤d
Index(S−1)=p_i
Get j the index of p_i
Get the block Bl'_j of index j in LiDistBl
T is the size of the position list of the block Bl'_j
T>0
Get the position P(T-1) to add the corresponding block
Add Bl'_j in the position P(T-1) in the ciphertext
Decrease T
Increase j
Decrease q
All deleted blocks are added

Figure 2.7: Insert removed blocks

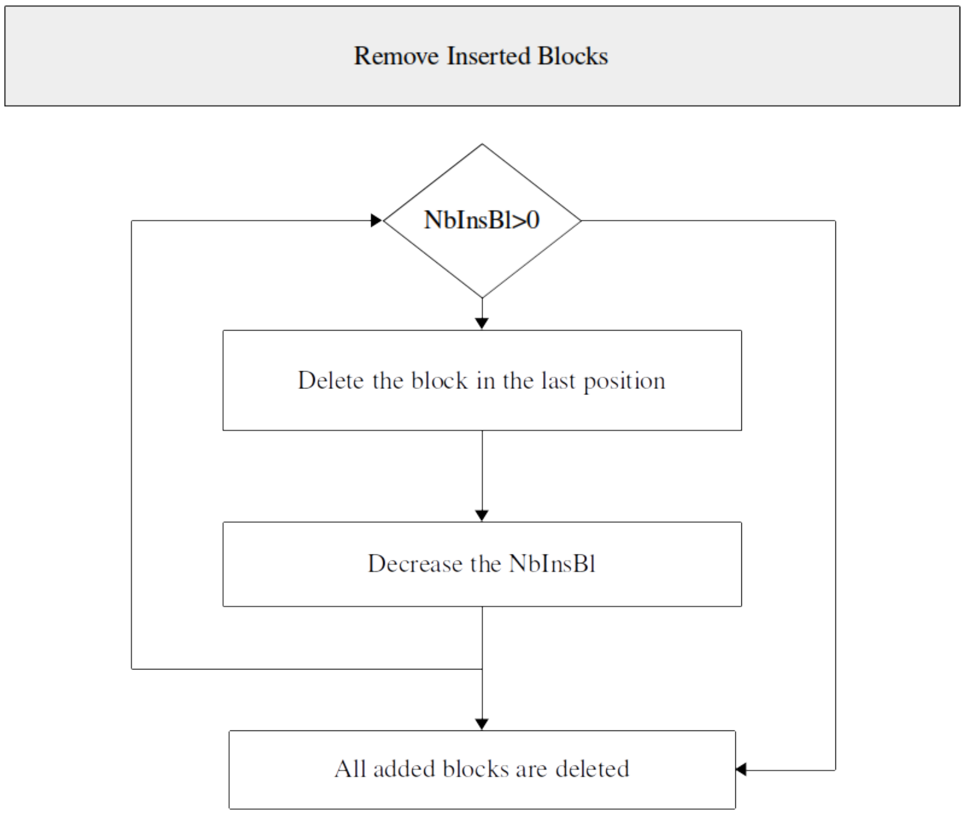

Remove Inserted Blocks
NbInsBl>0
Delete the block in the last position
Decrease the NbInsBl
All added blocks are deleted

Figure 2.8: Remove inserted blocks

Algorithm 2 PCS Decryption Algorithm

```
**Input**: The Ciphertext C and the secret key sk
**Output**: The plaintext M
**Begin**
$C \leftarrow divideIntoBlocks(C, s)$                    ▷ pre-processing
$LiDistBl \leftarrow distinctBlocks(C)$
**for** $0 < i \leq sizeOf(SuppLi)$ **do**               ▷ insert removed blocks
    $Insert(\pi, LiDistBl, SuppLi, C)$
**end for**
**while** $NbInsBl > 0$ **do**                           ▷ remove inserted blocks
    $C \leftarrow RemoveFromLast(C)$
    $NbInsBl \leftarrow NbInsBl - 1$
**end while**
$M \leftarrow C$
**End**
```

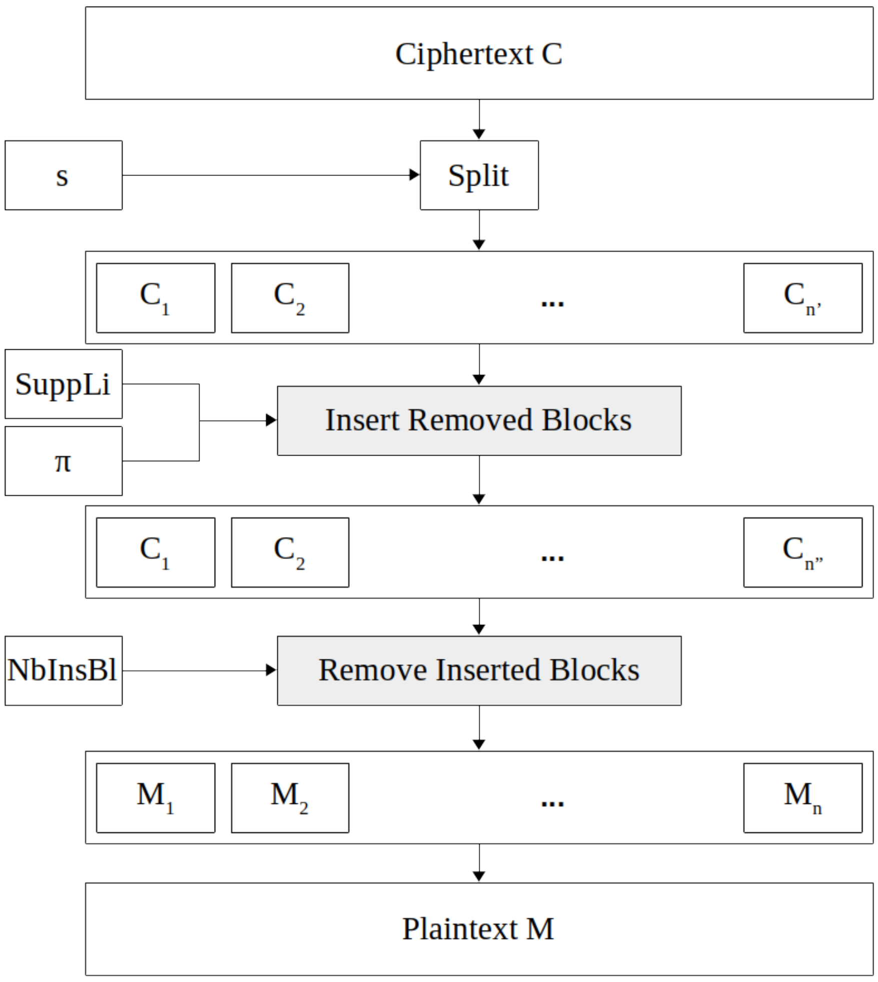


Figure 2.9: PCS Decryption

Encryption

Given an arbitrary length binary message, the SEC extension to binary blocks encryption algorithm firstly splits it into blocks of size $k \geq 2$. Then, it constructs the original chromosome that comprises the blocks together with their occurrences lists.

$(Bl_1, OccL_1)$	$(Bl_2, OccL_2)$	...	$(Bl_d, OccL_d)$

Afterward, q permutations of these lists are applied to the original chromosome to generate the initial population of q potential solutions. These latter are evaluated adopting the evaluation function denoted by F(X_j)= $\sum_{i=1}^{d}$ |card$(OccL_{ji})$- card$(OccL_i)$|, where $0 \leq j \leq q$ (d: number of different blocks in the plaintext). Accordingly, the best individuals are selected using the roulette wheel method, where the control function tries to eliminate individuals of minor change of genes compared to original-ch. Next, the MPX crossover operates on the selected individuals with a rate ranging from 60% to 100%. Afterward, the individuals produced by the crossing undergo the transposition mutation, which swaps randomly two genes, with a rate ranging from 0.1% to 5% to generate the next population. The steps from evaluation to the mutation are processed iteratively until the achievement of the stopping condition, which is the objective function convergence $0 \leq F(X_j) \leq 2 \times n$ (n: number of blocks in the plaintext). The final-Ch denotes the final solution, which is used in combination with the original-Ch to generate the symmetric key. It is a permutation of the set $\{1, 2, ..., m\}$. [57] exposes more details about the SEC binary extension algorithm. Figure 2.10 summarises the corresponding encryption mechanism.

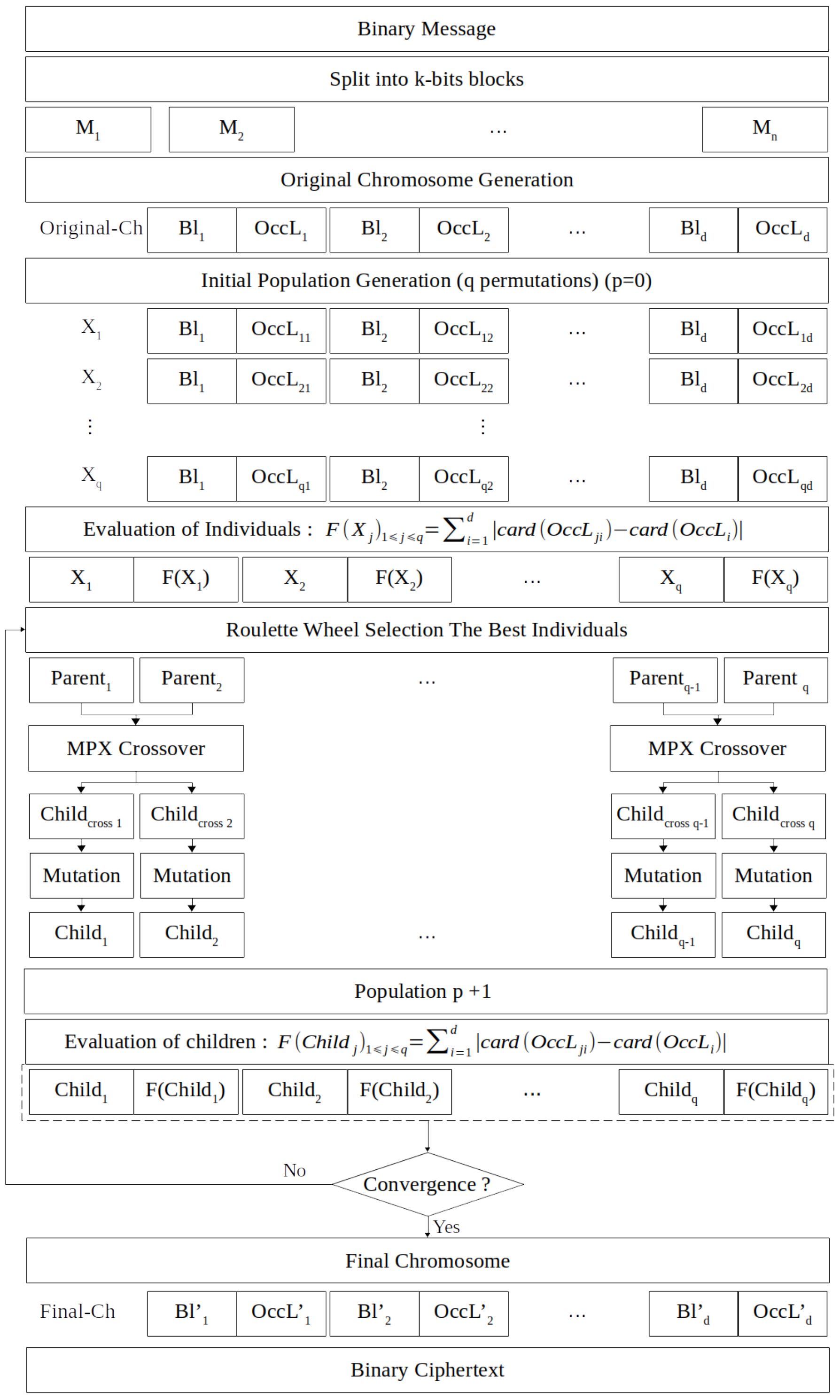

Binary Message
Split into k-bits blocks
M1 M2 … Mn
Original Chromosome Generation
Original-Ch Bl1 OccL1 Bl2 OccL2 … Bld OccLd
Initial Population Generation (q permutations) (p=0)
X1 Bl1 OccL11 Bl2 OccL12 … Bld OccL1d
X2 Bl1 OccL21 Bl2 OccL22 … Bld OccL2d
Xq Bl1 OccLq1 Bl2 OccLq2 … Bld OccLqd
Evaluation of Individuals : $F(X_j)_{1\leqslant j\leqslant q}=\sum_{i=1}^{d}|card(OccL_{ji})-card(OccL_i)|$
X1 F(X1) X2 F(X2) … Xq F(Xq)
Roulette Wheel Selection The Best Individuals
Parent1 Parent2 … Parentq-1 Parent q
MPX Crossover
Childcross 1 Childcross 2 Childcross q-1 Childcross q
Mutation
Child1 Child2 … Childq-1 Childq
Population p +1
Evaluation of children : $F(Child_j)_{1\leqslant j\leqslant q}=\sum_{i=1}^{d}|card(OccL_{ji})-card(OccL_i)|$
Child1 F(Child1) Child2 F(Child2) … Childq F(Childq)
No
Convergence ?
Yes
Final Chromosome
Final-Ch Bl'1 OccL'1 Bl'2 OccL'2 … Bl'd OccL'd
Binary Ciphertext

Figure 2.10: SEC Extension to Binary Blocks Encryption

Decryption

Given the binary ciphertext and the secret key comprising k and p the permutation produced in the encryption algorithm, the SEC extension to binary blocks decryption process starts by the split of the binary ciphertext into k-bits blocks. Next, it generates the Final-Ch by corresponding to each block its occurrence list. Afterward, the permutation p is useful to associate with each block its positions list to find the original chromosome Original-Ch, and consequently the plaintext. Figure 2.11 displays the decryption process of SEC extension to binary blocks.

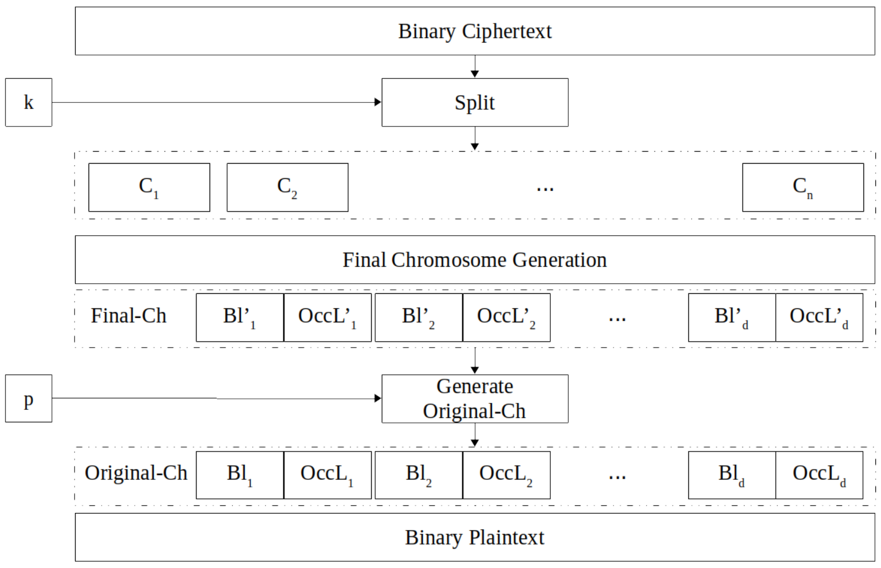


Figure 2.11: SEC Extension to Binary Blocks Decryption

2.5.2 SECEX-PCS multiple encryption

As stated before, the design of PCS makes the SEC results more satisfying by the combination of both systems. From Figure 2.12, given an arbitrary length binary message M, the encryption process starts with the SEC extension to the binary blocks encryption algorithm followed by the PCS encryption algorithm. The secret key $SK = \{K_{SECEX}, K_{PCS}\}$. Figure 2.13 illustrates the decryption process, which decrypts at first the ciphertext using the PCS decryption. Next, the resulting message goes through the SEC extension to the binary blocks decryption algorithm to get the plaintext.

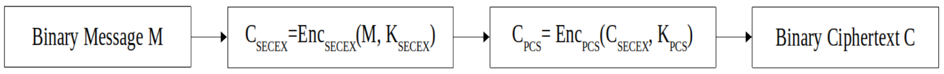

Binary Message M
C_{SECEX}=Enc_{SECEX}(M, K_{SECEX})
C_{PCS}= Enc_{PCS}(C_{SECEX}, K_{PCS})
Binary Ciphertext C

Figure 2.12: SEC Extension to Binary Blocks - PCS encryption

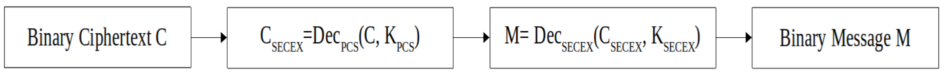

Binary Ciphertext C
C_{SECEX}=Dec_{PCS}(C, K_{PCS})
M= Dec_{SECEX}(C_{SECEX}, K_{SECEX})
Binary Message M

Figure 2.13: SEC Extension to Binary Blocks - PCS decryption

2.6 Experimental Results and Security Analysis

This section displays the PCS results of the statistical tests, namely Dieharder and NIST STS, the confusion and diffusion properties, and the performance and security compared to other systems, namely AES, DES, and 3DES.

2.6.1 Dieharder Tests

Dieharder is a battery of statistical tests, comprising 32 tests, designed by Brown to check the randomness and statistical properties of pseudo-random number generators and cryptographic primitives such as encryption systems, hash functions, and MACs. Thus, the PCS encryption algorithm generates a file that contains a 10Mb-bitstream. The battery algorithms compute the p-values. Accordingly, the sequence is said to be random or not. If the p-values are between $0 + \alpha$ and $1 - \alpha$, then the bitstream is indistinguishable from a random bitstream, where α is the significance level such that $\alpha = 0.005$. Table 2.1 displays the results of the statistical tests of PCS, SECEX and SECEX-PCS in comparison with AES, DES, and 3DES. It shows that PCS passed all the tests as the p-values are bounded by 0.2 and 0.9. Plus, the p-values of the sequence generated by AES are bounded by 0.05 and 1. To conclude, the PCS behavior is random. From Table 2.1, the p-values corresponding to SECEX are bounded by 0.07 and 0.96. The p-values of SECEX-PCS are bounded by 0.13 and 0.93. Consequently, both systems pass the tests. To conclude, the results of SECEX-PCS are better.

Table 2.1: Dieharder Tests of PCS, SECEX, and SECEX-PCS compared to AES, DES, and 3DES

Test	**P-values**					
	PCS	**AES**	**DES**	**3DES**	**SECEX**	**SECEX-PCS**
1	0.8625	0.0836	0.8133	0.6844	0.4089	0.9191
2	0.9571	0.0967	0.7043	0.0069	0.9937	0.4371
3	0.1402	0.7711	0.1952	0.0138	0.2241	0.6931
4	0.324	0.6936	0.8422	0.8119	0.7946	0.547
5	0.453	0.6593	0.3996	0.8619	0.073	0.4851
6	0.3559	0.7204	0.135	0.2125	0.8689	0.9231
7	0.0898	0.6363	0.4289	0.9598	0.2828	0.3692
8	0.2811	0.3142	0.7883	0.1243	0.0745	0.3191
9	0.9687	0.8797	0.7384	0.3628	0.8473	0.6549
10	0.7611	0.8451	0.7915	0.344	0.5826	0.9089
11	0.7733	0.8514	0.9016	0.3209	0.6881	0.8969
12	0.791	0.537	0.9916	0.2343	0.412	0.9019
13	0.2487	0.3863	0.2674	0.9371	0.5801	0.6786
14	0.9916	0.8732	**0.0007**	0.2897	0.3517	0.6727
15	0.1779	0.0058	0.2512	0.4485	0.3071	0.1332
16	0.7702	0.381	0.2296	0.4514	0.5596	0.5369
17	0.9093	0.863	0.4422	0.6496	0.5449	0.4591
18	0.4046	0.7107	0.2148	0.7392	0.5038	0.593
19	0.5431	0.6915	0.5303	0.4562	0.2487	0.8948
20	0.0704	0.4656	0.2151	0.3326	0.5767	0.3401
21	0.6388	0.5643	0.5482	0.5758	0.5286	0.592
22	0.4844	0.549	0.5006	0.56	0.6585	0.3864
23	0.4441	0.3475	0.2433	0.6842	0.3519	0.3857
24	0.4145	0.6588	0.6155	0.729	0.4371	0.6948
25	0.604	0.5363	0.6518	0.4936	0.5655	0.5568
26	0.1025	0.4934	0.3525	0.1255	0.0833	0.6017
27	0.2636	0.4758	0.0914	0.1132	0.2665	0.3721
28	0.8735	0.9448	0.2778	0.4307	0.0894	0.781
29	0.5212	0.4721	0.4319	0.3558	0.4331	0.2684
30	0.3727	0.709	0.2142	0.4891	0.4523	0.7374
31	0.6055	0.0507	0.915	0.2603	0.366	0.3292

2.6.2 Confusion and Diffusion Properties

This section presents the confusion and diffusion properties of PCS and AES. Shannons defined an encryption system satisfying these properties to be robust against statistical analysis [63]. AES was already studied before, and it has satisfying confusion and diffusion properties. The confusion property illustrates the complexity of the relationship between the key and the ciphertext. If the proportion of change observed in the ciphertext when changing one bit of the key is approximately 50%, then the confusion property is excellent. The diffusion property displays how the plaintext is related to the ciphertext. If almost 50% of the ciphertext bits change when a single bit in the plaintext is changed, then the diffusion property is good enough. The avalanche effect test if a system satisfies these properties. Figure 2.14 presents the diffusion property of PCS in comparison with AES. It shows that the rate of change in the output is around 50% for both systems. Consequently, in the PCS scheme, diffusion is accomplished.

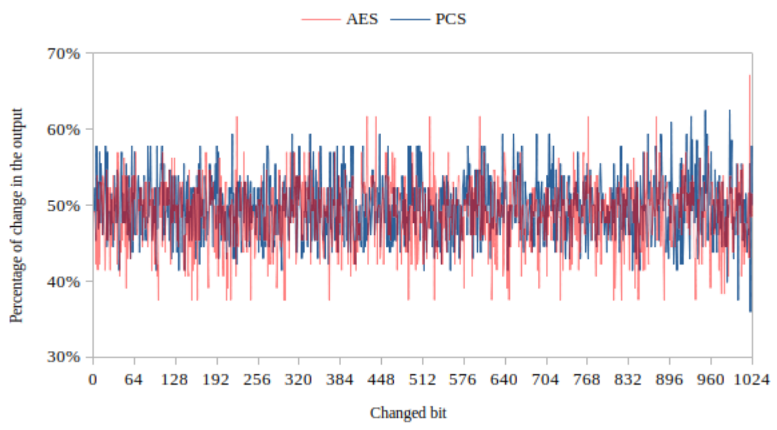


Figure 2.14: PCS Diffusion property

Figure 2.15 presents the PCS confusion property. Around 50% of the output bits transformed after the alteration of one bit in the plaintext.

From the statistical tests and the avalanche effect results, the conclusion is that the proposed system PCS had good randomness, confusion, and diffusion properties.

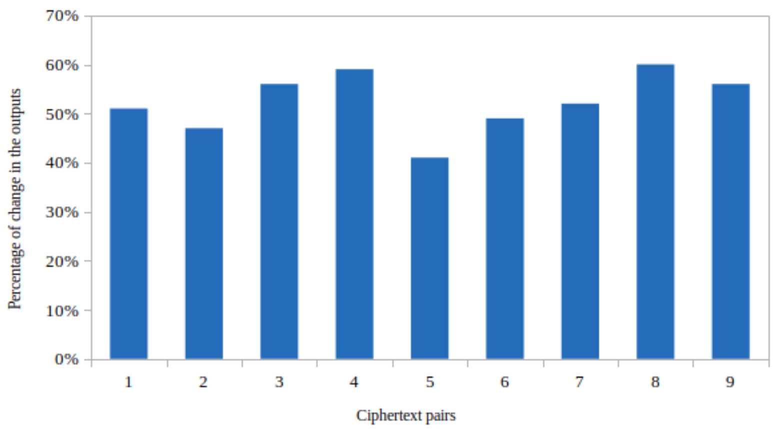


Figure 2.15: Diffusion property

2.6.3 Comparison of PCS with AES, DES and 3DES

Encryption and Decryption time

This part provides the time of encryption and decryption of PCS, AES, DES, and 3DES(Figure 2.16).

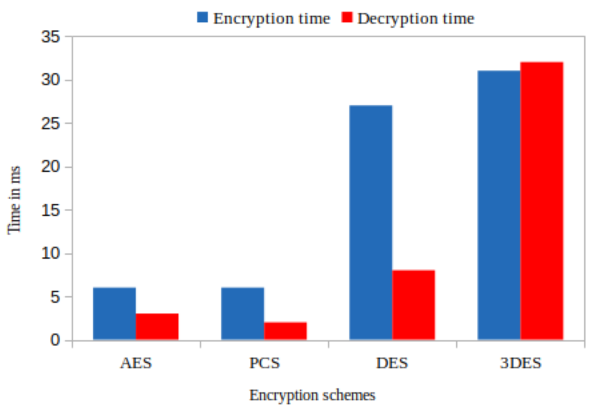


Figure 2.16: PCS Encryption and Decryption time compared to AES, DES, and 3DES

Figure 2.16 shows that DES takes more time to encrypt and decrypt than AES and PCS systems. Even though DES is vulnerable compared to 3DES, but 3DES takes more time than other cryptosystems. PCS and AES take an equivalent and shorter time to encrypt than DES and 3DES. Also, PCS takes less time than AES, DES, and 3DES.

Brute Force Attack

The adversary tries to test each possible key to obtain intelligible cleartext by transforming the ciphertext in this type of attack. Therefore, the security parameter that determines the security level guaranteed is the length of the key. When this latter is high, then the attack requires significant time and resources to find the proper key. The presence of a quantum computer can make this attack possible for some ciphers. According to the level of security wished, the symmetric key can take values from 128 to 256 bit. If the long term security is required without the presence of quantum computers, then it is recommended to use a 128-bit key. If the long term security is necessary even in the presence of quantum computers, then it is suggested to use a key of at least 256 bits. As there are three versions of AES (the 128 version, the 192 version, and the 256 version), the two levels could be insured. The secret key of PCS could be higher than 256 bit. Consequently, no attacker can get the secret of PCS to get the plaintext. Table 2.2 displays the security level of PCS, AES, DES, and 3DES.

Table 2.2: Brute Force Attack of AES, PCS, DES, and 3DES

Encryption schemes	AES-128	AES-192	AES-256	PCS	DES	3DES
Key length(b)	128	192	256	≥ 256	56	168
# possible keys	2^{128}	2^{192}	2^{256}	$\geq 2^{256}$	2^{56}	2^{168}

Frequency Analysis

This section presents the frequency analysis conducted to plaintext and its corresponding ciphertext. Figure 2.17 provides the results of this analysis. As mentioned previously, the motivation of PCS is to get a ciphertext having the blocks appearing with the same frequency. And figure 2.17 displays the difference between before and after encryption. In the other articles [58] to [62], the blocks frequency change, but without the concept of the same appearance frequency.

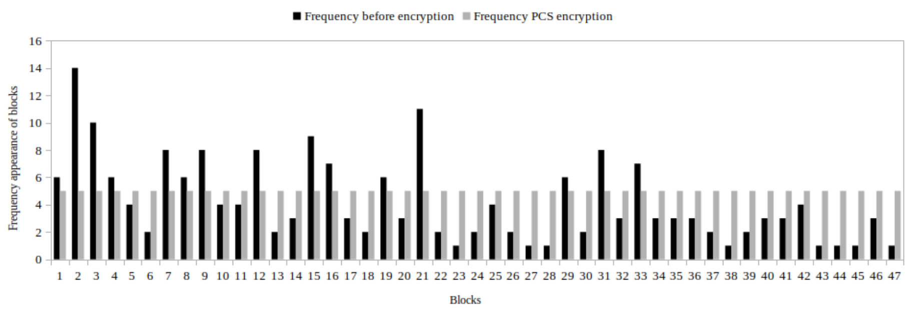


Figure 2.17: PCS Frequency Analysis

2.6.4 SECEX-PCS Frequency Analysis

This part investigates the frequency analysis of the different blocks in a message instance and after encryption with SECEX and PCS. Figure 2.18 displays the frequency change by comparing how SECEX influences the appearance of each block with the combination of SECEX with PCS for a message of size 2804 bits with kSecex=kPCS=6. As mentioned before, SECEx is limited to swapping the lists, while the combination makes all lists with the same cardinal, which is nine in this example. It provides robustness concerning the frequency cryptanalysis. Consequently, the SECEX-PCS results are better than the results of SECEX encryption.

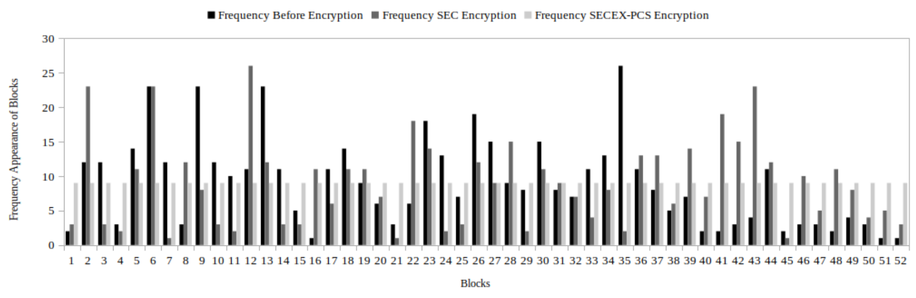


Figure 2.18: SEC Extension to Binary Blocks - PCS Frequency Analysis

2.7 Conclusion

This chapter provides a detailed description of the symmetric encryption scheme is provided, which is called the Partition Ciphering System (PCS) based on an adapted representation of the Equal Piles Problem (EPP) in combination with the study of SEC, a previously developed scheme. The motivation of this contribution is to design a robust encryption system resistant to frequency cryptanalysis. The Card-Partition problem, aiming to partition a set into subsets having the same cardinal, was proposed to withstand this attack. The idea of PCS was to construct a partition based on the proposed problem without using a genetic algorithm. The operations performed are simple, making the combination of PCS with other systems more interesting if frequency analysis resistance is required. The union of the SEC with PCS exhibits excellent resistance to this attack. Also, the results of the comparison of PCS with AES, DES, and 3DES are good enough to tell that PCS is secure. Regarding the Dieharder results, PCS has better results when compared to the other systems. Besides, the confusion and diffusion properties of PCS are suitable for a secure encryption system, as recommended by Shannon. Moreover, PSC provides a higher security level, which makes it robust against the brute force attack. Plus, it is resistant to statistical attacks like frequency cryptanalysis. To explore the impact of including new features, the second design extending this version is also proposed in the next chapter.

Chapter 3

Cellular Automata based Partition Ciphering System

3.1 Introduction

This chapter presents an extended version of PCS called Cellular Automaton-based Partition Ciphering System (CA-PCS) that includes new features ensuring high nonlinearity, high confusion and diffusion and a satisfying security level. The simplicity, the unpredictability, and the simple hardware and software implementations provided by cellular automata expand their applicability in the cryptography as well as other domains. CA-PCS comprises a hybrid cellular automaton, evolving multiple clock cycles to provide adequate cryptographic properties to strengthen the system. Afterward comes the insertion of missing blocks producing the balance in the output. Finally, an application of a random permutation improving the diffusion property follows. The CA cryptographic properties make the security level study provided by the system simple. Precisely, the nonlinearity, algebraic degree, balancedness, resiliency, and correlation immunity are computed and analyzed. Balancedness and high nonlinearity prevent attacks like the linear and differential cryptanalysis. High correlation immunity and resiliency provide better confusion property and make correlations attacks challenging. A high algebraic degree makes the system stronger against algebraic attacks. A comparison of CA-PCS with AES and PCS concerning the randomness, security, and performance follows. The rest of this chapter is organized as follow: Section 3.2 details the related work. Next, the CA-PCS encryption and decryption algorithms are presented, respectively, in section 3.3 and 3.4. Afterward, section 3.5 provides the results and security analysis of the proposed scheme.

3.2 Related Work

Cellular automata were first applied to cryptography by Wolfram [64]. He designed a pseudo-random number generator (PRNG) and a stream cipher using the nonlinear rule 30. In the last decade, Das et al. developed a one-dimensional programmable CA-based block cipher, where programmable CA depends on a specific signal called the control signal. According to which, in the model proposed, the CA evolves using one of the linear rules 51, 153, 195 for each different cell [65]. As well, Bhaumik [66] and Roy [67] proposed one-dimensional uniform CA-based primitives. In [66], Bhaumik designed a CA-based diffusion layer for a specific SPN block cipher using the ruleset 150, 150, 90, 150, 90, 150, 90, 150. Roy made use of the rules 51, 153,195 to produce a group of CAs that serve in the different steps of the system proposed[67]. Also, Mehta and Bouchkaren involved non-uniform one-dimensional reversible CAs in the design of block ciphers [68] [69]. In [68], Mehta involved the rulesets 51, 51, 153, 153, 153, 153, 51, 51, 51, 153, 153, 153, 153, 153, 153, 51, 195, 195, 195, 195, 51, 51, 51, 51, and 153, 153, 153, 153, 51, 51, 51, 51 in the encryption design. The general design structure [69] comprises four steps, and the CA is involved in the shift transformation using the rules deduced from the plaintext bytes[69]. Bouchkaren and Faraoun employed two-dimensional CA in their designs [70][71]. In [70], the encryption algorithm generates the reversible rules to provide a decipherable ciphertext. In [71], Faraoun used a genetic algorithm to find the best rules according to the strict avalanche criterion. Moreover, image encryption systems involving two-dimensional CAs could be found in [72], where Li et al. proposed their system based on balanced reversible CA so that randomness properties are interesting. Also, Niyat et al. used the CA rules 165, 105, 90, 150, 153, 101, 30, 86 to generate the key image together with a chaotic-mapping to design a strong system[73].

CA-PCS design

3.3 CA-PCS Encryption Algorithm

Three building blocks are involved in the development of the CA-PCS encryption system. A CA-based building block, the blocks insertion mechanism which is going through a random behavior, and a random permutation π uniformly generated.

3.3.1 CA Evolution

A hybrid one-dimensional CA, which makes use of the ruleset {90, 150, 30, 180, 45, 90, 150, 30}, combining linear and nonlinear rules. Linear rules produce high cycle length and better diffusion, and nonlinear rules provide more confusion and resistance to linear and differential attacks. The ruleset was determined by following the recommendations of Chakraborty and Bahtacharjee[19]. Balanced nonlinear rules having random behavior in addition to linear rules with high correlation immunity construct a ruleset adequate for cryptographic applications. Figure 3.1 displays one iteration of the CA evolution according to the proposed ruleset.

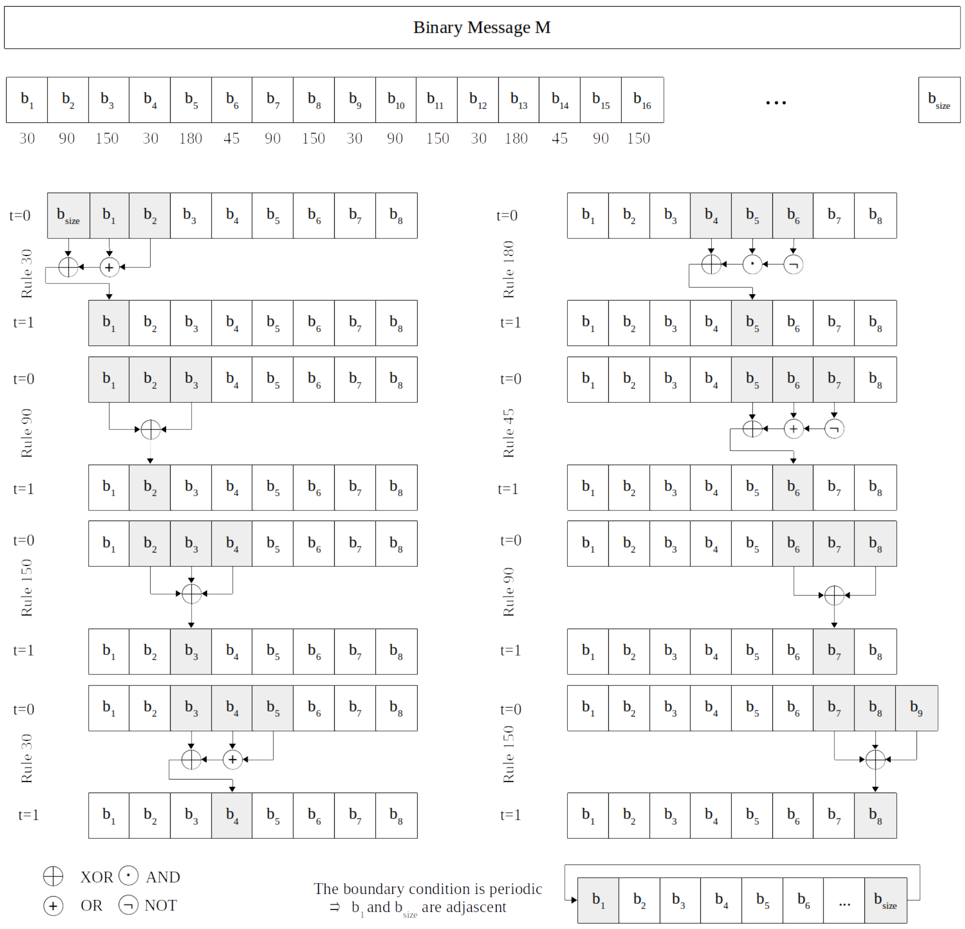

Binary Message M
b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 ... bsize
30 90 150 30 180 45 90 150 30 90 150 30 180 45 90 150
t=0
t=1
Rule 30
Rule 90
Rule 150
Rule 180
Rule 45
XOR
AND
OR
NOT
The boundary condition is periodic
⇒ b1 and bsize are adjascent

Figure 3.1: CA Evolution step

3.3.2 Blocks Insertion

The blocks insertion mechanism starts with the representation of the previous step's output as a partition. Next, the computation of the ideal cardinal, followed by the insertion of particular blocks to get equilibrium in the result of this step. First of all, the output of the CA building block is divided into s-bits blocks, such that s is a random number in the range [2,16]. An occurrence list $OccL_j$ is associated with each block Bl_j constructing the partition ($1 \leq j \leq d$, d is the number of different blocks in the output of the previous step). Then, the calculation of the Ideal Cardinal IC follows the formula:

$$IC = max\{Cardinal(OccL_1), Cardinal(OccL_2), ..., Cardinal(OccL_d)\}$$

Afterward, if the cardinal of Bl_j is inferior to IC, then Bl_j is added at a random position P_{jk} between 1 and the length of the result of the CA phase, such that k is in the range $[1, IC - Cardinal(OccL_j)]$. Next, InsertedBlocksPositionsList undergoes an update by the insertion of P_{jk}.

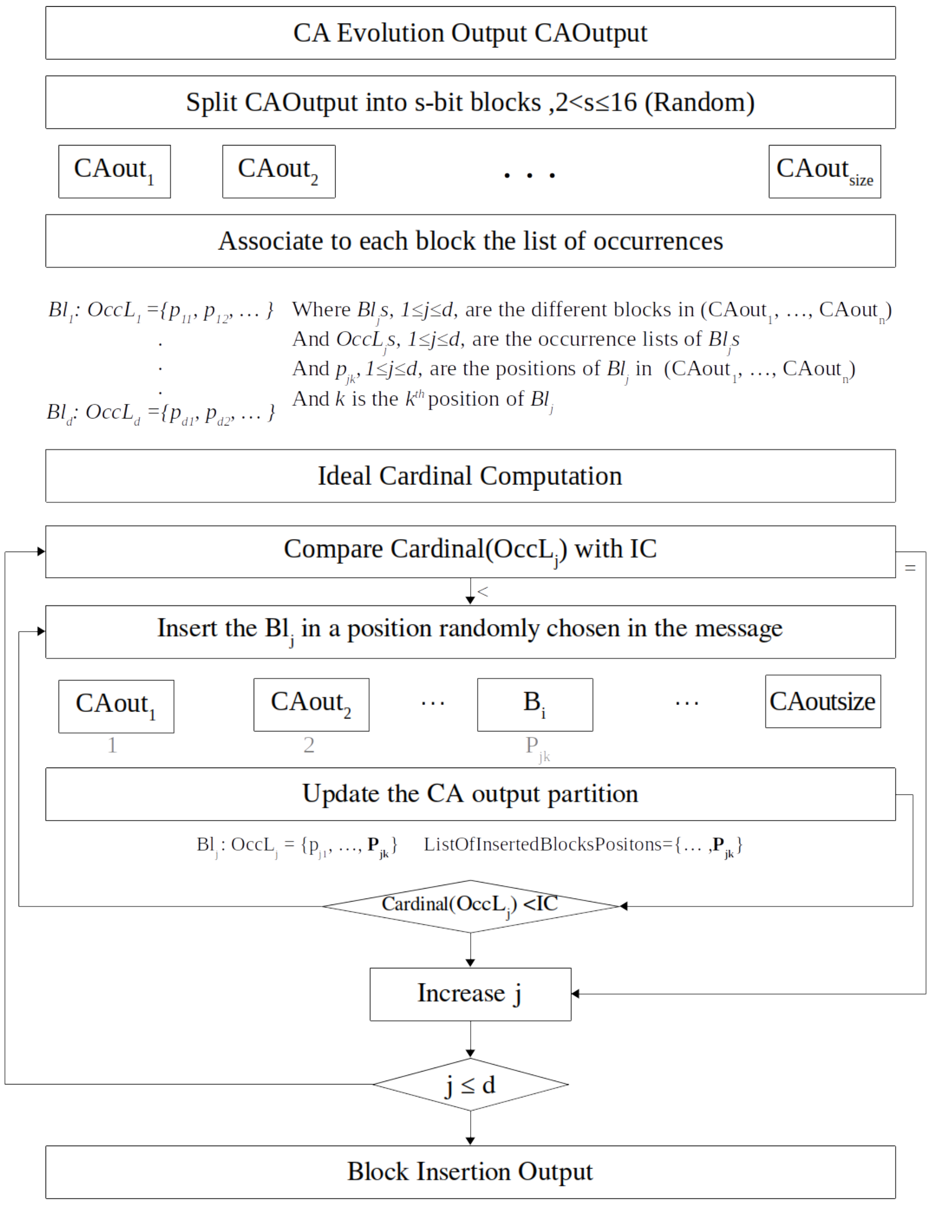

CA Evolution Output CAOutput
Split CAOutput into s-bit blocks ,2<s≤16 (Random)
CAout1
CAout2
. . .
CAoutsize
Associate to each block the list of occurrences
Bl1: OccL1 ={p11, p12, … }
Bld: OccLd ={pd1, pd2, … }
Where Bljs, 1≤j≤d, are the different blocks in (CAout1, …, CAoutn)
And OccLjs, 1≤j≤d, are the occurrence lists of Blj s
And pjk, 1≤j≤d, are the positions of Blj in (CAout1, …, CAoutn)
And k is the kth position of Blj
Ideal Cardinal Computation
Compare Cardinal(OccLj) with IC
=
<
Insert the Blj in a position randomly chosen in the message
CAout1
CAout2
Bi
CAoutsize
1
2
Pjk
Update the CA output partition
Blj : OccLj = {pj1, …, Pjk}
ListOfInsertedBlocksPositons={… ,Pjk}
Cardinal(OccLj) <IC
Increase j
j ≤ d
Block Insertion Output

Figure 3.2: Blocks Insertion Step

3.3.3 Permutation

This phase makes use of a random permutation to substitute the set of occurrence lists $\{OccL_1, OccL_2, ..., OccL_d\}$. Concretely, this step aims to change the occurrence lists of blocks $OccL_j$s. More formally, it is defined by π: $S \rightarrow S$ such that S is a d-elements set. There are $d!$ distinct permutations of $\{OccL_1, OccL_2, ..., OccL_d\}$. For instance, if d=8, then $\pi : \{OccL_1, OccL_2, OccL_3, OccL_4, OccL_5, OccL_6, OccL_7, OccL_8\} \rightarrow \{OccL_2, OccL_4, OccL_1, OccL_6, OccL_3, OccL_7, OccL_8, OccL_5\}$. Accordingly, $OccL_1 \rightarrow OccL_2$, $OccL_2 \rightarrow OccL_4$, $OccL_3 \rightarrow OccL_1$, $OccL_4 \rightarrow OccL_6$, $OccL_5 \rightarrow OccL_3$, $OccL_6 \rightarrow OccL_7$, $OccL_7 \rightarrow OccL_8$, $OccL_8 \rightarrow OccL_5$. Consequently, the positions of Bl_2 will comprise Bl_1, those of Bl_4 will contain Bl_2, and so on.

3.3.4 Key generation

The CA-PCS encryption algorithm generates the secret key, which is a session key, including four elements: $SK = \{$ CAKey, s, InsertedBlocksPositionsList, PermutaionKey $\}$. The CAKey is the plaintext M XORed with the input of the insertion blocks phase $CAKey = M \oplus InsertionBlockInput$. The second element s is the size of the blocks. The third element is the list of the positions of blocks insertion InsertedBlocksPositionsList. The fourth element is the PermutationKey, which is the ciphertext C XORed with the input of the permutation phase $PermutationKey = C \oplus PermutationInput$.

Figure 3.3 summarizes the encryption process of CA-PCS.

3.4 CA-PCS Decryption Algorithm

To decrypt a ciphertext C, that CA-PCS decryption algorithm processes as follows, having the secret key SK: $SK = \{$ CAKey, s, InsertedBlocksPositionsList, PermutationKey $\}$ It starts with the XOR of the ciphertext with the fourth element of SK, PermutationKey, to get the PermutationInput.

$$PermutationInput = C \oplus PermutationKey$$

Next, it splits the PermutationInput into s-bit blocks. Then, to get BlocksInsertionInput, it removes blocks from the positions figuring at the list of inserted blocks positions InsertedBlocksPositionsList, that is iterated starting with the last element as specified in algorithm 4. At last, the XOR oper-

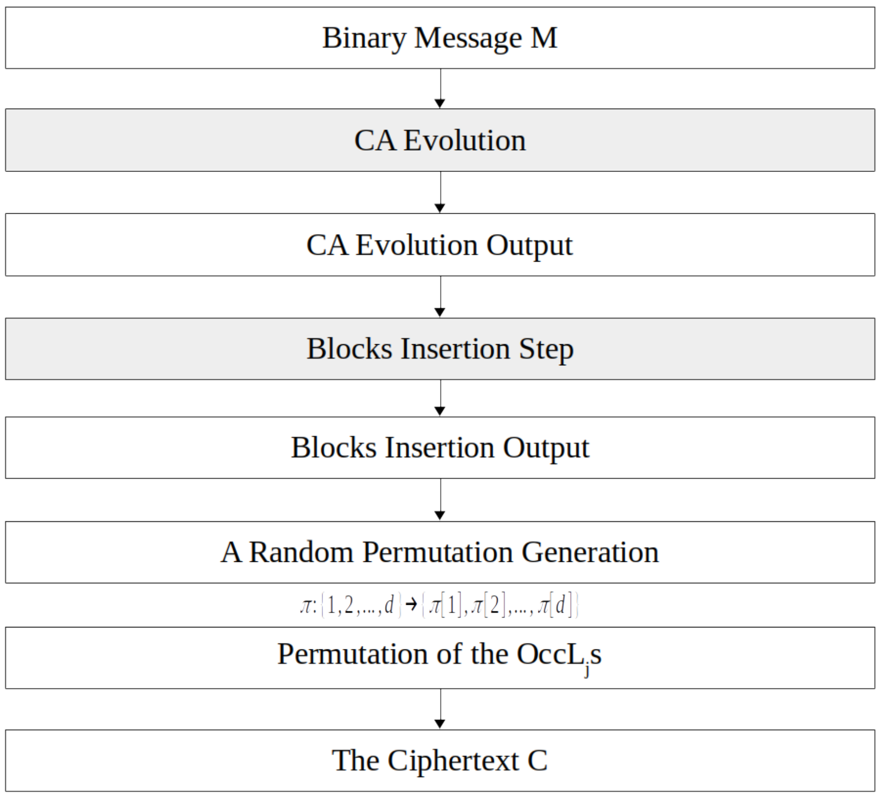

Binary Message M
CA Evolution
CA Evolution Output
Blocks Insertion Step
Blocks Insertion Output
A Random Permutation Generation
$\pi:\{1,2,\ldots,d\}\rightarrow\{\pi[1],\pi[2],\ldots,\pi[d]\}$
Permutation of the OccL$_j$s
The Ciphertext C

Figure 3.3: CA-PCS Encryption

Algorithm 3 CA-PCS Encryption Algorithm

```
**Input**: The message M
**Output**: The ciphertext C and the secret key SK
**Begin**
$s \leftarrow random(2, 16)$    ▷ random integer $2 < s \leq 16$
$ruleSet \leftarrow \{30, 90, 150, 30, 180, 45, 90, 150\}$
$M \leftarrow CAEvolution(M, ruleSet, 64)$
$CAKey \leftarrow M \oplus InsertionBlocksInput$
$PermutationInput \leftarrow DivideIntoBlocks(InsertionBlocksInput, s)$
$n \leftarrow sizeOf(PermutationInput)$
$m \leftarrow NumberOfDifferentBlocks(PermutationInput)$
$Partition \leftarrow ToPartition(PermutationInput)$
$ListOfBlocks \leftarrow DifferentBlocks(PermutationInput)$    ▷ $\{Bl_1, ..., Bl_d\}$
$IC \leftarrow ComputeIdealCardinality(Partition)$
**for** $1 \leq j \leq d$ **do**
    **while** $Cardinal(OccL_j) < IC$ **do**
        $PermutationInput \leftarrow insert(Bl_j,$ PermutationInput, randomPosition,$OccL_j$, $ListOfInsertedBlocksPositions)$
    **end while**
**end for**
$\pi \leftarrow generateRandomPermutation(\{1, 2, ..., d\})$
$Ciphertext \leftarrow applyPermutation(PermutationInput, permutation)$
$PermutationKey \leftarrow Ciphertext \oplus PermutationInput$
$SK \leftarrow \{CAKey, s, InsertedBlocksPositionsList, PermutationKey\}$
**End**
```

ation is applied to the BlocksInsertionInput and the CAKey to obtain the plaintext M.

$$M = BlocksInsertionInput \oplus CAKey$$

Figure 3.4 and Algorithm 4 details the decryption process of CA-PCS.

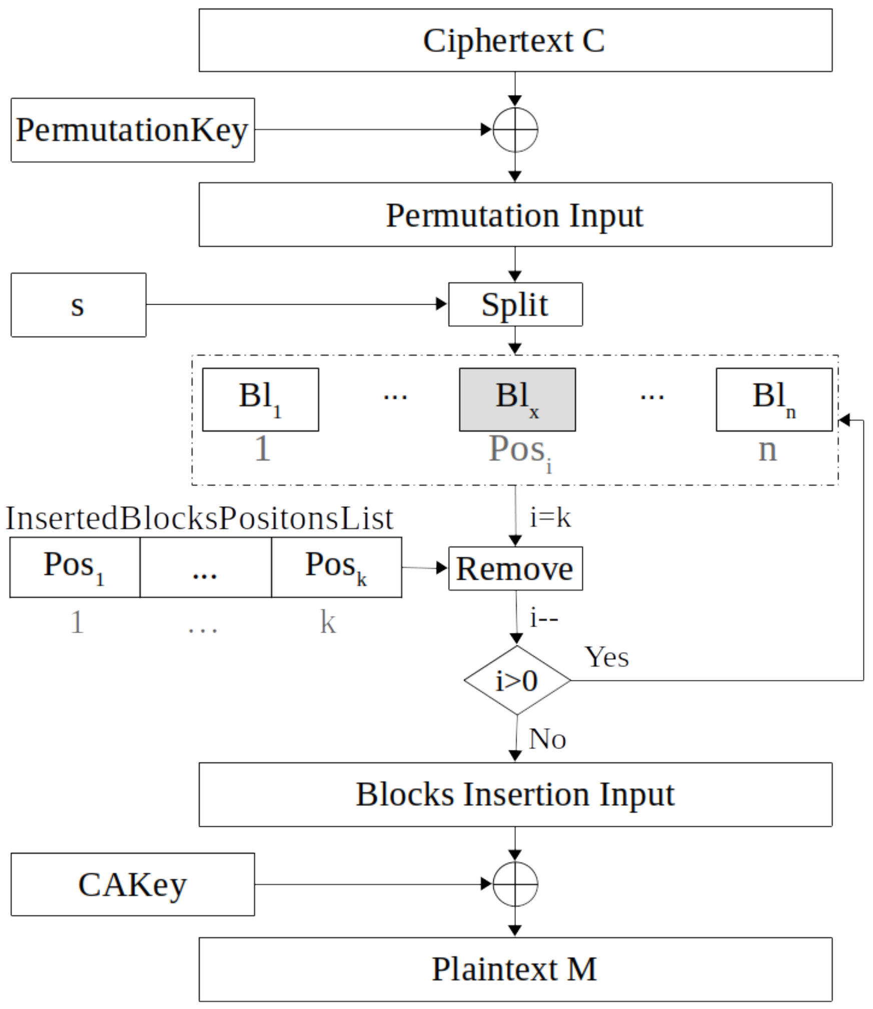


Figure 3.4: CA-PCS Decryption

Algorithm 4 Decryption algorithm

```
**Input**:The secret key SK and the ciphertext C
**Output**: The message M
**Begin**
$PermutationInput \leftarrow C \oplus PermutationKey$
$SplitPermutationInput \leftarrow DivideIntoBlocks(PermutationInput, s)$
**for** $k$ from $sizeOf(InsertedBlocksPositionsList)$ to 1 **do**
    $BlocksInsertionInput \leftarrow Remove(SplitPermutationInput, InsertedBlock\ PositionsList[k])$
**end for**
$M \leftarrow BlocksInsertionInput \oplus CAKey$
**End**
```

3.5 Results & Security Analysis

This section displays the statistical tests and the confusion and diffusion properties of CA-PCS compared to the AES.

Dieharder Test

This part investigates the randomness properties of CA-PCS using the dieharder tests battery. Statistical based attacks are hard to succeed if the bahevior of CA-PCS is indistinguishable from random functions. The CA-PCS, PCS, and AES encryption algorithms are processed repetitively to generate three files of 10Mb. Then, the average p-values of the sequences are measured. From table 3.1, in the case of CA-PCS, the p-values are between 0.2342 and 0.92. The PCS corresponding p-values range from 0.10 to 0.91. For AES, the p-values are bounded by 0.005 and 0.95. As a result, PCS, CA-PCS, and AES pass the test as all the values are in the range $[\alpha, 1-\alpha]$, where α is equal to 0.005. Nevertheless, the results of CA-PCS are better in comparison to PCS and AES. Since the difference between the lower bound p-value of CA-PCS and α is 0.2292, and between the upper bound and 1-α is 0.082. While for PCS, 0.0975 is the difference between the lower bound and α, and 0.0857 is the difference between the upper bound and 1-α. Also, for AES, the difference between the lower bound and α is 0.0008, and the difference between the upper bound and 1-α is 0.0502.

Table 3.1: Dieharder Results of CA-PCS, AES, and PCS

Tests	**CA-PCS P-values**	**AES P-values**	**PCS P-values**
Diehard birthdays	0.5357	0.0836	0.8625
Diehard operm5	0.4946	0.0967	0.8971
Diehard rank 32x32	0.5887	0.7711	0.1402
Diehard rank 6x8	0.7192	0.6936	0.3240
Diehard bitstream	0.4615	0.6593	0.4530
Diehard opso	0.5559	0.7204	0.3559
Diehard oqso	0.5092	0.6363	0.1898
Diehard dna	0.5686	0.3142	0.2811
Diehard count 1s str	0.4114	0.8797	0.8988
Diehard count 1s byt	0.6995	0.8451	0.7611
Diehard parking lot	0.2622	0.8514	0.773
Diehard 2dsphere	0.4555	0.5370	0.7910
Diehard 3dsphere	0.6735	0.3863	0.2487
Diehard squeeze	0.6888	0.8732	0.7991
Diehard sums	0.9130	0.0058	0.1779
Diehard runs	0.2342	0.3810	0.7702
Diehard craps	0.7063	0.8630	0.9093
Marsaglia tsang gcd	0.6682	0.7107	0.4046
Sts monobit	0.5815	0.6915	0.54319
Sts runs	0.4394	0.4656	0.1070
Sts serial	0.6616	0.5643	0.6388
Rgb bitdist	0.6689	0.5724	0.4844
Rgb minimum distance	0.5515	0.3475	0.4441
Rgb permutations	0.6639	0.6588	0.4145
Rgb lagged sum	0.5074	0.5363	0.6067
Rgb kstest test	0.2840	0.4934	0.1025
dab bytedistrib	0.5920	0.4758	0.2636
dab dct	0.8842	0.9448	0.8735
dab filltree	0.4757	0.4721	0.5212
dab filltree2	0.8987	0.7090	0.3727
dab monobit2	0.8994	0.0507	0.6055

Confusion and Diffusion Tests

This part exposes the confusion and diffusion properties of CA-PCS and AES. These properties are indispensable for symmetric systems. As detailed before, the confusion represents the concealed relation of the secret key with the ciphertext referring to the avalanche effect. In short, if changing one bit of the secret key produces about 50% of the ciphertext bits, then the confusion test is successful. As well, the diffusion displays the secret relationship of the plaintext with the ciphertext using the avalanche effect test. In other terms, if the alteration of one bit in the plaintext produces a change of 50% in the ciphertext bits, then the diffusion test is satisfying. Figure 3.5 presents the comparison of CA-PCS with AES in terms of the confusion test. From this figure, it is perceptible that the proportion of altered bits in the ciphertext is nearly 50% for both systems. More specifically, the ranges of values for CA-PCS and AES respectively are [40%, 61%] and [36%, 61%]. From these ranges, both systems fulfill the confusion test. However, the values of CA-PCS are considered better than those of AES.

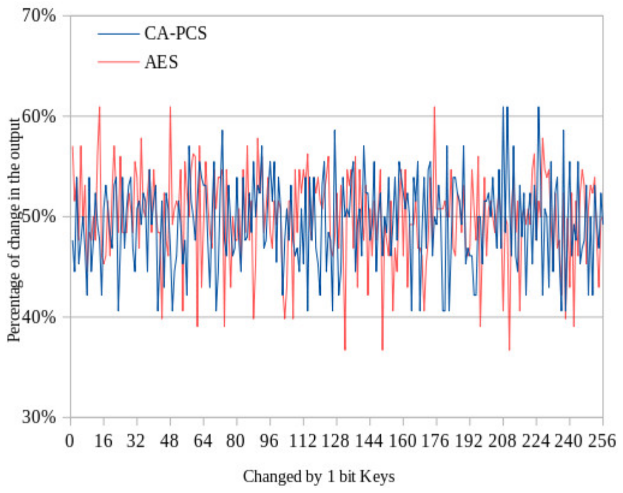


Figure 3.5: Confusion Test of CA-PCS and AES

Figure 3.6 presents the comparison of CA-PCS with AES concerning the diffusion test. Notably, changing one bit in the plaintext provides a modification of almost 50% of the ciphertext bits for both systems. The ranges of values for CA-PCS and AES respectively are [41%,61%] and [37%,67%]. From these values, both systems passe the diffusion test. However, CA-PCS has better values. Consequently, it has better diffusion property compared to AES.

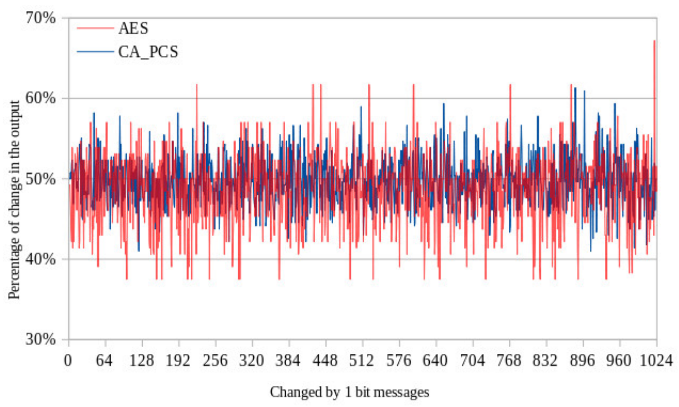


Figure 3.6: Diffusion Test of CA-PCS and AES

Encryption and Decryption Time Of CA-PCS, AES and PCS

This part provides a comparison of CA-PCS with the previous version PCS and AES systems in terms of the time of encryption and decryption. Figure

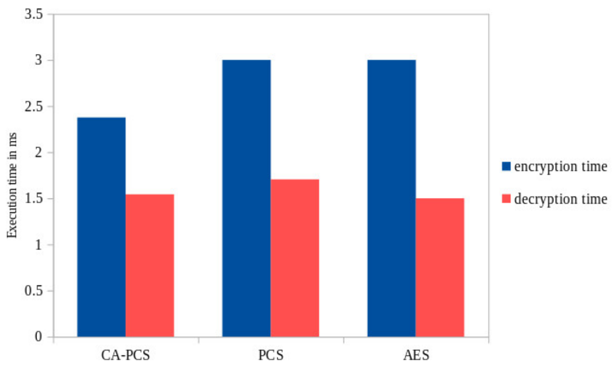


Figure 3.7: Encryption and decryption time of CA-PCS, PCS and AES

3.7 displays that the encryption algorithm of CA-PCS is faster than both PCS and AES encryption algorithms. These latter spend the same amount of time to encrypt. On the other hand, the decryption algorithms of all systems, namely, CA-PCS, PCS, and AES, requires the same amount of time to process. Consequently, CA-PCS is the system providing better results.

Frequency Analysis

This section provides the study of the PCS and the CA-PCS encryption systems concerning the frequency analysis of the outputs. As displayed in the PCS design, the motivation is to provide a ciphertext having an equilibrium

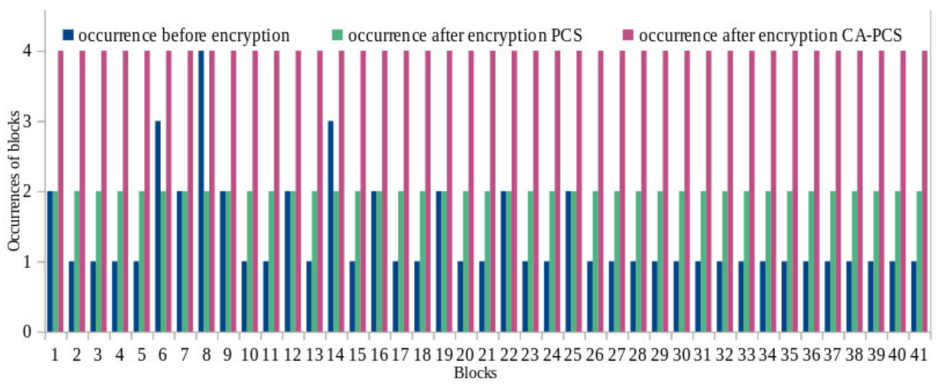


Figure 3.8: Frequency of blocks before and after encryption for CA-PCS and PCS

in the distribution of blocks. In other words, these blocks appear at the same rate, which makes the frequency analysis pointless. As CA-PCS is an extended version of PCS, this fact is still respected. CA-PCS and PCS differ in almost all the steps. CA-PCS starts with the CA evolution step. Next, the calculation of the ideal cardinal. Afterward, comes the blocks' insertion. And finally, the permutation is applied to the output of the previous step. In PCS, the ideal cardinal value produces two possible decisions, whether to insert or remove a block. CA-PCS design aims to offer better confusion and diffusion, as well as robustness to linear and differential attacks. Figure 3.8 presents the block analysis of frequency for the outputs of CA-PCS and PCS regarding the plaintext. From this figure, we noticed that an adversary could not have any information using this attack. Consequently, this latter is unsuccessful.

Cryptographic Properties of The Ruleset Used in the CA evolution

This section displays the cryptographic properties of the ruleset involved in the CA evolution step. Explicitly, the nonlinearity, the algebraic degree, the correlation immunity, the resiliency, and the balancedness. The application of the ruleset {30, 90, 150, 30, 180, 45, 90, 150 } is performed alternatively on the cells of the CA. However, the study of the hole CA space is not practical. Consequently, the study is limited to the ruleset applied for three iterations. Tables 3.2 to 3.6 expose how the cryptographic properties change through multiple clock cycles.

The values of the nonlinearity, as well as the algebraic degree, are consid-

Table 3.2: Nonlinearity

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	0	0	2	2	2	0	0
2	8	8	8	8	8	8	8	8
3	32	48	48	48	28	44	48	48

Table 3.3: Algebraic Degree

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	1	1	2	2	2	1	1
2	3	2	2	3	3	3	2	2
3	4	3	3	4	5	4	3	3

Table 3.4: Resiliency

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	0	0	1	2
2	0	2	2	0	1	0	2	2
3	0	0	0	0	0	0	0	1

Table 3.5: Correlation Immunity

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	0	0	1	2
2	0	2	2	0	1	0	2	2
3	0	0	0	0	0	0	0	1

Table 3.6: Balancedness

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

erably growing with iterations. Moreover, the balancedness remains during the three iterations. Consequently, due to the high nonlinearity and its impact on both correlation immunity and resiliency, their values reduce with iterations. Nonlinearity, algebraic degree, and balancedness are essential properties for different cryptographic systems. These properties strengthen the system and make it robust against linear and differential attacks, which exploit the linearity of the system and try to find a linear approximation. Also, statistical attacks are hard to succeed.

Brute-Force Attack

The adversary, in this type of attack, tries to check out every possible key to find out a valid plaintext via the ciphertext processing [22]. Hence, to ensure a specific level of security, the size of the key is the parameter specifying this latter. If the key is large, then more time and resources are required to find the appropriate key. In other words, except in the case of the availability of quantum computers, this attack is not successful. The key in symmetric systems should include at least 128 bits to provide the lowest security level required. To guarantee long term security if quantum computers are available, the size of the key must be at least 256 bits. As AES comprises three possible variants, the user could decide which security level is desired (AES-128, AES-192, and AES-256). The PCS key size is longer than 256 bit, as seen in the previous chapter. As well, the CA-PCS key size is not less than 256 bits too. Consequently, even in the presence of quantum computers, this attack could not provide a valid plaintext. The security level ensured by AES, PCS, and CA-PCS is presented in table 3.7.

Table 3.7: Brute Force Attack of AES, PCS and CA-PCS

Encryption schemes	AES-128	AES-192	AES-256	PCS	CA-PCS
Key length(b)	128	192	256	≥ 256	≥ 256
# possible keys	2^{128}	2^{192}	2^{256}	$\geq 2^{256}$	$\geq 2^{256}$

Linear and Differential Attacks

The linear attack, a known-plaintext attack, investigates a set of plaintexts and ciphertexts linear approximations [74]. The differential attack is a chosen-plaintext attack that explores the differences of plaintexts with ciphertexts [75]. These attacks should be impractical for a symmetric system, which must display good confusion through the nonlinear building blocks making the system robust against these attacks. Typically, in block ciphers, the S-boxes provide this feature. Also, good CAs could be involved to achieve the same objective. The CA-PCS design comprises a CA evolution step involving a CA ruleset with high nonlinearity. This property makes these attacks hardly achievable.

3.6 Conclusion

This chapter presents an extended version of PCS comprising other features making the study of other attacks possible and practical. It is titled CA-PCS and involves a hybrid cellular automaton and a random permutation to ensure better security compared to PCS. Each building block influences the strength of CA-PCS. For instance, the ruleset used in the CA evolution produces better confusion and statistical parameters, as well as resistance to attacks, namely the linear and differential attacks. According to the results, the randomness and balancedness provide robustness against statistical attacks. CA-PCS prevents linear and differential attacks since the nonlinearity and the algebraic degree are high. Also, brute force attacks are challenging.

Chapter 4

CFA : A Cellular Automata Based Cryptographic PseudoRandom Number Generator

4.1 Introduction

Random numbers are useful in different cryptographic applications such as session keys, initial vectors, seeds, salts, nonces, and others. To generate them, a Truly Random Number Generator (TRNG), which makes use of physical sources of randomness, is required. Nevertheless, not all applications could use them because of cost purposes. Therefore, a cryptographically secure PRNG, which generates sequences that are indistinguishable from random sequences, is an adequate candidate to fix this issue. It is a challenging task to design a cryptographic PRNG with a high-security level where all the cryptographic properties are satisfying. It results from the improvement of cryptanalysis methods and certain conflicting cryptographic properties, namely the nonlinearity and resiliency [19]. Consequently, the designers should find a trade-off between the security level and these properties depending on their needs. This chapter presents a new family of Pseudo-Random Number Generators (PRNGs) based on three building blocks, a hash function, a Cellular Automaton(CA), and a block cipher. The general design includes a seeding and a reseeding mechanism producing the unpredictability of the PRNG initial state. The system is modular and allows the modification or the adaptation of one or more modules according to the application or the user's requirements or the degree of randomness. Each building block in the system affords its characteristics to provide a high-quality of randomness, a high level of security, and adequate cryptographic properties. The choice of the rules used in the CA-based building block follows recommendations, also, the evaluation of their cryptographic properties and the application of statistical tests to conclude if the proposed design provides sequences with high quality of randomness. In this chapter, this design is independently used to generate random sequences. It could also be associated with larger systems such as stream ciphers. The rest of this chapter is organized as follow: In section 4.2, CA-based PRNGs are given. Next, in section 4.3 the general scheme of the generator and a specific implementation are described. The experimental results and an analysis of that specific implementation follows in section 4.4.

4.2 Related Work

The first CA-based PRNG for cryptographic use was proposed by Wolfram in [76], in which he employed a one-dimensional uniform CA evolving with the nonlinear rule 30 and r=1. Later, in 1989, Hortensius et al. [77] designed a PRNG involved in a built-in self-test based on non-uniform CAs using the nonlinear rules 30,45 and the linear ones 90,150. They have also studied the uniform CA rule 30. In 1994, Nandi et al.Nandi1994 studied hybrid CAs and suggested five rulesets from the combination of the linear rules 51, 153, and 195. The ruleset found are R1=153, 153, 153, 153, 51, 51, 51, 51, R2= 195, 195, 195, 195, 51, 51, 51, 51, R3=51,51,153, 153, 153, 153, 51, 51, R4=51, 51, 195, 195, 195, 195, 51, 51, R5=51, 153, 153, 153, 153, 153, 153, 51. Later in 1999, Tomassini et al [79] used an evolutionary method called cellular programming to find the hybrid ruleset 90, 105, 150, 165 depending on the entropy level provided. Afterward, Guan et al. proposed the adoption of controllable CAs. These latter use a sort of signals to control the cell's transitions each clock cycle for more unpredictability [80]. They also made use of self-programmable CAs SPCAs involving a control signal that picks the rules in the evolution process. They evaluated the rules 90, 165 and 150, 105 adopted by SPCAS, and suggested that they provide a high quality of randomness[81]. In[82] Seredynski et al. applied cellular programming to a set of 47 rules to determine the appropriate ones. In the case of r=1, the selected rules are 86, 90, 101, 105, 153, 165. Lately, Bhattacharjee et al.[83] studied a 3-state One-dimensional CA with r=1. The rule found to be good for the random generation is 120021120021021120021021210. All these designs are vulnerable to attacks or had some limitations. This area of study remains fresh and needs more attention for better advance. In this chapter, the design combines a non-uniform one-dimensional CA with other cryptographic primitives to increase the robustness and the randomness degree of its output.

4.3 CFA Generator Design

4.3.1 General Design

The motivation of the general CFA scheme was to establish a versatile PRNG system. In simple words, it could be adjusted depending on the system's needs. As well, a larger design may use it. That is to say, each building block is independent, which means that it can use previously developed cryptographic primitives, as it can involve new ones. Moreover, CFA was designed in a way to be secure against many attacks that could face a PRNG.Moreover, CFA takes into account further conditions, such as flexibility, efficiency, usability, and simplicity. The fundamental components of the CFA general design are as follows:

- A seed file: it comprises sequences with high entropy.
- A reseed trigger tool: it serves to start the mechanism of reseeding.
- A reseed technique: it refreshes the seed file.
- A generator algorithm: it provides random output it using a hash function, a block cipher, and an additional function to produce the output.

Figure 4.1 shows the fundamental components of the general design of CFA.

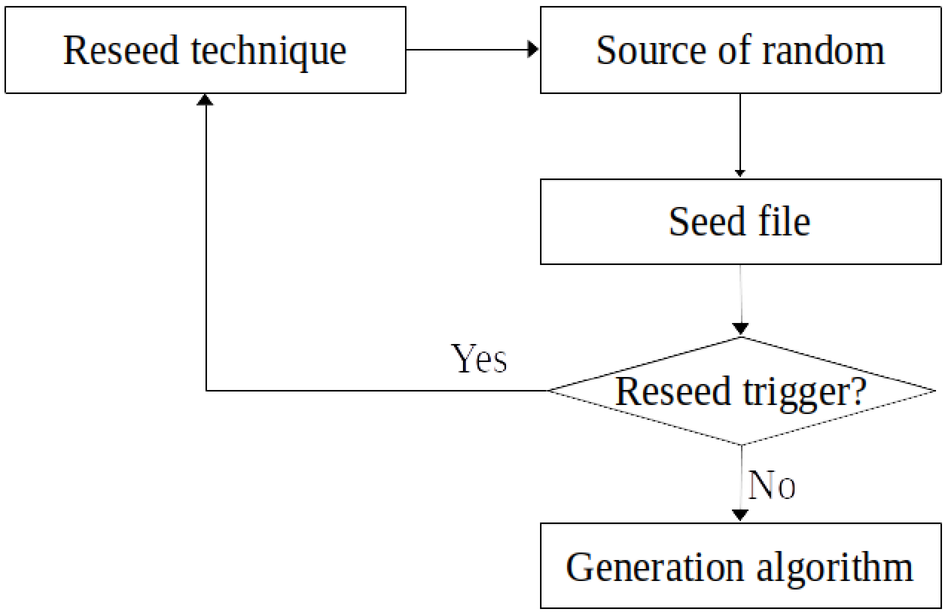


Figure 4.1: CFA General Design

Seed File

The seed file contains high entropy sequences that will be adopted to generate the output of the PRNG. These sequences originate from entropy sources or a PRNG with high-quality output according to accessible sources. Flash memory comprises the seed file containing those sequences, which are supplied to the generation algorithm using the reseed technique if necessary.

Reseed Trigger

The reseed trigger manages when the reseeding operation is indispensable based on a specific parameter, such as the number of generated streams or the PRNG's period. Besides, the reseeding could be actuated by this tool if an adversary made the PRNG in a compromised state.

Reseed Technique

This technique is essential to refresh the seed file. If the reseed trigger evokes the reseeding process, then the seed file is updated by including a newly generated high-entropy random sequence. After the use of all the sequences of the seed file, these later are removed and new ones are provided.

Generator Algorithm

It is the design's fundamental part. This generator makes use of a cellular automata with specific characteristics in addition to two cryptographic primitives to provide a high-entropy random bitstream. The rest of this section describes the general form of the generator and its particular implementation.

4.3.2 Description

The CFA generator requires three fundamental elements:

- A secure hash function h(x): It should be a one-way function and strong second pre-image attacks, as well as Birthday attacks.
- A CA function Evol(x): It evolves the CA iteratively and provides suitable characteristics to the PRNG.
- A block cipher E(x): It should have satisfying statistical properties and security against attacks.

The security level provided by CFA is determined by the min(m,k), such that m is the size of the h(x) output, and k is the key length of E(x). Figure 4.2 outlines the CFA generator involving h, Evol, and E.

A Specific Implementation: CFA-256

The fundamental components of CFA-256 are:

- A seed provided using the Bouncy Castle Java library.
- The hash function SHA-3-256 (Sponge construction)
- A hybrid 3-neighborhood one-dimensional CA.
- The encryption block AES-256-CTR.

A) Seed File Generation
Firstly, the CFA-256 design generates random seeds by the Bouncy Castle Java library's class ThreadedSeedGenerator. The seed file is filled by 1024-bit seeds as long as required. Next, the seed file is accessed to take seed as input to the hash function SHA-3.
B) SHA-3

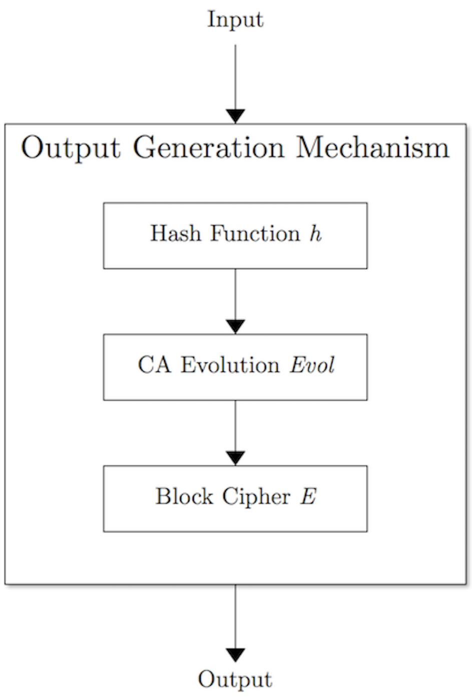


Figure 4.2: CFA generator

Differently from SHA-1 and SHA-2, which are based on the Merkle-Damgard structure, SHA-3 uses a sponge function. This latter comprises three phases, a preprocessing step where the message is split into blocks and padded if needed. Next comes the absorbing phase in which the resulting blocks are manipulated by the transformation function. At last, the squeezing phase provides the digest using the same function f. The transformation function f is a permutation of 24 rounds comprising the θ, ρ, Π, χ, and ι operations. Figure 4.3 illustrates the generation of the SHA-3 digest, where the bitrate

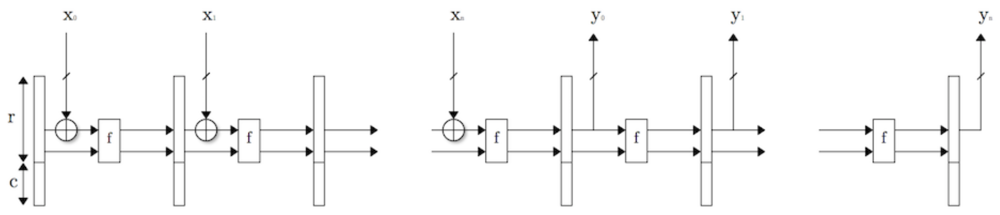


Figure 4.3: SHA-3 Hash Function

r refers to the size of the input blocks, and the capacity c that is dependent on the SHA-3 level of security, such that c=2 x output size. Both parameters sum to the SHA-3 state width b=1600. SHA-3 could produce, from an arbitrary-length message, digests of different sizes (i.e., 224, 256, 384, and 512 bits). In our case, the CFA-256 design includes the SHA-3 version producing a 256-bit digest, where the bitrate r equals 1088, and the capacity value c is 512. Table 4.1 [10] presents the values of the different parameters for SHA-3 depending on the possibles output length.

Table 4.1: SHA-3 Parameters [13]

Digest Size	224	256	384	512
Message Size	No max	No max	No max	No max
Block Size	1152	1088	832	576
Word Size	64	64	64	64
Number of Rounds	24	24	24	24
Capacity c	448	512	768	1024
Collision Resistance	2^{112}	2^{128}	2^{192}	2^{256}
Second Preimage Resistance	2^{224}	2^{256}	2^{384}	2^{512}

This step outputs a digest of 256 bits, which is input to the CA evolution step.

Step 2: CA Evolution

In this step, a non-uniform CA uses the ruleset $R = \{$ 30, 90, 150, 30, 110, 30, 90, 150$\}$ to produce the output. According to the instructions of [19], the combination of linear and non-linear rules forms this ruleset. All the cells evolve through the use of R successively for 128 clock cycles. Figure 4.4 illustrates one iteration of the CA evolution using R.

Then, the output of this step goes through the AES encryption with CTR mode.

Step 3: AES Encryption

At this stage, the CA evolution's output goes through the AES encryption process to produce random bitstream, which is considered the ciphertext depicted in Figure 4.5. In brief, the AES encryption starts with the bitwise XOR of the plaintext with the first round key, then the round function that includes four transformations is applied 13 times. Lastly, it follows the last

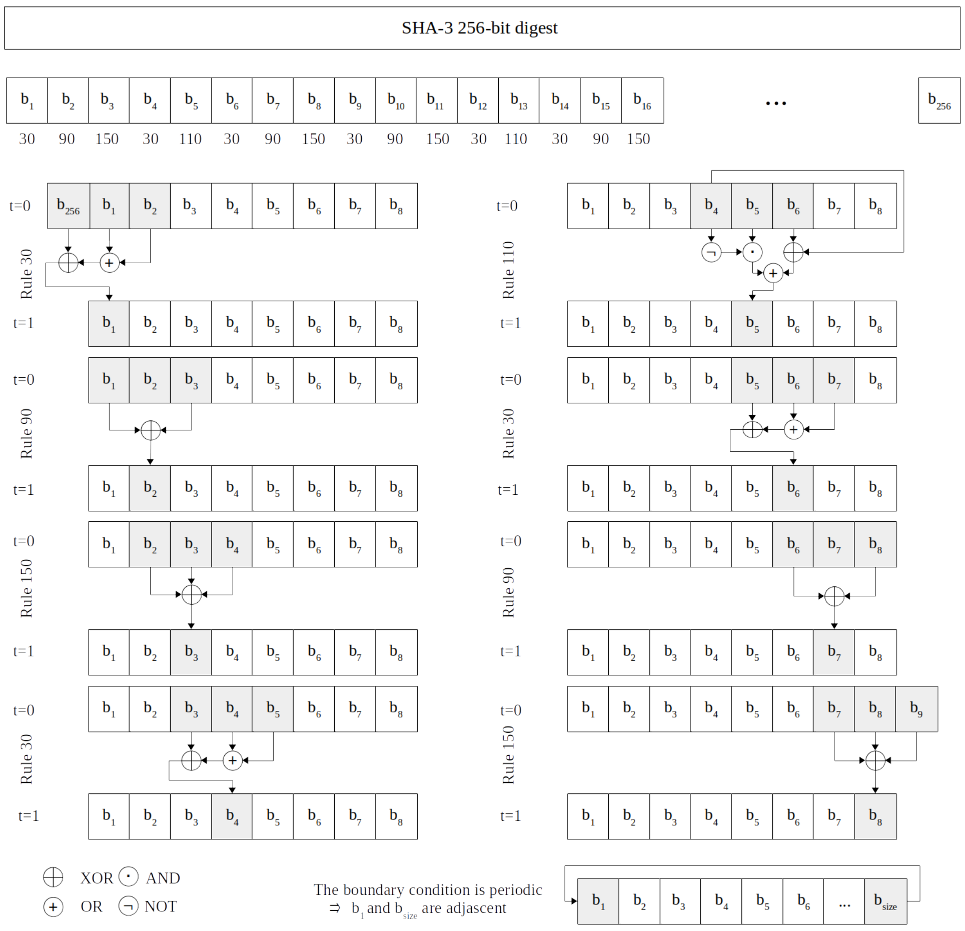

SHA-3 256-bit digest
b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 ... b256
30 90 150 30 110 30 90 150 30 90 150 30 110 30 90 150
t=0
t=1
Rule 30
Rule 90
Rule 150
Rule 110
XOR
AND
OR
NOT
The boundary condition is periodic
⇒ b1 and bsize are adjascent
bsize

Figure 4.4: CA Evolution Step

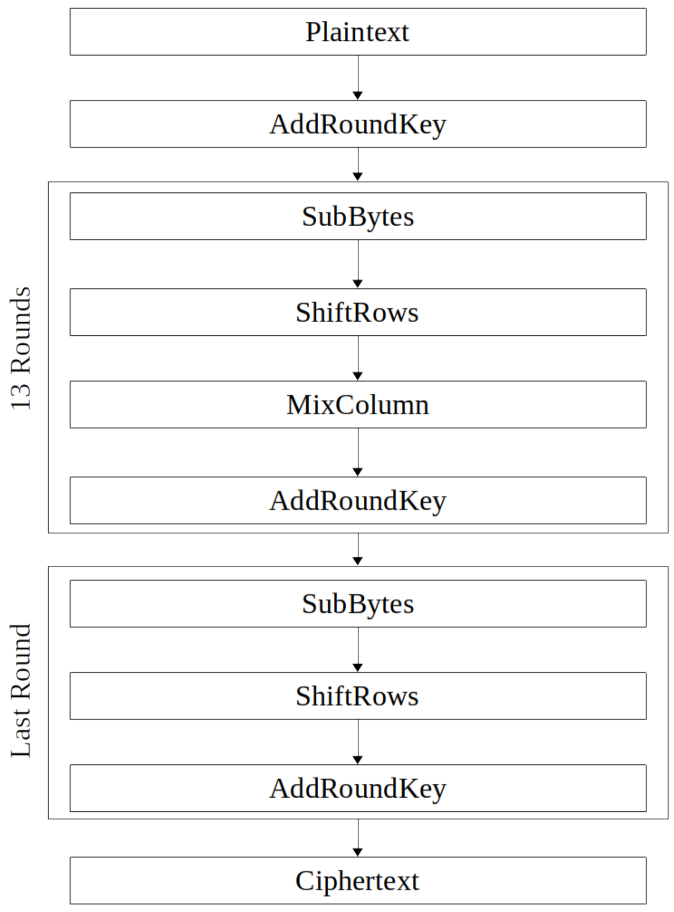


Figure 4.5: AES-256 Encryption [2]

round, which has a minimal difference compared to the previous rounds, to get the ciphertext.

CFA-256 Algorithm

Algorithm 5 presents how bitstreams of high entropy are supplied to the seed file. Algorithm 6 depicts the bitstreams generation through CFA-256.

Algorithm 5 Seed file creation/filling

```
Input: The number of desired sequences nbOfSeq
Output: The seed file seedFile
Begin
if seedFile exists then
    $Override(seedFile)$
else
    $Create(seedFile)$
end if
$n \leftarrow 0$
while $n < nbOfSeq$ do
    $Seq[n] \leftarrow ThreadedSeedGenerator(128)$
    while $H(seq[n]) \leq 0.9$ do
        $Seq[n] \leftarrow ThreadedSeedGenerator(128)$
    end while
    $seed[n] \leftarrow seq[n]$
    $n \leftarrow n + 1$
end while
$seedFile \leftarrow seed$
return $seedFile$
End
```

4.4 Results and Security Analysis

4.4.1 Security

This section presents the security of the proposed PRNG. A cryptographic primitive-based PRNG is as secure as its components[84]. Among the most common reasons producing a vulnerable PRNG, follows:

Entropy overestimation and guessable starting points

This kind of problem is possible in the absence of the seed/reseed process[86]. To prevent this type of attack, CFA-256 uses a seed file and a seeding/reseeding process. The bitstreams stored in the seed file have satisfying entropy are fed to the hash function. Also, the seed/reseeding operation makes the PRNG's state renewable periodically or at request if needed. Consequently, it is not practical to try finding the initial configuration of the PRNG. While in the other PRNGs proposed in [78],[79]and [82], no seed/reseed process is present. Thus, these PRNGs are vulnerable to such situations.

Algorithm 6 Pseudo-code CFA-256

```
Input: The seed $seed$
Output: The Random bits (256 bits) outputCFA
Begin
$j \leftarrow (-sizeOf(seed) - 2)\ mod\ 256$
$ruleSet \leftarrow \{30, 90, 150, 30, 110, 30, 90, 150\}$
$paddedSeed \leftarrow seed10^j1$
$s \leftarrow SHA3(paddedSeed, 256)$
$s[0] \leftarrow s[256]$
$s[257] \leftarrow s[1]$
for $1 \leq it \leq 128$ do
    for $1 \leq i \leq 256$ do
        $k \leftarrow i - 1\ modulo\ sizeOf(ruleSet)$
        if $(k == 0)(k == 3)(k == 5)$ then                    ▷ 30
            $evolution[i] \leftarrow s[i-1] \oplus (s[i] + s[i+1])$
        else if $(k == 1)(k == 6)$ then                       ▷ 90
            $evolution[i] \leftarrow s[i-1] \oplus s[i+1]$
        else if $((k == 2)(k == 7))$ then                     ▷ 150
            $evolution[i] \leftarrow s[i-1] \oplus s[i] \oplus s[i+1]$
        else                                                  ▷ 110
            $evolution[i] \leftarrow s[i-1] \oplus s[i+1] \oplus s[i-1].s[i+1] \oplus s[i-1].s[i].s[i+1]$
        end if
    end for
end for
$outputCFA \leftarrow AES256(evolution)$
End
```

Chosen-Input Attacks

This type of attack exploits the fact that PRNG's inputs are not preprocessed before being fed to its state. To prevent these attacks, CFA-256 includes a secure hash function, namely SHA-3. However, no technique or process prevents or reduces the possibility of chosen-input attacks in [78],[79]and [82].

Side-Channel Attacks

To avoid these types of attacks, CFA-256 involves a non-uniform CA, which includes thoughtfully selected linear and nonlinear rules having satisfying cryptographic properties. According to the tables presented in the cryptographic properties section, CFA-256 displays better characteristics than [78],[79]and [82].

Direct Cryptanalytic attacks

Direct Cryptanalytic Attacks are avoided in CFA-256 by the inclusion of a robust block cipher together with a secure hash function. It is important to note that a state compromise related attack implies that the PRNG state should change. In [78],[79]and [82], there is no method involved to lower the threat of direct cryptanalytic attacks.

4.4.2 Results

Cryptographic Properties of CA

This section presents the cryptographic properties of the CA evolution step ruleset 30,90,150,30,110,30,90,150. Since it is impractical to study these properties for a 256-cells CA, these results were conducted for an 8-cells CA (i.e., the size of the ruleset) during three clock cycles, to explore the cryptographic properties variation from one iteration to the other. The other cells of the 256-cells CA evolve using the same ruleset. So, from the study of the ruleset(8-cells CA), one could conclude the properties of the 256-cells CA. In the tables, the x_is denote the CA cells. The following tables 4.2 to 4.6 illustrate the variation of the values of the algebraic degree, the nonlinearity, the correlation immunity, the resiliency, and the balancedness.

Table 4.2: Nonlinearity

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	0	0	2	1	2	0	0
2	8	8	8	6	10	8	8	8
3	32	48	52	36	40	48	48	48

Table 4.3: Algebraic Degree

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	1	1	2	3	2	1	1
2	3	2	2	4	4	3	2	2
3	4	3	4	5	5	4	3	3

Table 4.4: Resiliency

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	-1	0	1	2
2	0	2	2	0	-1	-1	2	2
3	0	0	0	0	-1	-1	-1	1

Table 4.5: Correlation Immunity

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	0	0	1	2
2	0	2	2	0	0	1	2	2
3	0	0	0	0	0	2	0	1

These tables show the increase of nonlinearity and algebraic degree with iterations. On the other hand, the correlation immunity and resiliency decrease through evolutions. Also, the balancedness remains for the majority of cells from one iteration to another. Accordingly, the rules involved are satisfying regarding cryptographic properties. In comparison with the PRNGs proposed in [78],[79]and [82], the results are presented in tables for the PRNG presented in this chapter is adequate for cryptographic applications.

Table 4.6: Balancedness

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	×	✓	✓	✓
2	✓	✓	✓	✓	×	×	✓	✓
3	✓	✓	✓	✓	×	×	×	✓

Seredynski [82] Ruleset Cryptographic Properties

Ruleset: 86, 90, 101, 105, 153, 165 The following tables present the variation of the cryptographic properties corresponding to the ruleset provided by the CA-based PRNG defined by Seredinsky [82].

Table 4.7: Nonlinearity of [82]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	0	2	0	0	0	2	0
2	8	12	8	8	0	0	8	8
3	32	48	48	32	32	32	32	32

Table 4.8: Algebraic Degree of [82]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	1	2	1	1	1	1	1
2	2	2	3	2	1	1	3	2
3	2	3	4	3	2	3	3	3

Tomassini [79] Ruleset Cryptographic Properties

Ruleset: 90,105,150,165 The following tables present the different cryptographic properties of the ruleset employed by the CA-based PRNG specified by Tomassini [79].

Table 4.9: Correlation Immunity of [82]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	0	2	1	1	0	1
2	1	0	0	0	1	2	1	0
3	1	0	0	1	1	1	0	1

Table 4.10: Resiliency of [82]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	0	2	1	1	0	1
2	1	0	0	0	1	2	1	0
3	1	0	0	1	1	1	0	1

Table 4.11: Balancedness of [82]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

Table 4.12: Nonlinearity of [79]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

Table 4.13: Algebraic Degree of [79]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

Table 4.14: Correlation Immunity of [79]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	2	2	1	1	2	2	1
2	2	3	3	2	2	3	3	2
3	3	4	2	4	4	2	2	3

Table 4.15: Resiliency of [79]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	2	2	1	1	2	2	1
2	2	3	3	2	2	3	3	2
3	3	4	2	4	4	2	2	3

Table 4.16: Balancedness of [79]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

Nandi [78] Rulesets Cryptographic Properties

The following tables display the various cryptographic properties of the first ruleset used in the CA-based PRNG defined by Nandi [78].
Ruleset 1: {153,153,153,153,51,51,51,51}

Table 4.17: Nonlinearity of R1 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

Table 4.18: Algebraic Degree of R1 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

Table 4.19: Correlation Immunity of R1 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	0	0	0	0
2	1	1	1	1	0	0	0	0
3	3	3	3	2	0	0	0	0

Table 4.20: Resiliency of R1 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	0	0	0	0
2	1	1	1	1	0	0	0	0
3	3	3	3	2	0	0	0	0

Table 4.21: Balancedness of R1 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

The following tables illustrate the cryptographic properties of the second ruleset applied in the CA-based PRNG specified by Nandi [78].
Ruleset 2:{195,195,195,195,51,51,51,51}

Table 4.22: Nonlinearity of R2 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

Table 4.23: Algebraic Degree of R2 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

Table 4.24: Correlation Immunity of R2 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	0	0	0	0
2	1	1	1	1	0	0	0	0
3	1	1	3	3	0	0	0	0

Table 4.25: Resiliency of R2 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	0	0	0	0
2	1	1	1	1	0	0	0	0
3	1	1	3	3	0	0	0	0

Table 4.26: Balancedness of R2 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

The following tables depict the different cryptographic properties of the third ruleset used in the CA-based PRNG specified by Nandi [78].
Ruleset 3: 51,51,153,153,153,153,51,51

Table 4.27: Nonlinearity of R3 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

Table 4.28: Algebraic Degree of R3 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

Table 4.29: Correlation Immunity of R3 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	1	1	1	1	0	0
2	0	0	1	1	1	2	0	0
3	0	0	3	3	2	1	0	0

Table 4.30: Resiliency of R3 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	1	1	1	1	0	0
2	0	0	1	1	1	2	0	0
3	0	0	3	3	2	1	0	0

Table 4.31: Balancedness of R3 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

The following tables represent the cryptographic properties of the fourth ruleset utilized in the CA-based PRNG defined by Nandi [78].
Ruleset 4: 51,51,195,195,195,195,51,51

Table 4.32: Nonlinearity of R4 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

Table 4.33: Algebraic Degree of R4 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

Table 4.34: Correlation Immunity of R4 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	1	1	1	1	0	0
2	0	0	0	1	1	1	0	0
3	0	0	1	2	3	3	0	0

Table 4.35: Resiliency of R4 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	1	1	1	1	0	0
2	0	0	0	1	1	1	0	0
3	0	0	1	2	3	3	0	0

Table 4.36: Balancedness of R4 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

The following tables present the cryptographic properties of the fifth ruleset employed by the CA-based PRNG specified by Nandi [78].
Ruleset 5: 51, 153, 153, 153, 153, 153, 153, 51

Table 4.37: Nonlinearity of R5 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

Table 4.38: Algebraic Degree of R5 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

Table 4.39: Correlation Immunity of R5 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	1	1	1	1	1	0
2	0	1	1	1	1	1	2	0
3	0	3	3	3	3	2	1	0

Table 4.40: Resiliency of R5 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	1	1	1	1	1	0
2	0	1	1	1	1	1	2	0
3	0	3	3	3	3	2	1	0

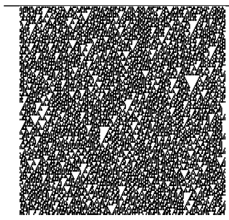
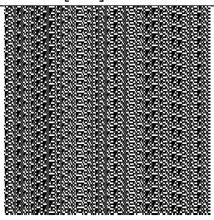
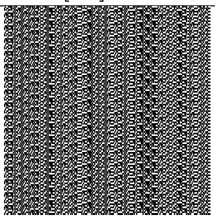
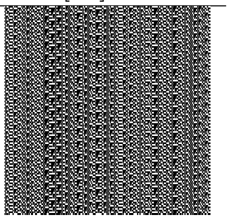
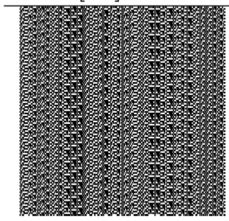
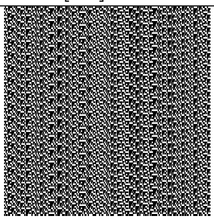
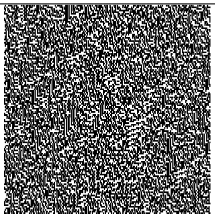
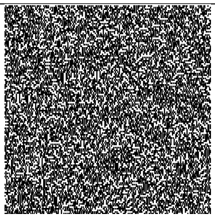
Table 4.41: Balancedness of R5 [78]

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

4.4.3 Cellular Automata Evolutions

The following table compares the CA evolution step of the proposed design (CFA-256) with the other CA-based PRNGs specified by Nandi [78], Tomassini [79], and Seredinsky [82] according to the space-time diagram, which is a graphical display of the CA conduct. It shows if there is a specific template followed by the CA in different situations. The 256-cells CAs corresponding to the PRNGs evolves 128 clock cycles. Table 4.42 displays that the rulesets adopted in [78] follow repeated templates. The rest designs [79], [82], and CFA-256 do not produce a reproduction of any particular pattern and have a uniform distribution of ones and zeros.

Table 4.42: Evolutions of Different Designs

CFA CA	[78] R1	[78] R2	[78] R3
			
[78] R4	[78] R5	[79]	[82]
			

4.4.4 Avalanche Effect

Feistel was the first to define the avalanche effect [87] that is one of the most important properties that should satisfy all cryptographic systems [22]. The rate of variation displayed in the output bits by minor modification of the input bits defines this concept[22]. Its mathematical expression figures in the following formula:

$$\forall I, I' H(I,I') = 1, avalanche(I') = \frac{H(F(I),F(I'))}{size(F(I))} \times 100$$

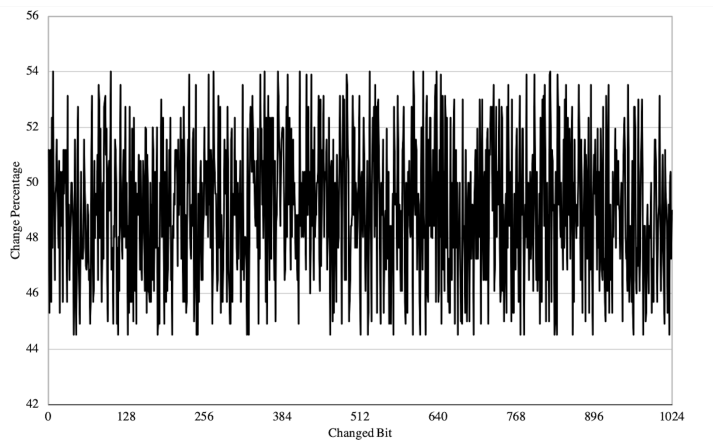


Figure 4.6: Avalanche Effect

where I is the original input, and I' is the input with one bit changed compared to I, and F is to the function that the I and I' undergo such that F is CFA-256 in this chapter.

The avalanche effect test of CFA-256 involves the following steps. At first, one hundred of 1024-bit of high entropy seeds are generated by the threaded seed generator provided by the bouncy castle library. Next, the outputs corresponding to each seed are computed $\{O_1, ..., O_{100}\}$. Also, to each seed, one bit is altered each time to compute its outputs, $\{\{O_1^1, O_1^2, ..., O_1^{1024}\}, ..., \{O_{100}^1, O_{100}^2, ..., O_{100}^{1024}\}\}$. Afterward, the hamming distances $H_{ij} = H(O_j\ , O_j^{(i)})$ $1 \leq i \leq 1024$, $1 \leq j \leq 100$ of the outputs of the changed seeds $\{\{O_1^1, O_1^2, ..., O_1^{1024}\}, ...,\{O_{100}^1, O_{100}^2, ..., O_{100}^{1024}\}\}$ and the original ones $\{O_1, ..., O_{100}\}$. Then for each bit, the average of the hamming distances is computed $AvgH_i = \frac{\sum_{j=1}^{100} H_{ij}}{100}$. Finally, the average avalanche effect is determined by the given formula. Figure 4.6 displays the results. The maximum percentage of the variation in the output is 54.29%, the minimum value is 44.31%, and the average is 49.08%. Accordingly, an alteration of one bit in the input produces a variation of approximately 50% of the output bits. Consequently, CFA-256 has a satisfying diffusion property.

4.4.5 NIST Statistical Test Suite

The National Institute of Standards and Technology designed the NIST Statistical Test Suite that comprises 15 tests to evaluate the PRNGs' behavior and their statistical properties. For each sequence, the test algorithms compute the p-value, which refers to the probability that a TRNG was used to generate it. More specifically, the p-values estimate the distance between the corresponding bitstream and random ones. The results of the tested algorithms are successful if the p-values are in the range $[\alpha, 1-\alpha]$, where α is the significance level, $\alpha = 0.01$. The NIST report [23] more details about this test suite. The CFA-256 algorithm generates a file, provided to the test suite, containing a 10-Mb bitstream of 256-bit sequences to evaluate its randomness and statistical properties. Table 4.43 displays the tests' results. From these results, applicable tests were successful since the p-values range between 0.3946 and 0.7857. Some tests are unapplicable because of parameter constraints, such as the size of the sequences. The size of the CFA-256 output is 256 bits, which is small to apply some tests. For instance, the Random Excursions Test requires a 1 000 000 bit sequence.

Table 4.43: NIST Statistical Test Suite results

Test name	p-value	Pass?
The Frequency (Monobit) Test	0.5884	✓
Frequency Test within a Block	0.4362	✓
The Runs Test	0.5381	✓
Tests for the Longest-Run-of-Ones in a Block	0.5448	✓
The Binary Matrix Rank Test	-	N/A
The Discrete Fourier Transform (Spectral) Test	0.4079	✓
The Non-Overlapping Template Matching Test	-	N/A
The Overlapping Template Matching Test	-	N/A
Maurer's "Universal Statistical" Test	0.3946	✓
The Linear Complexity Test	0.7515	✓
The Serial p-value1 Test	0.7857	✓
The Serial p-value2 Test	0.7030	✓
The Approximate Entropy Test	0.7335	✓
The Cumulative Sums (Cusums) Forward Test	0.6292	✓
The Cumulative Sums (Cusums) Reverse Test	0.7458	✓
The Random Excursions Test	-	N/A
The Random Excursions Variant Test	-	N/A

4.4.6 Dieharder battery of tests

This section presents the dieharder results of CFA-256 and the other CA-based PRNGs presented in [78],[79], and [82]. The significance level in dieharder equals 0.005, so the p-values should stand in the range [0.005, 0.995] to consider that the system passes a test. Table 4.44 gives the results, where the bold values represent the failed tests. For CFA-256, all the tests were successful, while the other systems fail several tests. Consequently, CFA-256 is a better candidate to generate random sequences.

Table 4.44: DIEHARDER Test Suite Results

Test	CFA	[78]R1	[78]R2	[78]R3	[78]R4	[78]R5	[79]	[82]
1	0.6623	0.9802	0.8421	0.4237	0.3459	0.9768	0.4378	0.2232
2	0.5403	0.4148	0.4755	0.2383	0.5784	0.0202	0.2292	0.6905
3	0.6715	0.8507	0.7957	0.9124	0.4491	0.1362	0.2419	0.3264
4	0.9007	0.4125	**0.0002**	0.4293	0.9708	0.5195	0.8282	0.1726
5	0.2464	0.9366	0.3630	0.6942	0.1792	0.5380	0.8924	0.6778
6	0.8277	0.0698	0.5419	0.3899	0.6108	0.1515	0.9086	**0.0001**
7	0.0854	0.8900	0.4643	0.6756	0.2325	0.1522	0.8017	0.5926
8	0.4641	0.7005	0.1575	0.0262	0.2483	0.1222	0.5061	0.5227
9	0.4858	0.3388	0.0116	0.7640	0.6682	0.0060	0.2537	0.9393
10	0.4470	**0.9960**	0.9675	0.6349	0.1520	0.3032	0.6834	0.1153
11	0.2059	0.7304	0.6101	0.9838	0.3066	0.9427	0.9505	0.2887
12	0.6967	0.7601	0.4023	0.3048	0.3965	0.1029	0.9210	0.3030
13	0.3451	0.8119	0.9807	0.1776	0.8243	0.5984	0.8955	0.7754
14	0.5753	0.8465	0.6718	0.7075	0.7877	0.3422	0.8840	0.0688
15	0.5935	0.0481	0.5318	0.5789	0.2052	0.4397	0.0267	0.0308
16	0.2164	0.9461	0.9868	0.4022	0.1908	0.0285	0.6756	0.2968
17	0.3887	0.7592	0.3333	0.9596	0.6374	0.0793	0.7020	0.8279
18	0.4494	0.4690	**0.9998**	**0.9968**	0.7307	0.9139	**0.9982**	0.4598
19	0.8838	0.9109	0.9752	0.1365	0.2587	0.7308	0.9376	0.4891
20	0.5977	0.1979	0.5449	0.5154	0.6203	**0.9966**	0.5226	0.0957
21	0.4952	0.1462	**0.9974**	**0.9986**	0.2273	0.1311	0.9534	0.6477
22	0.4434	0.3337	0.7738	**0.9957**	0.7480	0.6602	0.7407	0.2266
23	0.4228	0.0475	0.8172	0.1231	0.9650	0.6983	0.1185	0.0505
24	0.6120	0.4065	0.3640	0.1977	0.0096	0.5224	**0.9992**	**0.9983**
25	0.5594	0.8959	**0.9992**	**0.9960**	0.9813	0.2458	0.9397	**0.9961**
26	0.6816	0.3811	0.3171	0.6940	0.0900	0.3590	0.5957	0.6173
27	0.4042	0.2968	0.3197	0.6481	0.9695	0.6180	0.6694	0.3752
28	0.9476	0.7612	0.1361	0.8437	0.9904	06670	0.8914	0.0999
29	0.7678	0.2497	0.1738	0.1829	0.2446	0.7298	**0.9967**	0.0461
30	0.7613	0.9869	0.1332	0.0001	0.1170	0.3688	0.9254	0.193
31	0.2594	0.6661	0.8481	0.8585	0.4841	0.5552	0.8214	0.7918

4.5 Conclusion

Pseudorandom number generators are applicable in various fields. For cryptographic use, a PRNG should be cryptographically secure. This chapter provides a PRNGs family following a general design involving three fundamental components, a secure hash function, a CA-based building block, and a strong block cipher. The introduction of these components provides high security to the PRNG. The specific design presented here, titled CFA-256, includes SHA-3, a CA evolving using the ruleset 30,90,150,30,110,30,90,150, and AES-256. Both SHA-3 and AES-256 are considered secure and give their properties to CFA-256. As well, the hybrid CA improves the randomness, confusion, diffusion properties, and security of CFA against known attacks. According to the results of the statistical tests displayed, the sequences generated using CFA-256 are statistically independent and appear to be truly random. Moreover, the avalanche effect and the cryptographic properties supplied by the CA are satisfying and make CFA-256 a good candidate to generate pseudorandom sequences.

Chapter 5

NCASC : 3CASC VS 4CASC Cellular Automata based Stream Ciphers

5.1 Introduction

Stream ciphers are a class of symmetric cryptographic primitives that makes use of pseudorandom number generators to produce the keystream, which is XORed with the plaintext to get the ciphertext. Accordingly, the stream ciphers' security relies on the PRNG robustness. Most PRNGs designs include linear and nonlinear feedback shift registers. However, those designs are vulnerable to fault and correlation attacks, and many others, even if they provide a high quality of randomness. Cellular automata are good alternatives to replace feedback shift registers. Due to their characteristics providing fast encryption, better cryptographic properties, better randomness quality, and higher security level, CAs, and particularly 3-neighborhood CAs, are parts of several cryptographic systems. However, 3-neighborhood CAs are still vulnerable to some attacks. From this perspective, this chapter provides two CA-based stream ciphers, namely, 3CASC and 4CASC, to study their behavior and properties, as well as their security. Exploring all the rules of 4-neighborhood CAs is challenging. For this purpose, the 4-neighborhood rules are determined using 3-neighborhood ones, established referring to the recommendations of [19], by including a fourth variable. Appendix X details this process. The general design of the proposed system comprises three CAs: a linear CA, a nonlinear CA that uses only nonlinear rules, and a mixing CA involving both linear and nonlinear ones. The principal purpose of this chapter's contribution is to evaluate how the neighborhood size impacts the statistical characteristics, the cryptographic properties, as well as the security level. The rest of this chapter is structured as follows: Section 5.2 presents related works. Next, section 5.3 provides 3CASC and 4CASC design. Section 5.4 presents the security analysis, followed by the results in section 5.5.

5.2 Related Work

Grain [20] and Trivium ciphers [88] are from the finalists of the eSTREAM project candidates. Both of them are efficient, simple, and secure designs. Grain combines an LFSR and an NFSR, while Trivium makes use of a simple combination of LFSRs. Nevertheless, by the end of the eSTREAM project, those systems were vulnerable to several attacks, such as correlation attacks that make use of the relation of the output with the IV fault attacks that investigate the propagation of an inserted fault, and other attacks. Later, cellular automata were used instead of LFSRs and NFSRs to avoid these types of attacks. Their cryptographic properties and randomness characteristics make them a perfect alternative to earlier designs. Wolfram designed the first Cellular Automata-based stream cipher[64] [1] that makes use of rule 30 to generate the keystream. Later, Meier and Stafflebach applied MS-attack to this design to show its vulnerability. Afterward, most of the proposed CA-based stream ciphers involves CAs with the following characteristics (d=1, r=1, 2-state CA). For instance, NOCAS[89], CASTREAM[90], CAvium[15], CAR30[91], and CASca[[92] are designed based on Grain[89][91][92] and Trivium[90] [15]ciphers. In other words, the same design is kept such that the linear CAs and nonlinear CAs replace LFSRs and NFSRs, respectively. While a rotational bent function or NMIX[93] is used instead of the filter function. More constructions with different characteristics, such as r=3/2 or r=2, are proposed in [94],[95], and [96]. The neighborhood extension to a higher level provided better cryptographic properties and higher quality of randomness, as well as security against several attacks like algebraic, fault, and correlation attacks. Because of the complexity and the hardness of investigating multidimensional CA-based designs, fewer constructions are proposed(e.g., [97]).

5.3 General Design

This section outlines the N-CASC system, where N refers to the neighborhood that is either 3 or 4. N-CASC's general design takes inspiration from Grain and FResCA, with the purposes of the study of the impact of expanding the neighborhood on the cryptographic properties and security of the proposed system.

5.3.1 General Scheme

The design of the N-neighborhood Cellular Automata-based Stream Cipher (N-CASC) shares the same structure with Grain. However, the N-CASC fundamental components include CAs instead of the feedback registers and the combining function.

Encryption Scheme

The encryption system comprises two steps, namely the initialization and the encryption steps. Each one includes three fundamental components, which are the linear hybrid CA, the nonlinear non-uniform CA, and a hybrid CA combining both linear and nonlinear rules forming the mixing function. The rest of this section outlines these steps and their building blocks.

Initialization Phase This step is required to make the initial state of the system good enough before performing the encryption step. It runs a specific number of clock cycles all the CAs constituting this step to achieve a satisfying confusion as well as the cryptographic properties. This step considers at t_0 a first configurations$C_0 = (KEY, IV)$ of 256 bits, where the KEY and the IV comprise 128 bits. The bouncy castle java cryptographic library includes the ThreadedSeedGenerator class dedicated to the generation of random seeds for cryptographic use. Accordingly, this class produces the KEY and IV for this system. Afterward, the KEY and IV supplied to the nonlinear non-uniform CA and the linear non-uniform CA, respectively, are the CAs' initial state. Then, the CAs evolve, according to the encryption step defined below, n clock cycles as C_n, which is the initial state of the encryption step, is not attained yet. The number of rounds n is determined through the avalanche effect investigation. For this purpose, one hundred pairs of (KEY, IV) are generated, then the initialization step was applied for different values of n, explicitly 4,8,16,32,64,128. Then the computation of the avalanche effect average of all the pairs $C_0^1, ..., C_0^{100} = \{(KEY_1, IV_1), ...(KEY_{100}, IV_{100})\}$, and $C_n^1, ..., C_n^{100}$ is performed. The investigation results are presented in table 3, which displays that the better value should be $n = 64$. This study was made for the 3N version only since the comparison should consider similar parameters.

Table 5.1: Number of Rounds During Initialization Phase

Number of Rounds	**Average Avalanche Effect**
4	48.80859
8	48.84375
16	48.89062
32	48.94141
64	49.01562
128	48.42187

Encryption Phase The components included in this step are similar to those involved in the initialization step. A linear non-uniform CA that evolves according to a ruleset providing to the system a higher cycle length to avoid running the initialization phase frequently. A nonlinear non-uniform CA that processes by the use of only nonlinear rules having adequate cryptographic properties to increase the system's security level. Finally, a non-uniform CA that serves as a mixing function, it proceeds using a ruleset of both linear and nonlinear rules producing better confusion to the system.

Figures 5.1 and 5.2 display, respectively, the initialization and the encryption steps.

As illustrated in Figure 5.2, the ciphertext is the XOR of the keystream z produced by the evolution of the three CAs involved by the plaintext. The CAs states update for more keystreams proceeds as follows: the 32 leftmost and rightmost bits of z are XORed, respectively, to the 32 leftmost bits of the nonlinear CA and the 32 rightmost bits of the linear CA.

Fundamental components This subsection describes the fundamental components involved in the encryption/initialization step, as well, the difference between both versions 3N and 4N, the rules selection, and the CAs

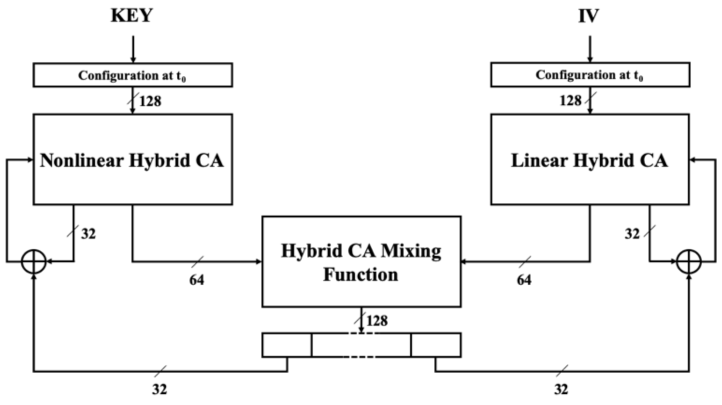


Figure 5.1: Initialization Step

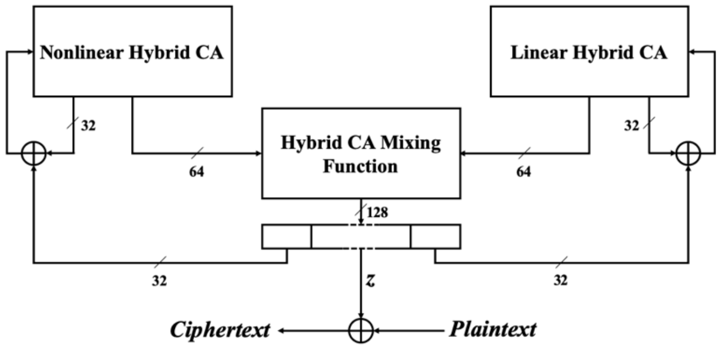


Figure 5.2: Encryption Step

properties variation and their impact on the security of the global system. The rules selected for 3CASC found according to the guidance provided by [19] and [98] . On the other hand, the rules used in 4CASC CAs got from the ones selected for the 3CASC. More precisely, each of these latter used in 3CASC is left-skewed to find similar rules for 4CASC, which go through an evaluation to choose the right one according to the cryptographic properties, following this order: the nonlinearity, algebraic degree, balancedness, correlation immunity, and resiliency, in addition to the space-time diagram.

Nonlinear Hybrid CA 3N Version Case

The nonlinear non-uniform CA is a 128-cell two-state one-dimensional cellular automaton. It evolves n/2=64 clock cycle using the nonlinear rules {30, 120, 180, 45, 30, 120, 180, 45}. After 64 iterations, all the cells depend on all the 128 cells, which means that changing any bit influences all the other cells' results and consequently high confusion property. At t_0, the cellular automaton is updated before encryption by the C_n's 64 leftmost bits produced through the initialization step.

4N Version Case

The 4CASC nonlinear Cellular automaton evolves using the ruleset { 43350, 38490, 25500, 22185, 43350, 38490, 25500, 22185 }. In the case of 4-Neighborhood CAs, only $\frac{n}{3} = 43$ iterations are necessary to reach high confusion property, since after 43 evolutions all the 128-cells depend on all the cells. However, for comparison purposes, $\frac{n}{2}$ evolutions are done.

This CA is entirely nonlinear in the purpose of enhancing the system's security. The nonlinear rules involved in this CA have high nonlinearity, high algebraic degree, as well as balancedness. These cryptographic properties guarantee NCASC strength against some attacks, particularly, algebraic and fault attacks[99].

Linear Hybrid CA **3N Version Case**

The 3CASC linear CA is a 128-cells CA that evolves using 90 and 150, providing high cycle length, $\frac{n}{2}$ times. Same as the nonlinear CA, this CA initially contains the 64 rightmost bits of the output of the initialization step concatenated with the rest of its cells.

4N Version Case

The 4CASC linear CA is a 128-cell CA that process using the ruleset {24330, 27030} for $\frac{n}{2}$ clock cycle.

The rules 90, 150 used in 3CASC provide maximal periodicity according to [16]. Consequently, this CA ensures a long period.

Hybrid CA Mixing Function **3N Version Case**

The 3CASC mixing CA is a 128-cell CA involving both linear and nonlinear rules {30, 60, 90, 120, 150, 180, 240, 15, 45} in the evolution process for n/2 clock cycle. Initially, it takes half of its cells from the linear CA (64 leftmost cells) and the other half from the nonlinear CA(64 rightmost cells)..

4N Version Case

The 4CASC mixing CA is a 128-cells CA that proceeds using the rules {43350, 49980, 42330, 38490, 27030, 25500, 65280, 255, 22185} n/2 clock cycles. Even though only n/3 iterations are sufficient for better results compared with the 3-neighborhood version.

This CA uses both linear and nonlinear rules taking as input the outputs of linear and nonlinear CAs. The cryptographic properties and the high periodicity provided by this CA increase the system security.

Algorithm 7 Pseudo-code 3CASC

```
Input: Key, IV, NbOfKestreams
Output: keystream
Begin
$LinearCARuleset \leftarrow \{90, 150\}$
$NonlinearCARuleset \leftarrow \{30, 120, 180, 45, 30, 120, 180, 45\}$
$MixingCARuleset \leftarrow \{30, 60, 90, 120, 150, 180, 240, 15, 45\}$
$NbOfRounds \leftarrow 64$
$NLCA \leftarrow Key$
$LCA \leftarrow IV$
$keystream \leftarrow$ ""
for $1 \leq r \leq NbOfRounds$ do                                  ▷ Initialization Phase
    $NLCA \leftarrow Evolution(NLCA, NonlinearCARuleset, 64)$        ▷ Nonlinear CA
    $LCA \leftarrow Evolution(LCA, LinearCARuleset, 64)$             ▷ Linear CA
    $MixCA \leftarrow MSB(NLCA, 64)LSB(LCA, 64)$
    $MixCA \leftarrow Evolution(MixCA, MixingCARuleset, 64)$         ▷ Mixing CA
end for
for $1 \leq r \leq NbOfRounds$ do                                  ▷ Keystream generation Phase
    $NLCA \leftarrow Evolution(NLCA, NonlinearCARuleset, 64)$        ▷ Nonlinear CA
    $LCA \leftarrow Evolution(LCA, LinearCARuleset, 64)$             ▷ Linear CA
    $MixCA \leftarrow MSB(NLCA, 64)LSB(LCA, 64)$
    $MixCA \leftarrow Evolution(MixCA, MixingCARuleset, 64)$         ▷ Mixing CA
    $keystream \leftarrow keystreamMixCA$
end for
return $keystream$
End
```

Decryption Scheme

As the encryption and the decryption use similar transformations. Given the key and the IV supplied to the initialization and the keystream generation mechanism to reproduce the keystream. The decryption applies the XOR operation to the ciphertext with the keystream to find the plaintext.

Algorithm 8 Pseudo-code 4CASC

```
Input: Key, IV, NbOfKestreams
Output: keystream
Begin
$LinearCARuleset \leftarrow \{42330, 27030\}$
$NonlinearCARuleset \leftarrow$ { 43350, 38490, 25500, 22185, 43350, 38490, 25500, 22185 }
$MixingCARuleset \leftarrow$ { 43350, 49980, 42330, 38490, 27030, 25500, 65280, 255, 22185 }
$NbOfRounds \leftarrow 64$
$NLCA \leftarrow Key$
$LCA \leftarrow IV$
$keystream \leftarrow$ ""
for $1 \leq r \leq NbOfRounds$ do                    ▷ Initialization Phase
    $NLCA \leftarrow Evolution(NLCA, NonlinearCARuleset, 64)$    ▷ Nonlinear CA
    $LCA \leftarrow Evolution(LCA, LinearCARuleset, 64)$    ▷ Linear CA
    $MixCA \leftarrow MSB(NLCA, 64)LSB(LCA, 64)$
    $MixCA \leftarrow Evolution(MixCA, MixingCARuleset, 64)$    ▷ Mixing CA
end for
for $1 \leq r \leq NbOfRounds$ do                    ▷ Keystream generation Phase
    $NLCA \leftarrow Evolution(NLCA, NonlinearCARuleset, 64)$    ▷ Nonlinear CA
    $LCA \leftarrow Evolution(LCA, LinearCARuleset, 64)$    ▷ Linear CA
    $MixCA \leftarrow MSB(NLCA, 64)LSB(LCA, 64)$
    $MixCA \leftarrow Evolution(MixCA, MixingCARuleset, 64)$    ▷ Mixing CA
    $keystream \leftarrow keystreamMixCA$
end for
return $keystream$
End
```

5.4 Security Analysis

This section presents the security analysis of the proposed design against known attacks and the purpose of each component to avoid these attacks.

5.4.1 Side Channel Attacks

This type of attacks aims to exploit the hardware implementation of a keystream generator to find its internal state when it comes to stream ciphers. It could be achieved via the design's physical properties analysis throughout the keystream generation. These properties could be thermal dissipation, energy consumption, and many other characteristics. The resistance to these attacks relies on the difficulty of reversing the generation mechanism [100]. 3CASC and 4CASC designs include linear CA and two nonlinear CAs, which makes the complexity of these attacks costly.

5.4.2 Time/Memory/Data Tradeoff Attack

This attack aims to reduce the brute force search attack's complexity using a pre-computed key/keystream pairs lookup table defined in the offline step. Then, the adversary monitors the keystream generator outputs to find the corresponding key. This attack's complexity is $O(2^{n/2})$, where n refers to the size of KEY || IV [100]. In the case of 3CASC and 4CASC n=256, consequently, this attack is hard to succeed.

5.4.3 Algebraic Attacks

The purpose of this family of attacks is to find algebraic equations representing the system's behavior. It could start by establishing a set of algebraic equations defining the relation of the initial configuration IC with the output. Afterward, the equations system representing the whole design could be determined using keystream bits captured. Solving such a system leads to find the Key and IV, namely the initial configuration. To avoid these attacks, the nonlinearity and the algebraic degree of the keystream generator process should be at their highest level [100]. According to tables 5.5 and 5.10, the growth of the algebraic degree and nonlinearity of 3CASC and 4CASC through iterations are satisfying. The number of evolutions used follows the recommendations related to this concern according to the size of the neighborhood. Consequently, 3CASC and 4CASC are secure to these attacks.

5.4.4 Linear Approximation Attacks

These are known-plaintext attacks that seek to get a linear approximation of the whole processing of symmetric primitives using some bits of the plaintext and the ciphertext relatively with the bits of the key[101]. Avoiding this class of attacks is based on the high nonlinearity and algebraic degree. In the case of 3CASC and 4CASC, the nonlinear CA and the mixing CA are the elements providing resistance to these attacks. According to tables 5.4, 5.5, 5.9 and 5.10, the increase of the nonlinearity and algebraic degree through iterations is significant. As a result, these attacks become hard to achieve.

5.4.5 Correlation Attacks

In the correlation attacks, the opponent tries to retrieve the keystream generator's internal state starting configuration using several bits of the keystream [102]. Avoiding such attacks is based on the balancedness, together with the high nonlinearity, resiliency, and correlation immunity. Referring to the tables presenting the cryptographic properties of 3CASC and 4CASC, the design displays satisfying strength to these attacks.

5.4.6 Fault Attacks

Fault attacks aim to find out the keystream generator state or the secret key by solving an equation system established using pairs of ciphertext with fault and without fault. In LFSRs and NFSR-based stream ciphers, the attacker could choose positions to include fault so that the resulting equation system is simple to resolve. However, in CA-based stream ciphers, the equation system found is hard to solve due to the CAs diffusion property[95]. The number of the CA evolutions used in 3CASC and 4CASC provides the spread of fault to all the cells. Consequently, it makes these attacks hard to achieve.

5.5 Results

This section presents the results of the statistical tests, the avalanche effect test, and the cryptographic properties to evaluate and compare the security and the quality of randomness provided by 3CASC and 4CASC.

5.5.1 Dieharder Battery of Tests

This part provides the dieharder test battery results. Each test is associated with a p-value reflecting the evaluated bitstream randomness quality. As stated before, a system passes a test if the corresponding p-value is between α and $1 - \alpha$, such that α referring to the significance level is defined to be 0.005. Table 5.2 illustrates the dieharder results of 3CASC and 4CASC.

As Table 5.2 indicates, both 3CASC and 4CASC pass each test in the dieharder battery. Consequently, both have satisfying statistical characteristics and produce keystreams with a high randomness quality providing the indistinguishability property.

5.5.2 NIST Statistical Test Suite

This section presents the statistical characteristics and the randomness quality of the keystreams generated using 3CASC and 4CASC through the NIST Statistical Test Suite. Similarly to the dieharder tests, the p-values determine if the system succeeds in the corresponding test. Here the significance level α is set to be 0.001. Two files of 10 000 000 bit sequences are input to the statistical test suite to test the keystreams generated by 3CASC and 4CASC, respectively. Table 5.3 displays their results.

There is no doubt, according to Table 5.3, that 3CASC and 4CASC succeed all the tests, which guarantee their quality of randomness and statistical characteristics.

5.5.3 Avalanche Effect Test

This section presents the avalanche effect results for both 3CASC and 4CASC. For each version, the ThreadedSeedGenerator class, provided by the bouncy castle library, generates one hundred of 256-bit initial configurations $IC_1 = \{KEY_1, IV_1\}$, ...,$IC_{100} = \{KEY_{100}, IV_{100}\}$. Then the keystreams $z_1 = NCASC(IC_1)$, $z_2 = NCASC(IC_2)$, ...,$z_{100} = NCASC(IC_{100})$ are generated. Then, for each IC_i, the 1-bit changedIC_i^js, $1 \leq j \leq 256$, and their cor-

Table 5.2: Dieharder Battery of Tests

Test	3CASC		4CASC	
	p-value	Pass?	p-value	Pass?
Diehard birthdays	0.8424	✓	0.5206	✓
Diehard OPERM5	0.5199	✓	0.9833	✓
Diehard 32x32 Binary Rank	0.9003	✓	0.5421	✓
Diehard 6x8 Binary Rank	0.0487	✓	0.6334	✓
Diehard_bitstream	0.4212	✓	0.6958	✓
Diehard OPSO	0.0051	✓	0.4837	✓
Diehard OQSO	0.8170	✓	0.2401	✓
Diehard DNA	0.1679	✓	0.3812	✓
Diehard Count the 1s (stream)	0.3554	✓	0.1257	✓
Diehard Count the 1s (byte)	0.5995	✓	0.5644	✓
Diehard Parking Lot	0.1729	✓	0.5762	✓
Diehard Minimum Distance (2d Circle)	0.9929	✓	0.4650	✓
Diehard 3d Sphere (Minimum Distance)	0.4359	✓	0.8811	✓
Diehard Squeeze	0.1712	✓	0.5164	✓
Diehard Sums	0.1439	✓	0.5196	✓
Diehard Runs	0.8278	✓	0.4736	✓
Diehard Craps	0.7255	✓	0.8218	✓
Marsaglia and Tsang GCD	0.7844	✓	0.8227	✓
STS Monobit	0.1308	✓	0.1903	✓
STS Runs	0.0263	✓	0.5934	✓
STS Serial Test (Generalized)	0.4169	✓	0.5562	✓
RGB Bit Distribution	0.5877	✓	0.5663	✓
RGB Generalized Minimum Distance	0.4043	✓	0.6391	✓
RGB Permutations	0.3396	✓	0.4819	✓
RGB Lagged Sum	0.5583	✓	0.5224	✓
RGB Kolmogorov-Smirnov	0.1747	✓	0.5765	✓
DAB Byte Distribution	0.4969	✓	0.9117	✓
DAB DCT (Frequency Analysis)	0.6231	✓	0.5832	✓
DAB Fill Tree	0.4335	✓	0.7668	✓
DAB Fill Tree 2	0.5519	✓	0.4413	✓
DAB Monobit 2	0.5114	✓	0.9481	✓

Table 5.3: NIST STS

Test Name	3CASC		4CASC	
	p-value	Pass?	p-value	Pass?
The Frequency (Monobit) Test	0.5836	✓	0.5086	✓
Frequency Test within a Block	0.3504	✓	0.4573	✓
The Runs Test	0.49111	✓	0.49209	✓
Tests for the Longest-Run-of-Ones in a Block	0.5741	✓	0.4788	✓
The Binary Matrix Rank Test	0.53414	✓	0.73991	✓
The Discrete Fourier Transform (Spectral) Test	0.2648	✓	0.4877	✓
The Non-Overlapping Template Matching Test	0.5425	✓	0.5342	✓
The Overlapping Template Matching Test	0.2872	✓	0.4491	✓
Maurer's "Universal Statistical" Test	0.3238	✓	0.3238	✓
The Linear Complexity Test	0.5384	✓	0.5341	✓
The Serial p-value1 Test	0.3417	✓	0.4989	✓
The Serial p-value2 Test	0.9114	✓	0.5018	✓
The Approximate Entropy Test	0.6691	✓	0.5747	✓
The Cumulative Sums (Cusums) Forward Test	0.7026	✓	0.6547	✓
The Cumulative Sums (Cusums) Reverse Test	0.3649	✓	0.5086	✓
The Random Excursions Test	0.5224	✓	0.4254	✓
The Random Excursions Variant Test	0.5836	✓	0.4489	✓

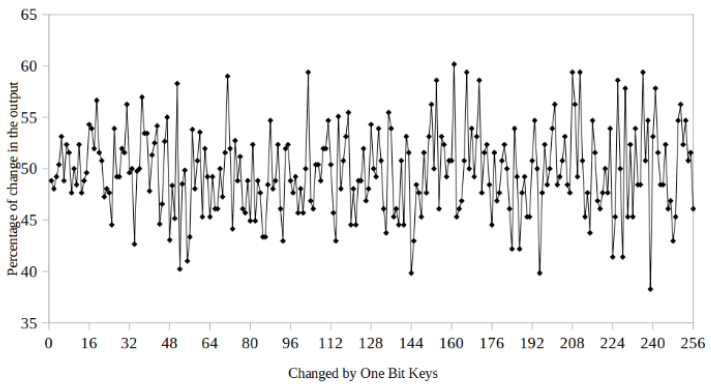


Figure 5.3: Avalanche Effect Test Results for CASC 3N

responding keystreams $z_i^{(j)} = NCASC(IC_i^j)$ are generated $\{\{z_1^1, ..., z_1^{256}\}, ..., \{z_{100}^1, ..., z_{100}^{256}\}\}$. Next, the hamming distances between the keystreams $\{\{z_1^1, ..., z_1^{256}\}, ..., \{z_{100}^1, ..., z_{100}^{256}\}\}$ and $\{z_1, ..., z_{100}\}$ are computed. Then, the average of the hamming distances is computed $AvgHi = \frac{\sum_{j=1}^{100} H_{ij}}{100}$ for each $1 \leq i \leq 256$. At last, the average avalanche effect is determined by the following formula $AverageAvalanche(IC_i, IC_i^j) = AvgH_i/128 * 100$. Such that $1 \leq i \leq 100$ and $1 \leq j \leq 256$. Figure 5.3 and 5.4 displays the average avalanche effect for all the altered bits of 3CASC and 4CASC, respectively.

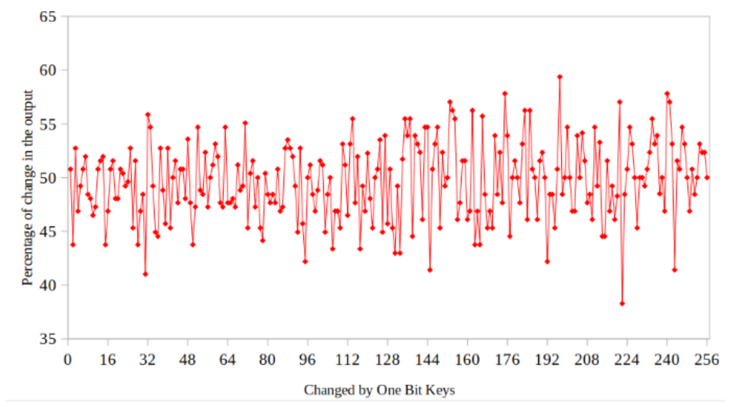


Figure 5.4: Avalanche Effect Test Results for CASC 4N

From these figures, the values of the average of the avalanche effect are approximately 50% for both 3CASC and 4CASC. In particular, 4CASC displays better results compared to 3CASC. Accordingly, 3CASC and 4CASC produce statistically independent keystreams from the initial configurations.

5.5.4 Cryptographic Properties of CASC 3N and CASC 4N

In addition to the tests above, the cryptographic properties study is a further method to evaluate the robustness, the randomness quality, the statistical features, and the confusion property of the proposed design. This part investigates the nonlinearity, algebraic degree, balancedness, resiliency, and correlation immunity. These properties are presented by tables from 5.4 to 5.13 for nonlinear and the mixing CAs rulesets of 3CASC and 4CASC. The nonlinear and mixing CA rulesets study relies on the computation of these properties for eight cells and nine cells, respectively, for three iterations, where the x_is refer to CA cells.

Nonlinear Block Cryptographic Properties

3N Ruleset: {30, 120, 180, 45, 30, 120, 180, 45}
4N Ruleset: {43350, 38490, 25500, 22185, 43350, 38490, 25500, 22185}

Mixing Function Block Cryptographic Properties

3N Ruleset: {30, 60, 90, 120, 150, 180, 240, 15, 45}
4N Ruleset: {43350, 49980, 42330, 38490, 27030, 25500, 65280, 255, 22185}

Table 5.4: Nonlinearity

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
1	2	2	2	2	2	2	2	2	3N
	4	4	4	4	4	4	4	4	4N
2	8	8	8	8	8	8	8	8	3N
	32	56	48	48	48	56	48	48	4N
3	44	40	44	40	36	40	44	40	3N
	464	432	432	448	440	440	448	400	4N

Table 5.5: Algebraic Degree

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
1	2	2	2	2	2	2	2	2	3N
	2	2	2	2	2	2	2	2	4N
2	3	3	3	3	3	3	3	3	3N
	3	3	4	3	4	3	4	3	4N
3	5	5	4	4	5	5	4	4	3N
	5	5	6	5	5	5	6	5	4N

Table 5.6: Resiliency

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
1	0	0	0	0	0	0	0	0	3N
	1	1	1	1	1	1	1	1	4N
2	0	1	0	0	1	1	0	0	3N
	1	1	0	0	1	1	0	0	4N
3	1	1	0	0	0	1	0	0	3N
	0	-1	0	0	0	-1	-1	1	4N

Table 5.7: Correlation Immunity

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
1	0	0	0	0	0	0	0	0	3N
	1	1	1	1	1	1	1	1	4N
2	0	1	0	0	1	1	0	0	3N
	1	1	0	0	1	1	0	0	4N
3	1	1	0	0	0	1	0	0	3N
	0	0	0	0	0	0	0	1	4N

Table 5.8: Balancedness

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
1	✓	✓	✓	✓	✓	✓	✓	✓	3N
	✓	✓	✓	✓	✓	✓	✓	✓	4N
2	✓	✓	✓	✓	✓	✓	✓	✓	3N
	✓	✓	✓	✓	✓	✓	✓	✓	4N
3	✓	✓	✓	✓	✓	✓	✓	✓	3N
	✓	×	✓	✓	✓	×	×	✓	4N

Table 5.9: Nonlinearity

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
1	2	0	0	2	0	2	0	0	2	3N
	4	0	0	4	0	4	0	0	4	4N
2	8	8	8	8	12	8	8	0	8	3N
	32	32	48	32	48	48	0	32	48	4N
3	32	32	48	48	48	32	32	32	32	3N
	384	256	384	448	448	384	384	384	384	4N

Table 5.10: Algebraic Degree

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
1	2	1	1	2	1	2	1	1	2	3N
	2	1	1	2	1	2	1	1	2	4N
2	2	2	2	3	2	2	2	1	3	3N
	3	2	2	3	2	2	1	2	3	4N
3	3	2	3	4	3	3	2	2	4	3N
	4	3	3	5	3	3	2	2	5	4N

Table 5.11: Resiliency

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
1	0	1	1	0	2	0	0	0	0	3N
	1	2	2	1	3	1	0	0	1	4N
2	2	0	2	1	0	0	0	0	0	3N
	1	2	2	2	0	-1	3	1	3	4N
3	1	0	1	0	2	0	0	0	0	3N
	0	1	2	2	0	-1	0	-1	1	4N

Table 5.12: Correlation Immunity

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
1	0	1	1	0	2	0	0	0	0	3N
	1	2	2	1	3	1	0	0	1	4N
2	2	0	2	1	0	0	0	0	0	3N
	1	2	2	2	0	0	3	1	3	4N
3	1	0	1	0	2	0	0	0	0	3N
	0	1	2	2	0	0	0	0	1	4N

Table 5.13: Balancedness

Iteration	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
1	✓	✓	✓	✓	✓	✓	✓	✓	✓	3N
	✓	✓	✓	✓	✓	✓	✓	✓	✓	4N
2	✓	✓	✓	✓	✓	✓	✓	✓	✓	3N
	✓	✓	✓	✓	✓	×	✓	✓	✓	4N
3	✓	✓	✓	✓	✓	✓	✓	✓	✓	3N
	✓	✓	✓	✓	✓	×	✓	×	✓	4N

These tables show that, except for resiliency and correlation immunity, balancedness is preserved. As well, algebraic degree and nonlinearity raise through iterations for 3CASC and 4CASC. The diminution of both resiliency and correlation immunity is due to their conflicting relation with nonlinearity and algebraic degree[19]. Consequently, one should reach a trade-off. In the case of primitives like stream ciphers, to meet a high nonlinearity and algebraic degree are more interesting for security issues and randomness features. According to the results presented in the tables, 4CASC has higher nonlinearity and algebraic degree compared to 3CASC. Fewer iterations in 4CASC are enough to achieve the same security level provided by a higher number of evolutions in 3CASC. Consequently, passing from three neighborhood rulesets (3CASC) to four neighborhood rulesets (4CASC) enhances the statistical and the cryptographic properties of the proposed design.

5.6 Conclusion

This chapter presents a newly proposed stream cipher titled N neighborhood Cellular Automata-based Stream Cipher (NCASC). This cipher takes inspiration from the Grain design by substituting feedback shift registers with CAs. It consists of three fundamental components, namely a linear CA, a nonlinear CA of only nonlinear rules, and a mixing CA including linear and nonlinear rules replacing LFSR, NFSR, and mixing function, respectively. This chapter provides a detailed description of the stream cipher designs and compares two versions 3CASC and 4CASC. This study was performed referring to the statistical characteristics, cryptographic properties, and security analysis to deduce the beneficial and negative impacts of both versions. Accordingly, 4CASC displays better statistical characteristics and higher algebraic degree and nonlinearity compared to 3CASC. From another perspective, 3CASC has a higher level of correlation immunity and resiliency. Referring to the conclusions of the study conducted in this chapter, future propositions will combine 3-neighborhood CAs with 4-neighborhood CAs to benefit from their properties to achieve a higher security level and randomness quality. Also, the current study could be expanded to another level of neighborhood configuration.

Part II

Integrity

Chapter 6

LCAHASH 1.1 and HCAHF Hash Functions Families

6.1 Introduction

Cryptographic hash functions are symmetric cryptographic primitives that guarantee data integrity used for passwords storage, virus detection, or intrusion detection. Also, they could be involved in the design of PRNGs, digital signatures, or Message Authentication Codes (MACs). The current chapter presents two hash functions, namely LCAHASH 1.1 and HCAHF. The first one is an enhanced version of the LCAHASH[10] design that involves modular computation and a cellular automaton, which makes use of a 7-variable or 8-variable global rule generated during the design processing depending on the digest's size, that evolves only one iteration. LCAHASH 1.1 involves a cellular automaton that evolves using rules 30 and 90 for 128 or 256 iterations, which makes the global function a 128-variable /256-variable one, to ensure a higher security level and better statistical characteristics compared to LCAHASH. The second hash function, namely HCAHF, is a CA-based hash function that makes use of different features providing better security against known attacks and greater cryptographic and statistical properties. Three functions construct the HCAHF system: the pre-processing step, the compression function, and the last transformation. Each step includes features producing security against several attacks and high quality of randomness.

This chapter is organized as follows: In section 6.2, we mention some related work. In section 6.3 and 6.4, we describe the old and the new design of LCAHASH, respectively. Next, HCAHF design is detailed in section 6.5. Afterwards, we present the security analysis of both LCAHASH 1.1 and HCAHF in section 6.6. Section 6.7 provides their experimental results. Finally, a conclusion is presented in Section 6.8.

6.2 Related Work

In[103], Damgard proposed a technique to design free-collision hash functions and presented three ways to use his proposition, namely a Knapsack problem-based design, a Wolfram's PRNG based design including CAs, and modular squaring-based construction. Later, in [104], Daemen et al. provided the weak points of the system presented in [103] and suggested a hash functions framework together with a CA-based hash function titled CellHash. The work of[105] introduces Subhash that is an improved version of CellHash. In [106], Chang attacked the designs of [104] and [105] and proposed robust versions of them. Later, Mihalevic et al. [?] developed a CA-based hash functions family without providing the rules involved. In [8], Jeon included linear and nonlinear rules in the design of a CA-based hash function. [9] proposed an efficient and secure sponge construction inspired hash function involving CAs. This chapter presents two efficient hash functions providing a security level at least like other hash functions involving other features, such as SPONGENT[108], GLUON[109], QUARK[110], PHOTON[111], and SHA-3[112]. The first construction, titled LCAHASH1.1, is based on a Merkle-Damgard structure that improves the hash function proposed by Charifa et al. in [10] LCAHASH, which has some flaws concerning cryptographic and statistical characteristics. A single evolution of a uniform CA that uses a 7-variables (128-bit) or 8-variables (256-bit) boolean function as the global transition rule. Instead, LCAHASH1.1 makes use of a hybrid CA evolving 128 or 256 times using a linear and a nonlinear transition rule to increase the system's statistical properties and security. The second design is a wide-pipe Merkle Damgard inspired hash functions family that includes additional features like the hybrid CA and the padding algorithm providing resistance to known attacks and satisfying cryptographic properties and statistical characteristics.

6.3 The LCAHASH 1.0 Design

This section describes the LCAHASH 1.0 design [10]. The LCAHASH 1.0 algorithm takes as input an arbitrary length message and produces a 128-bit or 256-bit digest. At first, a padding is applied to the binary message M if the size is not a multiple of 128 or 256. Next, M is split into 128/256-bit blocks (M_1, ..., M_m). Then, a random n-bit initial value IV_1 is xored with a randomly chosen block M_{index}. Afterward, a carefully chosen 13-bit prime number N is used to compute $R_i = M_i mod N$. Next, R_is are appended to M' that is split into n-bit blocks and padded if necessary. Next,the M_i's are xored to IV_2 to produce M_{rule}. Afterward, M_{rule} is evolved using the 7-variable or 8-variable rule M_{rule} to get the digest. Figure X illustrates the LCAHASH 1.0 process. Algorithm 9 provides the pseudo-code of the LCAHASH 1.0

Algorithm 9 Pseudo-code LCAHASH 1.0

```
Input: Message M, $IV_1$, $IV_2$, index, prime number N
Output: digest
Begin
Split M into n-bits blocks                 ▷ n =128 or 256, $\{M_1, ..., M_m\}$
if $M_m$ is not a multiple of n then
    Pad $M_m$
end if
$M_{index} \leftarrow M_{index} \oplus IV_1$
for $1 \leq i \leq m$ do
    $R_i \leftarrow M_i mod N$
end for
$M' \leftarrow R_1 ... R_m$
if $M' \leq n$ then
    $M_{rule} \leftarrow M' IV_2$
else
    Split M' into n-bits blocks
    $M_{rule} \leftarrow IV_2 \oplus M_1' \oplus ... \oplus M_k'$
end if
$Digest \leftarrow Evol_{M_{rule}}(M_{rule})$
End
```

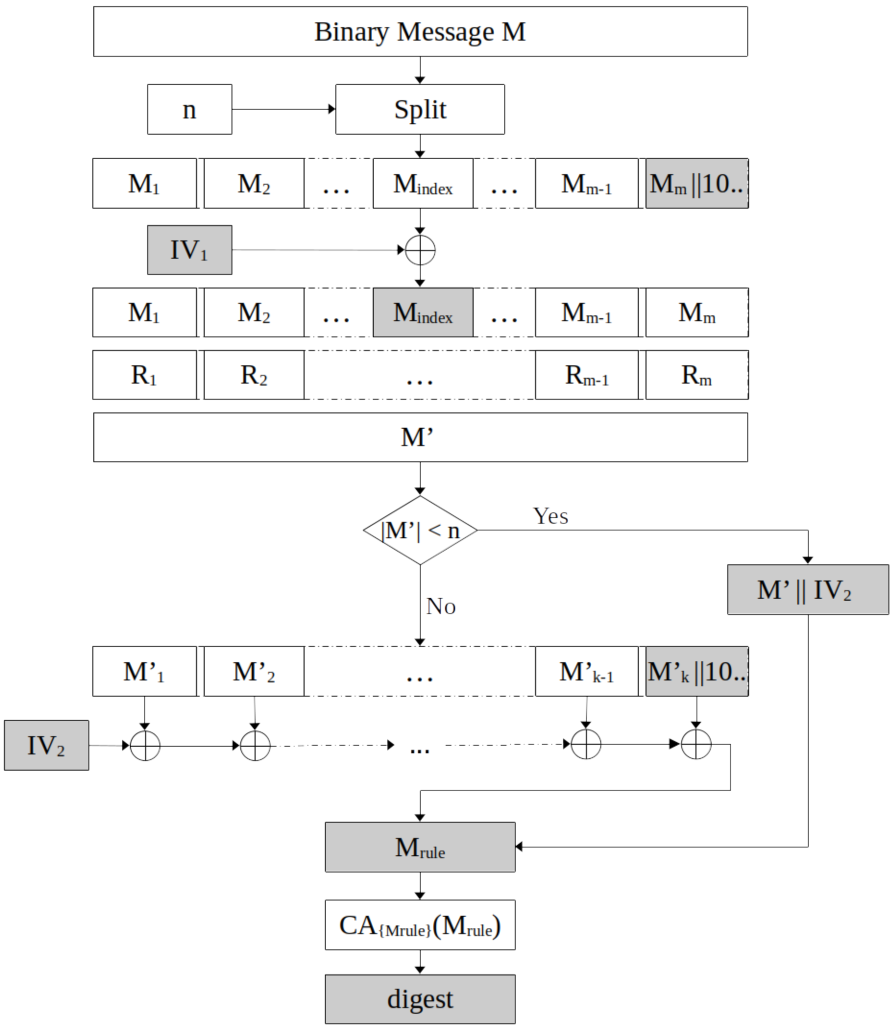


Figure 6.1: LCAHASH 1.0 design

6.4 The LCAHASH 1.1 Design

This section describes the new design of LCAHASH. The LCAHASH 1.1 algorithm proceeds as follows: it takes as input an arbitrary length message M and produces an n-bits digest, where n is either 128 or 256. M is first divided into n-bit blocks, and padded if necessary. Afterwards, a randomly chosen block M_{index} is Xored with the random n-bit initial value IV_1. In the forthcoming phase, a 13-bit prime number N to compute R_is $R_i = M_i mod N$, which are appended to form M'. If M' is not a multiple of n, then it is padded and split into n-bit blocks $\{M'_1$, ..., $M'_k\}$. Next, all the M'_is are Xored with each other together with the n-bit IV_2 resulting M_{evol}. Lastly, M_{evol} underges n CA evolutions using the rules 30,90 to get the digest.

Algorithm 10 Pseudo-code LCAHASH 1.1

```
**Input**: Message M, $IV_1$, $IV_2$, index, prime number N
**Output**: digest
**Begin**
Split M into n-bits blocks                    ▷ n =128 or 256, $\{M_1, ..., M_m\}$
**if** $M$ is not a multiple of n **then**
    Pad $M_m$
**end if**
$M_{index} \leftarrow M_{index} \oplus IV_1$
**for** $1 \leq i \leq m$ **do**
    $R_i \leftarrow M_i modN$
**end for**
$M' \leftarrow R_1 ... R_m$
**if** $M'$ is not a multiple of n **then**
    Pad $M'$
**end if**
Split M' into n-bits blocks
$M_{Evol} \leftarrow IV_2 \oplus M'_1 \oplus ... \oplus M'_k$
$Digest \leftarrow Evol_{\{30,90\}}(M_{Evol}, n$ iterations )
**End**
```

Figure 1 displays the different steps of LCAHASH 1.1.

Figure 2 shows a sample evolution of M_{evol} using the non-uniform cellular automaton with ruleset {30,90}.

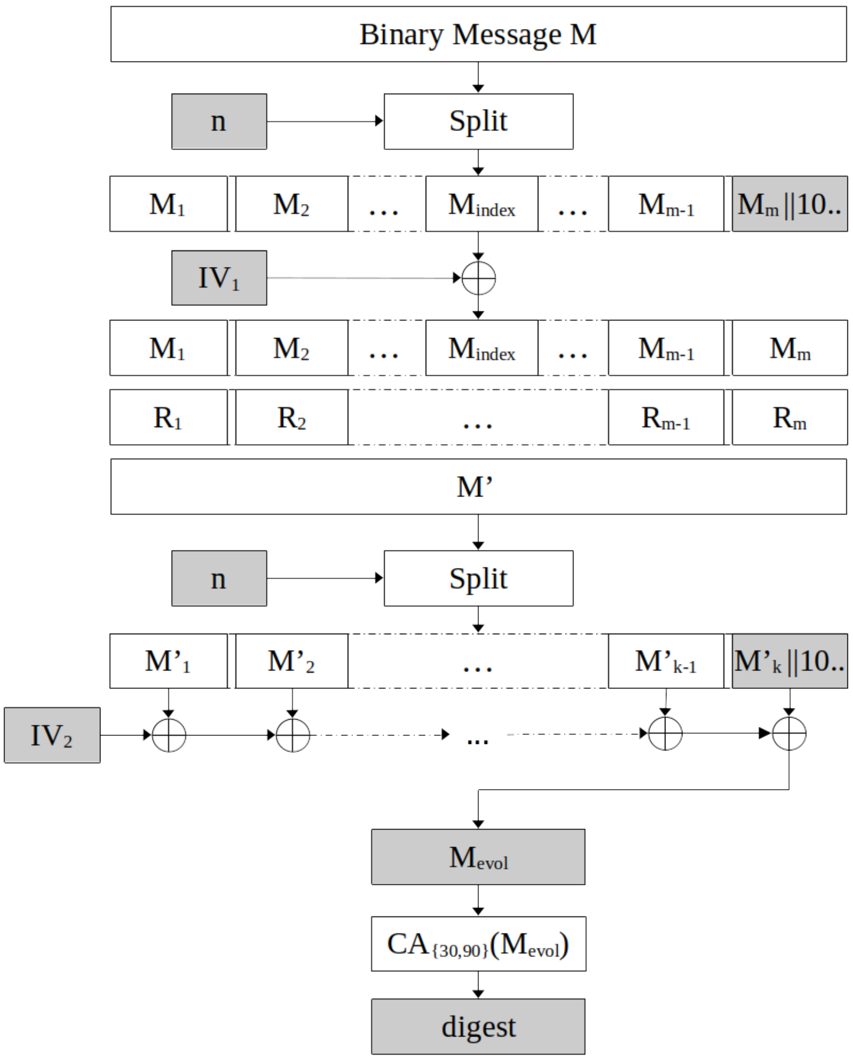


Figure 6.2: LCAHASH 1.1 steps

6.5 HCAHF Design

This section describes a hash function design inspired by the wide-pipe Merkle Damgard construction [114]. HCAHF is the general structure that outputs m-bit digests, here a specific design HCAHF-256 is detailed refering to the version that produces 256-bit digests. The HCAHF design comprises three steps, namely a preprocessing step, a CA-based compression function, and a hybrid CA-based transformation produces the digest. In the rest of this section, HCAHF-256 steps are depicted.

6.5.1 Preprocessing Phase

Padding Scheme

This step purpose is to provide security against the length-extension attack[113]. It is applied to every message, even if its size is a multiple of 256. The padding algorithm involved in this steps is titled Merkle Damgard Strengthening [114]. It appends to the rightmost bit a '1' bit followed by zeros followed by the message length encoding represented over 64 bits to make the message length a multiple of 256.

Message Splitting

After applying the MD-Strengthening padding, the resulting message is then divided into 256-bit blocks.

Salt

At this stage, a random salt value is generated using a PRNG. Then it is inserted before the first block M_1. The main purpose of this parameter is to minimise the collisions possibility avoid pre-computation attacks like dictionary attack due to the fact that the same message could hash to two different digests $h1 = h(m, salt1)$ and $h2 = h(m, salt2)$ if different salt values are used $salt_1 \neq salt_2$ [113].

6.5.2 Compression Phase

This step involves a compression function f, which combines CA evolotions refered by E and the XOR operation. f is applied iteratively over the blocks together with a random IV set to be the initial chaining variable h_0. The IV was generated by the ThreadedSeedGenerator class provided by the bouncy castle java library. The last chaining variable is input to the last transformation to provide the final digest. (n+1) 256-bit blocks are iterated to compute the compression function result to produce a 256-bit block.

The Function E

This function refers to the cellular automaton involving 3-neighborhood rules, such that the boundary conditions are set to be periodic. Each 256-bit block M_i goes through 128 evolutions by applying the rule R_i determined by the eight leftmost bit of M_i. (ex: if the lesftmost 8-bits of Mi are '01011010' then $R_i = 90$. Each block M_i is evolved 128 times using a rule R_i as follows:

$$e_0 = E(salt, R_0)$$

$$e_1 = E(M_1, R_1)$$

$$...$$

$$e_n = E(M_n, R_n)$$

If R_is are have not good cryptographic properties, then it is replaced by a randomly chosen rule from the class 3 and 4 of ones having satisfying properties (R_E={30, 45, 60, 75, 86, 89, 90, 101, 102, 105, 106, 120, 135, 147, 149, 150, 165, 169, 195, 225}).

The XOR Function

Each block M_i is evolved and XORed with the preceeding block evolution result.

The Function f

The compression function f could be expressed by the following formula:

$$h_0 = IV$$

$$h_1 = f(e_0, h_0)$$

$$...$$

$$h_i = f(e_{i-1}, h_{i-1})$$

$$\dots$$

$$h_{n+1} = f(e_n, h_n)$$

6.5.3 Transformation Phase

This step involves the function T that takes as input the output of the previous step to produce the digest. T represents a non-uniform CA that evolves 128 times by means of the set of rules $R_{digest} = \{30, 90, 150, 30, 135, 30, 90, 150\}$, which is defined according to the guidelines provided by [19].

$$digest = T(M_{compressed}, R_{digest})$$

During the transformation phase, a function T is applied to $M_{compressed}$, the 256-bit output of the previous phase.

The number of evolutions in this construction is defined to be n/2=128 to guarantee a high CA period according to [26]. Figure 6.3 summarizes the different steps of HCAHF.

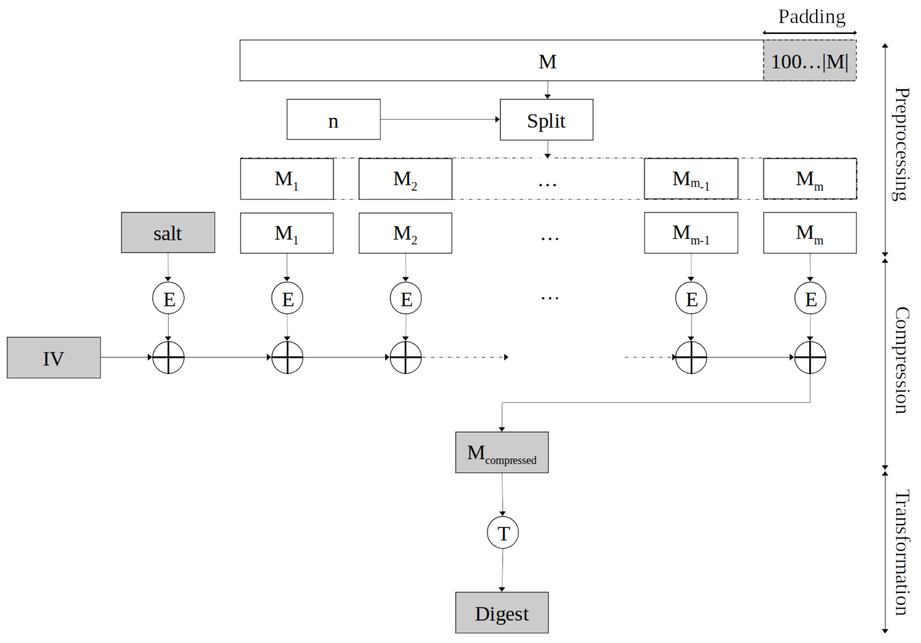


Figure 6.3: HCAHF Design

6.5.4 Pseudo-Code for The Padding Scheme of HCAHF-256

Algorithm 11 details the padding scheme used for HCAHF-256.

Algorithm 11 Pseudo-code of the padding scheme of HCAHF-256

```
Input: M the message to pad
Output: $M_{padded}$ the padded message
Begin
m ← sizeOf(M) mod 256
x ← 256 - m
if sizeOf(M) is multiple of 256 then
    NbZero ← 191
else if sizeOf(M) is not a multiple of 256 then
    if x < 65 then
        y ← x + 256
        NbZero ← y - 65
    else
        NbZero ← x - 65
    end if
end if
$M_{padded} \leftarrow M10^{NbZero}sizeOf(M) \in \{0,1\}^{64}$
return $M_{padded}$
End
```

6.5.5 Pseudo-Code for The Generation Mechanism of HCAHF-256

Algorithm 12 shows the different steps by which a 256-bit digest is generated by HCAHF-256.

6.6 Security Analysis

6.6.1 LCAHASH 1.1 Complexity

To pad and divide an L-bit message M into n-bit blocks it takes $n-\frac{L}{n}$ steps. To apply the XOR of IV1 with M_{index}, n mod 2 addition operations are needed. The computation of R_is needs $m \times nmod2$ divisions, such that m is the number of blocks in M. Padding and splitting the L'-bit M' into n-bit blocks demand at most $n-\frac{L'}{n}$ operations. Next, the XOR of M_i's with the IV2 needs k mod 2 additions. At last, in the CA evolution of M_{evol}, n^2 steps are necessary. To conclude, the LCAHASH 1.1 complexity is O(n^2).

Algorithm 12 Pseudo-code of HCAHF-256

```
Input: M the message to hash
Output: digest
Begin
IV ← randomSequence(256)
salt ← randomSequence(256)
ruleSet← {30, 90, 150, 30, 135, 30, 90, 150}
$M_{padded} \leftarrow padding(M)$
splittedM ← $split(M_{padded}, 256)$
saltedM ← insert (salt, splittedM)
Xor ← IV
for each block b in saltedM do
    if IntegerValue(8firstBits(b)) is in S the set of class 3 and 4 rules then
        $rules_i \leftarrow IntegerValue(8firstBits(b))$
    else
        $rules_i \leftarrow randomRule(S)$
    end if
    $E \leftarrow evolv(b, rules_i, 128, periodicBoundaries)$
    $Xor \leftarrow Xor \oplus E$
end for
$s \leftarrow Xor$
$evolution \leftarrow CAEvolution(s, ruleSet, 128)$
$digest \leftarrow evolution$
return digest
End
```

6.6.2 HCAHF Complexity

This part provides the HCAHF complexity. To padd a message M, it takes at most $n + 64$ steps, such that n is the length of blocks. To divide the padded message, it needs $m = \frac{Len}{n}$, such that Len is the length of the padded message. To generate the salt and IV it requires 2n steps. The application of the function E to each block needs $(m + 1) \times n \times \frac{n}{2}$ steps. The compression step uses $m + 1 \times n$ mod 2 additions. The transformation T involves $n \times \frac{n}{2}$ steps. To conclude, the complexity of HCAHF is of order $O(m \times n^2)$, such that n is the digest length and m is the number of blocks.

6.6.3 Pre-image and Second Pre-image Resistance

The LCAHASH 1.1 security is located in the CA global function since the IVs, N and index are supposed to be known. Then, it is easy to deduce that it requires $n \times 2^n$, such that n is the blocks size, step to determine the CA global function. Consequently, LCAHASH 1.1 is robust against pre-image and second pre-image attacks. Similarly, the security of HCAHF depends of the compression function f and the transformation T. To achieve the pre-image and second pre-image attacks against HCAHF-256 , it needs 2^{256} steps. Consequently, HCAHF-256 is secure againts these attacks.

6.6.4 Collision Resistance

Refering to te birthday attack and the LCAHASH 1.1 parameters, to get two messages M and M' that hash to the same digest it needs $2^{\frac{n}{2}}$ trials. Consequently, in the case of LCAHASH 1.1, n could be either 128 or 256, $2^{\frac{128}{2}}$ or $2^{\frac{256}{2}}$ steps are proceeded to find a collision. Thus, LCAHASH 1.1 resists collision attacks. Similarly, in the case of HCAHF-256, to find a collision, it takes 2^{128} steps. Then, HCAHF-256 is secure against this attack.

6.6.5 Cryptanalytic Attacks

This type of attacks is addressed to the hash functions designs, especially the compression function involved. These attacks aim to decrease the algorithm complexity which minimise the brute force attacks complexity. Among these attacks there are the length-extension attacks[116], herding attack[119], multicollision attacks[115], and fixed-point attacks[117]. Preventing these attacks requires inclusion of some features. In the HCAHF design, a padding scheme titled Merkle Damgard Strengthening padding is involved in the first step to make it robust against length-extension attacks. In addition, the salt value included increase the complexity of the multicollision attacks, the fix point attack, length extension attack, as well as herding attack. The uniform CAs involved the function E, which evolve using either linear or nonlinear rules depending on the blocks, and the non-uniform CA used in the T transformation are also useful to avoid some attacks like linear and differential attacks. Moreover, the nonlinear CA ensure satisfying confusion, while the diffusion is provided by the XOR operation and checked by the avalanche effect test. Moreover, the statistical tests show that the output of HCAHF-256 and LCAHASH 1.1 are independent to the output. Consequently, the HCAHF and LCAHASH 1.1 designs are resistant to statistical attacks.

6.7 Results

6.7.1 LCAHASH 1.1 Avalanche Effect

The avalanche effect test of LCAHASH 1.1 proceeds as follows. At first, one hundred of 1024-bit messages are generated. Next, the corresponding digests are computed $\{h_1, ..., h_{100}\}$. Thereafter, for each message M_i, one bit position is modified each time and the corresponding digest is then computed, $\{\{h_1^{(1)}, ..., h_1^{(1024)}\}, ..., \{h_{100}^{(1)}, ..., h_{100}^{(1024)}\}\}$. Next, the hamming distances $H_{ij} = H(h_j, h_j^{(i)})$, $1 \leq i \leq 1024$ and $1 \leq j \leq 100$ of the digests produced from the changed messages $\{\{h_1^{(1)}, ..., h_1^{(1024)}\}, ..., \{h_{100}^{(1)}, ..., h_{100}^{(1024)}\}\}$ and the original one $\{h_1, ..., h_{100}\}$. Then, for each i $\in \{1, ..., 1024\}$, the average of the hamming distances is computed $AvgHi = \sum_{j=1}^{100} \frac{H_{ij}}{100}$. At last, the average avalanche effect is deduced refering to the formula

$$AvgAvalanche(M_i, M_i^{(j)}) = \frac{AvgHi}{n} \times 100$$

such that $1 \leq i \leq 1024$ and $1 \leq j \leq 100$. Figure 6.4 and Table 6.1 illustrate the LCAHASH 1.1 avalanche effect results for n=128.

6.7.2 HCAHF-256 Avalanche Effect

This section displays the HCAHF-256 avalanche test results. To get these results, one hundred 1024-bit messages are randomly generated. Next, their

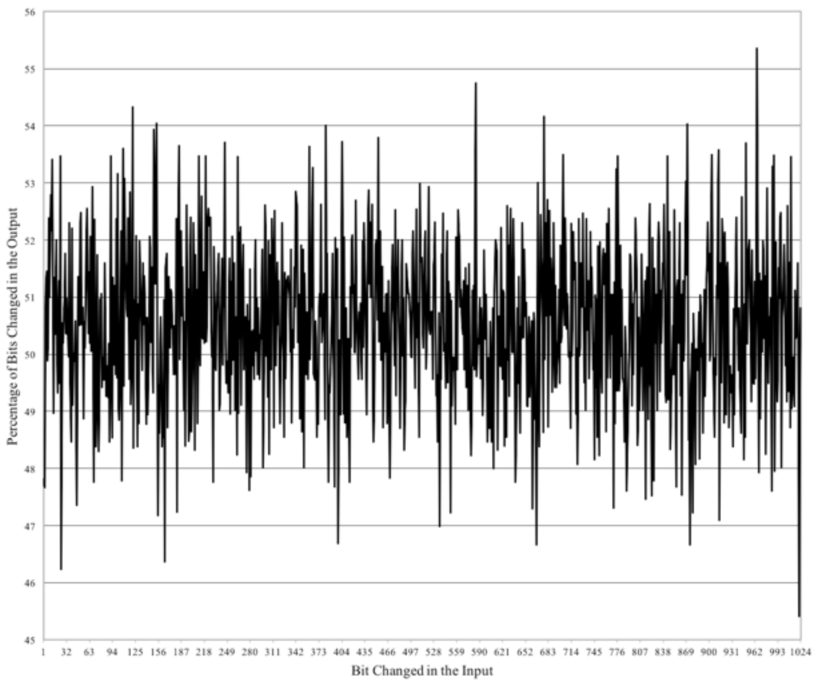


Figure 6.4: Avalanche Effect for LCAHASH (128-bit version)

Table 6.1: Min, Max and Mean Average Hamming Distances

Min	45.41%
Max	55.35%
Mean	50.51%

digests are calculated $\{H_1, H_2, ..., H_{100}\}$. Thereafter, for each message M_i, one bit position is modified each time and the corresponding digest is computed $\{\{H_1^{(1)}, ..., H_1^{(1024)}\}, ..., \{H_{100}^{(1)}, ..., H_{100}^{(1024)}\}\}$. Furthermore, the Hammig distances $Hd_{ij} = Hd(H_j, H_j^{(}i))$, $i \in \{1, ..., 1024\}$, $j \in \{1, ..., 100\}$ of the outputs produced from the changed messages $\{\{H_1^{(1)}, ..., H_1^{(1024)}\}, ..., \{H_{100}^{(1)}, ..., H_{100}^{(1024)}\}\}$ and the original ones $\{H_1, H_2, ..., H_{100}\}$ are computed. Then, for each $i \in \{1, ..., 1024\}$, the average of the hammings distances is found by the formula $AvgHd_i = \frac{\sum_{j=1}^{100}}{100}$. Finally, the average avalanche effect is deduced refering to the formula $AvAvalanche(M_j, M_j^i) = \frac{AvgHd_i}{256} \times 100$, $i \in \{1, ..., 1024\}$, $j \in \{1, ..., 100\}$. Figure 6.5 illustrates the avalanche effect test results of HCAHF-256. According to this figure , HCAHF-256 displays satisfying results since the values range around 50% , consequently, the hash values provided by HCAHF-256 are statistically independent from the inputs.

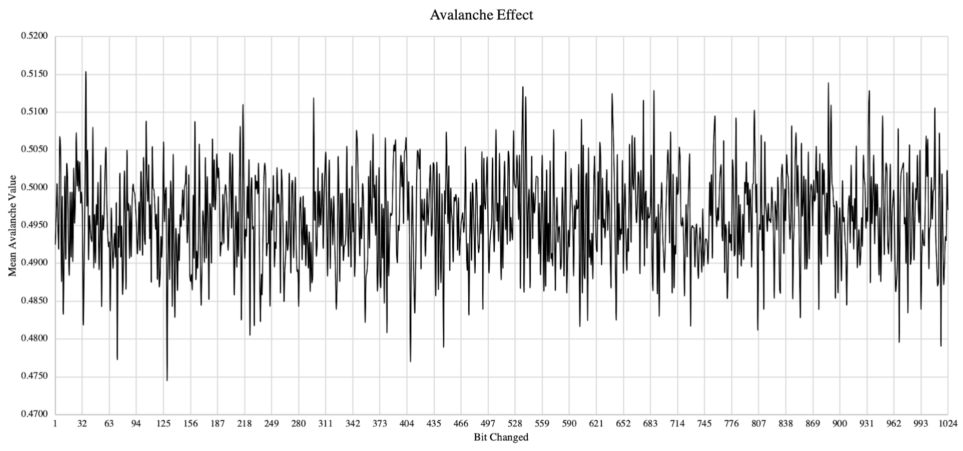


Figure 6.5: Avalanche Test Results

6.7.3 Statistical Tests

Dieharder Tests

In this section, the statistical tests provided by the Dieharder battery of tests are applied to the LCAHASH 1.1 and HCAHF-256 outputs to study the randomness quality produced. Two 10 MB files are filled by 78128 digests computed using the 128 version of LCAHASH 1.1 and 39063 digests produced using HCAHF-256, respectively. Table 6.2 presents the Dieharder results. The p-values should range in [α , $1 - \alpha$], where $\alpha = 0.005$ is the significance level. Accordingly, all the p-values are in the range $0.008 > \alpha$ and $0.9596 < 1 - \alpha$. LCAHASH 1.1 and HCAHF-256 passed all the tests, which means that both designs have a random behaviour and satisfying statistical properties making statistical attacks hard to achieve.

NIST Statistical Test Suite (STS)

This subsection presents the HCAHF-256 hash values NIST STS results to check the randomness quality provided. At first a 10 MB file is generated using 39063 256-bit hash values ($HCAHF_{256}(M^1)$, ..., $HCAHF_{256}(M^{39063})$). Then it is fed to the test suite. The results of each test are displayed in Table 6.3. As known from the previous chapters, the p-values in NIST STS should range between α and $1 - \alpha$, such that the significance level α is set to be 0.001. Accordingly, all applicable tests are succeeded. Five out of seventeen tests are not applicable because of the size of the hash values (256 bits). Longer sequences are required, for instance, the random excursions test requires at least 1000000-bit sequences to get results.

The results of NIST STS and dieharder tests imply that HCAHF-256 has a pseudo-random behaviour, which is a fundamental feature of cryptographic hash functions. In other words, HCAHF-256 hash values are indistinguishable from random sequences.

6.7.4 Cryptographic Properties of the Class 3 and 4 Rules and R_T

A complementary study is provided in this part. The cryptographic properties of the class 3 and 4 rules that could be invovled in the E function, in addition to those of R_T are presented here for 8-cells and 3 evolutions with the periodic boundary condition. Table 6.4 displays the class 3 and 4 rules cryptographic properties for one iteration, such that NL, CI, AD, Res, and Bal refer to nonlinearity, correlation immunity, algebraic degree, resiliency, and balancedness respectively.

From this table, we conclude the set of the rules that could be involved in E {30, 45, 60, 75, 86, 89, 90, 101, 102, 105, 106, 120, 135, 147, 149, 150, 165, 169, 195, 225}. Tables from 6.5 to 6.9 show the variation of the cryptographic properties through multiple evolutions.

Table 6.2: Diehard Test Suite

Tests	LCAHASH 1.1		HCAHF-256	
	P-values	Pass?	P-values	Pass ?
Diehard birthdays	0.3944	✓	0.8559	✓
Diehard operm5	0.5331	✓	0.9309	✓
Diehard rank 32x32	0.5760	✓	0.3301	✓
Diehard rank 6x8	0.7220	✓	0.8151	✓
Diehard bitstream	0.9483	✓	0.7112	✓
Diehard opso	0.1375	✓	0.9008	✓
Diehard opso	0.9219	✓	0.1161	✓
Diehard dna	0.6396	✓	0.6635	✓
Diehard count 1s str	0.4911	✓	0.0808	✓
Diehard count1s byt	0.9394	✓	0.0082	✓
Diehard parking lot	0.2254	✓	0.7936	✓
Diehard 2d sphere	0.6463	✓	0.0254	✓
Diehard 3d sphere	0.7971	✓	0.8868	✓
Diehard squeeze	0.8315	✓	0.9114	✓
Diehard sums	0.1793	✓	0.0138	✓
Diehard runs	0.7973	✓	0.4253	✓
Diehard craps	0.4773	✓	0.5868	✓
Marsaglia tsang gcd	0.8530	✓	0.3477	✓
Sts monobit	0.9595	✓	0.7161	✓
Sts runs	0.6414	✓	0.2438	✓
Sts serial	0.5015	✓	0.5342	✓
Rgb bitdist	0.4625	✓	0.5511	✓
Rgb minimum distance	0.6379	✓	0.7152	✓
Rgb permutations	0.5922	✓	0.5	✓
Rgb lagged sum	0.4944	✓	0.533	✓
Rgb kstest test	0.6942	✓	0.5784	✓
Dab bytedistrib	0.5645	✓	0.5964	✓
Dab dct	0.4265	✓	0.0541	✓
Dab filltree	0.4165	✓	0.7417	✓
Dab filltree2	0.5458	✓	0.4559	✓
Dab monobit2	0.6795	✓	0.0432	✓

Table 6.3: NIST STS Results

Test name	p-value	Pass?
The Frequency (Monobit) Test	0.4963	✓
Frequency Test within a Block	0.4888	✓
The Runs Test	0.4927	✓
Tests for the Longest-Run-of-Ones in a Block	0.4954	✓
The Binary Matrix Rank Test	-	N/A
The Discrete Fourier Transform (Spectral) Test	0.4815	✓
The Non-Overlapping Template Matching Test	0.6973	✓
The Overlapping Template Matching Test	-	N/A
Maurer's "Universal Statistical" Test	-	N/A
The Linear Complexity Test	0.5644	✓
The Serial p-value1 Test	0.4915	✓
The Serial p-value2 Test	0.4993	✓
The Approximate Entropy Test	0.4914	✓
The Cumulative Sums (Cusums) Forward Test	0.514	✓
The Cumulative Sums (Cusums) Reverse Test	0.5132	✓
The Random Excursions Test	-	N/A
The Random Excursions Variant Test	-	N/A

According to these tables, the algebraic degree and the nonlinearity keep increasing and balancedness is maintained with iterations. Unlike the correlation immunity and resiliency which decrease after two evolutions. This decrease is justified by the fact that some properties are contradictory, for example high resiliency could not be met together with high nonlinearity by the same ruleset. As a result, for hash functions requirements, the focus is on nonlinearity, algebraic degree and balancedness more than the other properties to get a secure hash system.

6.7.5 Performance

To test the LCAHASH 1.1 performance, the java implementation was run over an Intel Core i3-4010 32-bit processor clocked at 1.7 GHz with 4 Go of RAM. Table 6.10 presents the results of LCAHASH 1.0 and LCAHASH 1.1. According to this table, the LCAHASH 1.1 displays better results compared to LCAHASH 1.0.

Table 6.4: Cryptographic Properties of Class 3 and 4 ECAs

Rule	**NL**	**CI**	**AD**	**Res**	**Bal**
18	2	0	2	-1	×
22	1	0	3	-1	×
30	2	0	2	0	✓
41	1	0	3	-1	×
45	2	0	2	0	✓
60	0	1	1	1	✓
75	2	0	2	0	✓
86	2	0	2	0	✓
89	2	0	2	0	✓
90	0	1	1	1	✓
101	2	0	2	0	✓
102	0	1	1	1	✓
105	0	2	1	2	✓
106	2	0	2	0	✓
110	1	0	3	0	×
120	2	0	2	0	✓
121	1	0	3	0	×
122	1	0	3	0	×
124	1	0	3	0	×
126	2	1	2	1	×
128	1	0	3	0	×
135	2	0	2	0	✓
137	1	0	3	0	×
146	1	0	3	0	×
147	2	0	2	0	✓
149	2	0	2	0	✓
150	0	2	1	2	✓
151	1	0	3	0	×
161	1	0	3	0	×
165	0	1	1	1	✓
169	2	0	2	0	✓
182	1	0	3	0	×
183	1	0	2	0	×
193	1	0	3	0	×
195	0	1	1	1	✓
225	2	0	2	0	✓

Table 6.5: Algebraic Degree

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	1	1	2	2	2	1	1
2	3	2	2	3	3	3	2	2
3	4	3	3	4	5	4	3	3

Table 6.6: Nonlinearity

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	2	0	0	2	2	2	0	0
2	8	8	8	8	4	8	8	8
3	32	48	48	48	16	36	48	48

Table 6.7: Correlation Immunity

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	0	0	1	2
2	0	2	2	0	0	0	2	2
3	0	0	0	0	0	0	1	1

Table 6.8: Resiliency

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	1	2	0	0	0	1	2
2	0	2	2	0	0	0	2	2
3	0	0	0	0	0	0	1	1

Table 6.9: Balancedness

Iterations	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓

Table 6.10: Software Performance Of LCAHASH 1.1

Hash function	**Output size (bit)**	**Cycle per byte (cpb)**	**Clock (GHz)**
LCAHASH 1.0	128	324	1.7
	256	374	1.7
LCAHASH 1.1	128	995	1.7
	256	2844	1.7
GLUON	112	1951	2.66
U-QUARK	128	43373	2.66
D-QUARK	160	53103	2.66
S-QUARK	224	25142	2.66
PHOTON	80	1243	2.66

6.8 Conclusion

In this chapter, two hash functions are presented. The first one, LCAHASH 1.1, enhances the LCAHAHSH 1.0 design by replacing the single CA evolution using a 7 or 8-variable boolean function to be the rule defined during the hashing process of LCAHASH 1.0 by n CA evolutions using the ruleset 30,90, which means that the global transition function comprise n-variable rules, where n is the digest size. The motivation of this contribution is to improve the statistical characteristics and the cryptographic properties of the original design. LCAHASH 1.1 is a cryptographic hash function providing good statistical characteristics from the results and has better performance and cryptographic properties compared to the LCAHASH 1.0 design. The second CA-based hash functions family is titled HCAHF. It comprises three steps, namely the preprocessing step, the compression function, and a final transformation. Firstly, the preprocessing step starts with the MD strengthening padding followed by the split of the padded message and the introduction of the salt. This step makes the system stronger to attacks like length-extension attack, herding attack and other ones. Secondly, the compression step involves a different CA for each block together with the XOR of their results. This step produces good confusion and diffusion because of the characteristics of the operations included. Lastly, the final transformation includes a non-uniform CA that evolves by means of a ruleset having good cryptographic properties. This step gives the system more confusion and the randomness properties, which were checked by the statistical tests and proved that HCAHF has good statistical characteristics in addition to its resistance to attacks. Future work could include a hardware implementation of HCAHF and could also check its applicability within blockchain.

Conclusion

This thesis objective is to achieve two fundamental information security goals, confidentiality and integrity, using cellular automata to benefit from their characteristics providing complex behaviour with fast, simple and parallel computations. The first four contributions guarantee data confidentiality. The first one is Partition Ciphering System(PCS), which is inspired by an adaptation of the Equal Piles Problem to provide security against statistical attacks and brute force attacks. The motivation of this construction was the combination with SEC extension to binary blocks, and its impact on the ability to apply the statistical attacks. As expected, it is challenging to make use of this type of attacks to decrypt or find out the secret key of the plaintext. For a more in-depth study, an extended version of PCS, called Cellular Automaton based Partition Ciphering System(CA-PCS), was proposed. CA-PCS, the second contribution of this thesis, involves a non-uniform cellular automaton using a carefully chosen ruleset, and a random permutation. Each included feature impacts the CA-PCS strenght, better confusion, diffusion, and security are provided. For instance, the rules used by the cellular automaton have a high nonlinearity and algebraic degree, which means that attacks like linear and differential cryptanalysis are challenging. The third contribution, namely the CFA PRNGs family, follows a general design that includes a hash function, a block cipher and a cellular automaton. In CFA-256 specific implementation, SHA3-256 and AES-256 together with a non-uniform cellular automaton are involved to produce a strong PRNG satisfying cryptographic requirements, such as randomness, unpredictability, high cycle length, statistical properties, as well as security against known attacks. This design could be adapted to fit with different applications scales. For instance, in lightweight cryptography, the hash function and the block cipher will be lightweight designs and accordingly, the cellular automaton size could be decreased to the desired requirements, the rules and the number of evolutions could also be updated. The fourth contribution concerns the stream cipher designs NCASC inspired by Grain cipher, the eStream project finalist, such that N refers to the number of neighbors cells. This design replace feedback shift registers and the mixing function by linear and nonlinear cellular automata. It consists of three CAs, a linear CA, a Nonlinear CA involving only nonlinear rules, and a mixing CA involvine a CA combining linear and nonlinear rules. The first version, 3CASC, uses 3-neighborhood rules having satisfying behaviour and cryptographic properties. The secon version, 4CASC, uses 4-neighborhood rules that were deduced from the rules involved in 3CASC, by left-skewing the 3-neighborhood rule(i.e. by involving an additional variable). The main purpose of this contribution is to study 3CASC and 4CASC and compare their results referring to their statistical and cryptographic properties as well as their security analysis. The transition from 3 to 4 neighborhood increases some cryptographic properties like nonlinearity and algebraic degree, this insure better security against some attacks. From another perspective, the correlation immunity and resiliency provided by 3-neighborhood rulesets are better. So according to the security level required, the user could choose between them. The two last contributions aim to ensure data integrity. The fifth contribution, LCAHASH 1.1, is an enhancement of the LCAHASH design. It replaces the cellular automaton that used a 7 or 8-variable global function with a 128 or 256-variable global function using the rules 30 and 90 for 128 or 256 iterations to ensure higher security and statistical properties. The sixth contribution, HCAHF, is a new CA-based hash functions family, inspired by the wide-pipe Merkle Damgard construction, that goes through three fundamental steps, each one serves as a countermeasur to prevent attacks and to provide the required behaviour and properties of a secure hash function design. A specific version, HCAHF-256, outputs 256-bit digests providing satisfying characteristics.

Perspectives

This dissertation presents five cryptographic systems ensuring data confidentiality and integrity. Each of them could be extended for either better security or to enlarge the range of possible applications, the following perspectives could achieve these goals:

- The CA-PCS could be upgraded to define a new cellular automata based block cipher, with the introduction of CA-based Sboxes and P-box. In addition to the exploration of more CA configurations.
- A new CFA implementation for specific applications with constrained devices by means of different CA configuration.
- An extended version of NCASC could involve both 3 and 4-neighborhood rules to benefit from the advantages of both configurations.
- A Message Authencation Code based on CA-based block cipher.

List of Publications

1 **F. E. Ziani** and O. Fouzia, "*Partition ciphering system: A difficult problem based encryption scheme*," ***International Journal of Advanced Computer Science Applications***, Volume 10 Issue 11,pp. 286-291, 2019.

2 **F. E. Ziani**, A. Sadak, C. Hanin, B. Echandouri, and F. Omary, "*CA-PCS: A cellular automata based partition ciphering system*," ***International Journal of Advanced Computer Science Applications***Volume 11 Issue 3,pp. 603-609, 2020.

3 **F. E. Ziani**, A. Sadak, C. Hanin, B. Echandouri, and F. Omary, "*CASC 3N vs. 4N: Effect of increasing cellular automata neighborhood size on cryptographic strength*," ***International Journal of Advanced Computer Science Applications***, 2020.

4 B. Echandouri, F. Omary, F. **E. Ziani**, and A. Sadak, "*SEC-CMAC a new message authentication code based on the symmetrical evolutionist ciphering algorithm*," ***International Journal of Information Security and Privacy***, 2018.

5 A. Sadak, B. Echandouri, **F. E. Ziani**, C. Hanin, and F. Omary, "LCAHASH-1.1: A new design of the LCAHASH system for IoT," ***International Journal of Advanced Computer Science Applications***, 2019.

6 A. Sadak, **F. E. Ziani**, B. Echandouri, C. Hanin, and F. Omary, "*HCAHF: A new family of CA-based hash functions*," ***International Journal of Advanced Computer Science Applications***, 2019.

7 **F. E. Ziani**, A. Sadak, C. Hanin, B. Echandouri, and F. Omary, "*CFA: A New Family of Hybrid CA-Based PRNGs*," submitted.

Bibliography

[1] S. Wolfram, Random sequence generation by cellular automata, Advances in Applied Mathematics, vol. 7, no. 2, pp. 123-169, 1986

[2] M. Szaban and F. Seredynski, Designing conflict free cellular automata-based PRNG, J. Cell. Autom., 2018.

[3] Z. Zarezadeh, Cellular automaton-based pseudorandom number generator, Complex Syst., 2017.

[4] C. Hanin, F. Omary, S. Elbernoussi, and B. Boulahiat, "Design of new pseudo-random number generator based on non-uniform cellular automata," Int. J. Secur. its Appl., 2016, doi: 10.14257/ijsia.2016.10.11.10.

[5] M. Szaban, "Pseudorandom number generator based on totalistic cellular automaton," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2019.

[6] Damgård, I., "A Design Principle for Hash Functions," Advances in Cryptology ŋ Crypto 89, Lecture Notes in Computer Science, vol. 435, p. 416ŋ427, 1990

[7] Mihaljevic, M., Zheng, Y., Imai, H. A Family of Fast Dedicated One-Way Hash Functions Based on Linear Cellular Automata Over GF (q). IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 40-47, 1999.

[8] Jeon, J. , One-Way Hash Function Based on Cellular Automata. IT Convergence and Security 2012 Lecture Notes in Electrical Engineering, 21-28, 2012.

[9] Kuila, S., Saha, D., Pal, M., Chowdhury, D. R. (2014). CASH: Cellular Automata Based Parameterized Hash. Security, Privacy, and Applied Cryptography Engineering Lecture Notes in Computer Science, 59-75, 2014.

[10] C. Hanin, B. Echandouri, F. Omary, and S. E. Bernoussi, LCAHASH: A novel lightweight hash function based on cellular automata for RFID, Ubiquitous Networking Lecture Notes in Computer Science, pp. 287"298, 2017.

[11] C. Hanin, F. Omary, S. Elbernoussi, K. Achkoun, and B. Echandouri, A new block cipher system using cellular automata and ant colony optimization (BC-CaACO), Int. J. Inf. Secur. Priv., 2018.

[12] K. Achkoun, C. Hanin, and F. Omary, "SPF-CA: A new cellular automata based block cipher using key-dependent S-boxes," J. Discret. Math. Sci. Cryptogr., 2019.

[13] J. K. K. Jeyaprakash, J. G. Seka, and K. Villayutham, "KAMAR: A Lightweight Feistel Block Cipher Using Cellular Automata," Circuits Syst., 2016.

[14] P. Sarkar, "Hiji-bij-bij: A new stream cipher with a self-synchronizing mode of operation," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2003.

[15] S. Karmakar, D. Mukhopadhyay, and D. R. Chowdhury, "CAvium - Strengthening Trivium stream cipher using Cellular Automata," J. Cell. Autom., 2012.

[16] M. Mukherjee, N. Ganguly, and P. Pal Chaudhuri, "Cellular automata based authentication (CAA)," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2002.

[17] B. Echandouri, C. Hanin, F. Omary, and S. Elbernoussi, "LCAHASH-MAC: A new lightweight message authentication code using cellular automata for RFID," in Proceedings - 2017 International Conference on Wireless Networks and Mobile Communications, WINCOM 2017, 2017.

[18] A. Mouloudi, F. Omary, A. Tragha, and A. Bellaachia, "An Extension of Evolutionary Ciphering System," 2006 International Conference on Hybrid Information Technology, 2006.

[19] K. Chakraborty and D. R. Chowdhury, "CSHR: Selection of Cryptographically Suitable Hybrid Cellular Automata Rule," Lecture Notes in Computer Science Cellular Automata, pp. 591-600, 2012.

[20] M. Hell, T. Johansson, and W. Meier, "Grain: A stream cipher for constrained environments," Int. J. Wirel. Mob. Comput., 2007.

[21] P. C. van Oorschot, Computer Security and the Internet, Information Security and Cryptography,2020

[22] W. Stallings, Cryptography and network security: principles and practice. Pearson Prentice Hall, 2017.

[23] A. L. Rukhin, A statistical test suite for random and pseudorandom number generators for cryptographic applications. Gaithersburg, MD: U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, 2000

[24] Robert G. Brown’s General Tools Page. [Online]. Available: https://phy.duke.edu/ rgb/General/dieharder.php.

[25] M.O. Rayes, One-Time Password. In: van Tilborg H.C.A., Jajodia S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA. , 2011

[26] C. Fontaine, ’Synchronous Stream Cipher’. In: van Tilborg H.C.A., Jajodia S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA., 2011

[27] C. Fontaine, ’Self-Synchronizing Stream Cipher’. In: van Tilborg H.C.A., Jajodia S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA. , 2011

[28] C. E. Shannon, "Communication Theory of Secrecy Systems*," Bell System Technical Journal, vol. 28, no. 4, pp. 656"715, 1949.

[29] J. Daemen and V. Rijmen, "Specification of Rijndael," Information Security and Cryptography The Design of Rijndael, pp. 31"51, 2020.

[30] A. R. Bogdanov, L. J. B. Knudsen, G. undefined Leander, C. undefined Paar, A. undefined Poschmann, M. undefined Robshaw, Y. undefined Seurin, and C. undefined Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," Cryptographic Hardware and Embedded Systems - CHES 2007 Lecture Notes in Computer Science, pp. 450"466, 2007.

[31] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. Win, "The cipher SHARK," Fast Software Encryption Lecture Notes in Computer Science, pp. 99"111, 1996.

[32] A. Biryukov, Feistel Cipher. In: van Tilborg H.C.A., Jajodia S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA. , 2011

[33] – ,Data Encryption Standard (DES). United States: National inst of standards and technology gaithersburg md, 1999.

[34] C. De Cannière, "Triple DES," in Encyclopedia of Cryptography and Security, 2011.

[35] S. R., "The Blowfish Encryption Algorithm," Dr. Dobb"s J., 1994.

[36] F. H. Mathis, "A Generalized Birthday Problem," SIAM Rev., 1991, doi: 10.1137/1033051.

[37] H. Tiwari, "Merkle-Damgård construction method and alternatives: A review," J. Inf. Organ. Sci., 2017, doi: 10.31341/jios.41.2.9.

[38] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge Functions," ECRYPT hash Work., 2007.

[39] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The keccak reference," Submiss. to NIST (Round 3), 2011.

[40] R. L. Rivest and J. C. N. Schuldt, "Spritz " a spongy RC4-like stream cipher and hash function," CRYPTO 2014 Rump Sess., 2014.

[41] M. Blanton, Message Authentication Codes. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. , 2009

[42] J. Song, R. Poovendran, J. Lee, and I. T., "RFC 4493: The AES-CMAC Algorithm," Network Working Group, 2006.

[43] S. Turner and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms," RFC, 2011.

[44] D. W. Davies, "A Message Authenticator Algorithm Suitable for a Mainframe Computer," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 1985, .

[45] J.-P. Aumasson, Serious cryptography: a practical introduction to modern encryption. San Francisco: No Starch Press, 2018.

[46] K. Bhattacharjee, N. Naskar, S. Roy, and S. Das, "A survey of cellular automata: types, dynamics, non-uniformity and applications," Natural Computing. 2020.

[47] J. T. Schwartz, J. von Neumann, and A. W. Burks, "Theory of Self-Reproducing Automata," Math. Comput., 1967.

[48] D. Mukhopadhyay and A. Kundu, "Preliminaries on Cellular Automata," 2019.

[49] S. Wolfram, "Universality and complexity in cellular automata," Phys. D Nonlinear Phenom., 1984.

[50] T. W. Cusick and P. Stanica, Cryptographic Boolean Functions and Applications. 2009.

[51] F. Omary, A. Tragha, A. Bellachia, A. Lbekkouri, A. Mouloudi, "An Evolutionary Algorithm to Cryptography". Advances in Computational Methods in sciences and Engineering , Vol. 4, pp. 1749- 1752, 2005.

[52] D. R. Jones and M. A. Beltramo, " Solving Partitionning Problems with Genetic Algorithms," Proceedings of the Fourth International Conference on Genetic Algorithms, 1991

[53] E. Falkenauer, "Solving Equal Piles with the Grouping Genetic Algorithm,"Proceedings of the Sixth Intenational Conference on Genetic Algorithms,1995.

[54] W. A. Greene, "Genetic Algorithms For Partitioning Sets," International Journal on Artificial Intelligence Tools, vol. 10, no. 01n02, pp. 225-241, 2001.

[55] E. Falkenauer, "Applying Genetic Algorithms to Real-World Problems," Evolutionary Algorithms The IMA Volumes in Mathematics and its Applications, pp. 65-88, 1999.

[56] F. Omary, A. Mouloudi, A. Tragha, and A. Bellaachia, "A New Ciphering Method Associated with Evolutionary Algorithm," Computational Science and Its Applications - ICCSA 2006 Lecture Notes in Computer Science, pp. 346-354, 2006.

[57] F. Omary, A. Tragha, A. Bellaachia, and A. Mouloudi, "Design and Evaluation of Two Symmetrical Evolutionist-Based Ciphering Algorithms," International Journal of Computer Science and Network Security, vol. 7, no. 2, pp. 181-190, 2007.

[58] S. Trichni, F. Omary, B. Boulahiat, and M. Bougrine, "A new approach of mutation's operator applied to the ciphering system SEC," 6th ICCIT: International Conference on Computer Sciences and Convergence Information Technology (ICCIT 2011), 2011.

[59] M. Bougrine, F. Omary, S. Trichni, and B. Boulahiat, "New evolutionary tools for a new ciphering system SEC version," 2012 IEEE International Carnahan Conference on Security Technology (ICCST), 2012.

[60] Z. Kaddouri, F. Omary, A. Abouchouar , and M. Daari, "Balancing Process to the Ciphering System SEC," Journal of Theoretical and Applied Information Technology, 2013.

[61] Z. Kaddouri, F. Omary, and A. Abouchouar, "Binary Fusion Process to The Ciphering System SEC Extension To Binary Blocks," Journal of Theoretical and Applied Information Technology, 48(1), 2013.

[62] A. Mouloudi,"New Symmetric Encryption System Based on Evolutionary Algorithm,? International Journal of Computer Science and Information Technology, vol. 7, no. 6, pp. 39-49, 2015.

[63] C. E. Shannon, "A Mathematical Theory of Communication," Bell System Technical Journal, vol. 27, no. 4, pp. 623 - 656, 1948.

[64] S. Wolfram, "Cryptography with Cellular Automata," Lecture Notes in Computer Science Advances in Cryptology " CRYPTO "85 Proceedings, pp. 429-432.

[65] A. Ray and D. Das, "Encryption Algorithm for Block Ciphers Based on Programmable Cellular Automata," Communications in Computer and Information Science Information Processing and Management, pp. 269-275, 2010.

[66] J. Bhaumik and D. R. Chowdhury, "Design and implementation of Cellular Automata based diffusion layer for SPN-type block cipher," International Conference on Informatics, Electronics and Vision (ICIEV), 2012.

[67] S. Roy, S. Nandi, J. Dansana, and P. K. Pattnaik, "Application of cellular automata in symmetric key cryptography," International Conference on Communication and Signal Processing, 2014.

[68] R. K. Mehta and R. Rani, "Pattern generation and symmetric key block ciphering using cellular automata," International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2016.

[69] S. Bouchkaren and S. Lazaar, "A New Cryptographic Scheme Based on Cellular Automata," Lecture Notes in Electrical Engineering Proceedings of the Mediterranean Conference on Information and Communication Technologies 2015, pp. 663 - 668, 2016.

[70] S. Bouchkaren and S. Lazaar, "A fast cryptosystem using reversible cellular automata," International Journal of Advanced Computer Science and Applications, vol. 5, no. 5, 2014.

[71] K. M. Faraoun, "A genetic strategy to design cellular automata based block ciphers," Expert Systems with Applications, vol. 41, no. 17, pp. 7958 - 7967, 2014.

[72] K. Li, M. Sun, L. Li, and J. Chen, "Image Encryption Algorithms Based on Non-uniform Second- Order Reversible Cellular Automata with Balanced Rules," Intelligent Computing Theories and Application Lecture Notes in Computer Science, pp. 445 - 455, 2017.

[73] A. Y. Niyat, M. H. Moattar, and M. N. Torshiz, "Color image encryption based on hybrid hyper-chaotic system and cellular automata," Optics and Lasers in Engineering, vol. 90, pp. 225 - 237, 2017.

[74] A. Biryukov and C. Cannière, "Linear Cryptanalysis for Block Ciphers," Encyclopedia of Cryptography and Security, pp. 351 - 354, 2011.

[75] E. Biham, "Differential Cryptanalysis," Encyclopedia of Cryptography and Security, pp. 147 - 152.2011

[76] S. Wolfram, Origins of Randomness in Physical Systems. Physical Review Letters, 55(5), 449.1985

[77] 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

[78] 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

[79] Tomassini, M., Sipper, M., Zolla, M., and Perrenoud, M. (1999). Generating High-Quality Random Numbers in Parallel by Cellular Automata. Future Generation Computer Systems, 16(2-3), 291-305.

[80] Guan, S. U., and Zhang, S. (2003). An Evolutionary Approach to The Design of Controllable Cellular Automata Structure for Random Number Generation. IEEE Transactions on Evolutionary Computation, 7(1), 23-36.

[81] Guan, S. U., and Tan, S. K. (2004). Pseudorandom Number Generation With Self-Programmable Cellular Automata. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23(7), 1095-1101.

[82] Seredynski, F., Bouvry, P., and Zomaya, A. Y. (2004). Cellular Automata Computations and Secret Key Cryptography. Parallel Computing, 30(5-6), 753-766.

[83] Bhattacharjee, K., Paul, D., and Das, S. (2017). Pseudo-Random Number Generation Using a 3-state Cellular Automaton. International Journal of Modern Physics C, 28(06), 1750078.

[84] Kelsey, J., Schneier, B., Wagner, D., and Hall, C. (1998). Cryptanalytic Attacks on Pseudorandom Number Generators. International Workshop on Fast Software Encryption, 168-188.

[85] Kelsey, J., Schneier, B., and Ferguson, N. (2000). Yarrow-160: Notes on The Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. Selected Areas in Cryptography Lecture Notes in Computer Science, 13-33.

[86] Gutmann, P. (1998). Software Generation of Practically Strong Random Numbers. Usenix Security Symposium.

[87] Feistel, H. (1973). Cryptography and Computer Privacy. Scientific American, 228(5), 15-23.

[88] C. D. Cannière, "Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles," Lecture Notes in Computer Science Information Security, pp. 171–186, 2006

[89] S. Karmakar and D. R. Chowdhury, "NOCAS : A Nonlinear Cellular Automata Based Stream Cipher," Discrete Mathematics and Theoretical Computer Science, pp. 135–146, 2012.

[90] S. Das and D. R. Chowdhury, "CASTREAM: A New Stream Cipher Suitable for Both Hardware and Software," Lecture Notes in Computer Science Cellular Automata, pp. 601–610, 2012.

[91] S. Das and D. Roychowdhury, "CAR30: A new scalable stream cipher with rule 30," Cryptography and Communications, vol. 5, no. 2, pp. 137–162, Jul. 2013.

[92] S. Ghosh and D. R. Chowdhury, "CASca:A CA Based Scalable Stream Cipher," Mathematics and Computing Springer Proceedings in Mathematics Statistics, pp. 95–105, 2015.

[93] J. Bhaumik and D. R. Chowdhury, "Nmix: An Ideal Candidate For Key Mixing," Proceedings of the International Conference on Security and Cryptography, 2009.

[94] J. Jose and D. R. Chowdhury, "FResCA: A Fault-Resistant Cellular Automata Based Stream Cipher," Lecture Notes in Computer Science Cellular Automata, pp. 24–33, 2016.

[95] J. Jose and D. R. Chowdhury, "Investigating four neighbourhood cellular automata as better cryptographic primitives," Journal of Discrete Mathematical Sciences and Cryptography, vol. 20, no. 8, pp. 1675–1695, 2017.

[96] R. Lakra, A. John, and J. Jose, "CARPenter: A Cellular Automata Based Resilient Pentavalent Stream Cipher," Developments in Language Theory Lecture Notes in Computer Science, pp. 352–363, 2018.

[97] M. Perrenoud, M. Sipper, and M. Tomassini, "On the generation of high-quality random numbers by two-dimensional cellular automata," IEEE Transactions on Computers, vol. 49, no. 10, pp. 1146–1151, 2000.

[98] S. Karmakar, D. Mukhopadhyay, and D. R. Chowdhury, "d-Monomial Tests of Nonlinear Cellular Automata for Cryptographic Design," Lecture Notes in Computer Science Cellular Automata, pp. 261–270, 2010.

[99] S. Maiti, S. Ghosh, and D. R. Chowdhury, "On the Security of Designing a Cellular Automata Based Stream Cipher," Information Security and Privacy Lecture Notes in Computer Science, pp. 406–413, 2017.

[100] M. U. Bokhari, S. Alam, and F. S. Masoodi, "Cryptanalysis Techniques for Stream Cipher: A Survey," International Journal of Computer Applications, vol. 60, no. 9, pp. 29–33, 2012.

[101] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," Advances in Cryptology — EUROCRYPT '93 Lecture Notes in Computer Science, pp. 386–397, 1994.

[102] T. Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphertext Only," IEEE Transactions on Computers, vol. C-34, no. 1, pp. 81–85, 1985.

[103] I. B. Damgård, "A design principle for hash functions," Advances in Cryptology ? CRYPTO? 89 Proceedings Lecture Notes in Computer Science, pp. 416?427, 1989.

[104] J. Daemen, R. Govaerts, and J. Vandewalle, "A framework for the design of one-way hash functions including cryptanalysis of Damgård?s

one-way function based on a cellular automaton," Advances in Cryptology - ASIACRYPT 91 Lecture Notes in Computer Science, pp. 82?96, 1991.

[105] J. Daemen, R. Govaerts, and J. Vandewalle, "A hardware design model for cryptographic algorithms," Computer Security ? ESORICS 92 Lecture Notes in Computer Science, pp. 419?434, 1992.

[106] D. Chang, "Preimage attacks on CellHash, SubHash and strengthened versions of CellHash and SubHash," IACR Cryptology ePrint Archive 2006: 412, 2006.

[107] M. Mihaljeviç, Y. Zheng, and H. Imai, "A cellular automaton based fast one-way hash function suitable for hardware implementation," Public Key Cryptography Lecture Notes in Computer Science, pp. 217?233,1998.

[108] A. Bogdanov, M. Kneÿevi?, G. Leander, D. Toz, K. Var?c?, and I. Verbauwhede, "SPONGENT: A lightweight hash function," Cryptographic Hardware and Embedded Systems ? CHES 2011 Lecture Notes in Computer Science, pp. 312?325, 2011.

[109] T. P. Berger, J. D?Hayer, K. Marquet, M. Minier, and G. Thomas, "The GLUON Family: A lightweight hash function family based on FCSRs," Progress in Cryptology - AFRICACRYPT 2012 Lecture Notes in Computer Science, pp. 306?323, 2012.

[110] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "Quark: A lightweight hash," Cryptographic Hardware and Embedded Systems, CHES 2010 Lecture Notes in Computer Science, pp. 1?15, 2010.

[111] J. Guo, T. Peyrin, and A. Poschmann, "The PHOTON family of lightweight hash functions," Advances in Cryptology - CRYPTO 2011 Lecture Notes in Computer Science, pp. 222?239, 2011.

[112] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "The making of KECCAK," Cryptologia, vol. 38, no. 1, pp. 26?60, Feb. 2014.

[113] E. Biham and O. Dunkelman, "A framework for iterative hash functions: HAIFA," IACR Cryptology ePrint Archive 2007: 278, 2007.

[114] S. Lucks, "A failure-friendly design principle for hash functions," Lecture Notes in Computer Science Advances in Cryptology - ASIACRYPT 2005, pp. 474?494, 2005.

[115] A. Joux, "Multicollisions in iterated hash functions. application to cascaded constructions," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2004, doi: 10.1007/978-3-540-28628-8_19.

[116] D. Gligoroski, "Length Extension Attack on Narrow-Pipe SHA-3 Candidates," Communications in Computer and Information Science ICT Innovations 2010, pp. 5?10, 2011.

[117] R. D. Dean, "Formal aspects of mobile code security," (Unpublished doctoral dissertation). Princeton University, Princeton, NJ., 1999.

[118] J. Kelsey and B. Schneier, "Second preimages on n-bit hash functions for much less than 2 n work," Lecture Notes in Computer Science Advances in Cryptology - EUROCRYPT 2005, pp. 474?-490, 2005.

[119] J. Kelsey and T. Kohno, "Herding hash functions and the nostradamus attack," Advances in Cryptology - EUROCRYPT 2006 Lecture Notes in Computer Science, pp. 183?200, 2006.

[120] M. Wang, M. Duan, and J. Zhu, "Research on the security criteria of hash functions in the blockchain," Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts - BCC 18, 2018.

[121] S. Wolfram, A new kind of science. USA: Wolfram Media, 2002.

[122] C. Carlet, "Boolean functions for cryptography and error-correcting codes," Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 257?397, 2010.

Appendices

Appendix A

Rules Selection for the CA-PCS hybrid CA

The ruleset selection mechanism adopted to form the CA-PCS cellular automaton is provided by[19]. Given the set of linear and nonlinear rules S found in [19], split S into two sets, a linear set S_L and a nonlinear one S_{NL}. $S_L = \{60, 90, 102, 105, 150, 165, 195\}$ and $S_{NL} = \{22, 30, 37, 41, 43, 45, 91, 110, 120, 135, 180, 210\}$. Tables A.2 and A.1 display the cryptographic properties of S_L and S_{NL} respectively. Given S_L and S_NL, to construct the

Table A.1: The cryptographic properties of S_{NL}

Rule	AD	NL	Bal	CI
22	7	45	×	1
30	5	40	✓	1
37	7	49	×	0
41	7	44	×	1
43	5	44	✓	0
45	4	40	✓	1
91	7	49	×	0
110	6	38	×	1
120	5	48	✓	0
135	5	48	✓	2
180	4	32	✓	1
210	4	32	✓	0

desired cellular automaton, the following guidelines of [19] were followed:

1. Pick a nonlinear rule with good cryptographic properties : $\{30\}$

Table A.2: The cryptographic properties of S_L

Rule	AD	NL	Bal	CI
60	1	0	✓	3
90	1	0	✓	3
102	1	0	✓	3
105	1	0	✓	4
150	1	0	✓	4
165	1	0	✓	3
195	1	0	✓	3

2. Chose more rules to have similar number of linear and nonlinear rules in the rest of cells to find a compromise between algebraic degree and correlation immunity: {45,90,105,180}

3. To increase the algebraic degree, two or three nonlinear rules should appear consecutively in the ruleset : {30,90,45,180,105}

4. To make correlation immunity better, more than one linear rule should follow a nonlinear rule: {30,90,105,30,180,45,90,105}

5. To make the period higher, pick rules having large period :{90,150} $\rightarrow$ {30,90,150,30,180,45,90,150}

The resulting rulset to use in CA-PCS having satisfying cryptographic properties is

$$R_{CA-PCS} = \{30, 90, 150, 30, 180, 45, 90, 150\}$$

Appendix B

Rules Selection for the CFA hybrid CA

The same process of A is followed to get the ruleset of CFA.

1. Pick a nonlinear rule with good cryptographic properties : {30}
2. Chose more rules to have similar number of linear and nonlinear rules in the rest of cells to find a compromise between algebraic degree and correlation immunity: {30,60, 110,90}
3. To increase the algebraic degree, two or three nonlinear rules should appear consecutively in the ruleset : {30,60,30,110,90}
4. To increase the algebraic degree, two or three nonlinear rules should appear consecutively in the ruleset : {30,60,90,30,110,30,60,90}
5. To make the period higher, pick rules having large period :{90,150} $\rightarrow$ {30,90,150,30,110,30,90,150}

The resulting rulset to use in CFA having satisfying cryptographic properties is

$$R_{CFA} = \{30, 90, 150, 30, 110, 30, 90, 150\}$$

Appendix C

4-Neighborhood Rules using 3-Neighborhood rules (NCASC)

This appendix presents shows the rulesets involved in 3CASC and 4CASC for the nonlinear CA, the same concept work for the other CAs.

C.1 NHCA rules

C.1.1 3-Neighborhood rules

Rule 30: $x_i^{t+1} = x_{i-1}^t \oplus (x_i^t + x_{i+1}^t)$
Rule 120: $x_i^{t+1} = x_{i-1}^t \oplus (x_i^t \cdot x_{i+1}^t)$
Rule 180: $x_i^{t+1} = x_{i-1}^t \oplus (x_i^t \cdot \overline{x_{i+1}^t})$
Rule 45: $x_i^{t+1} = x_{i-1}^t \oplus (x_i^t + \overline{x_{i+1}^t})$

C.1.2 4-Neighborhood rules

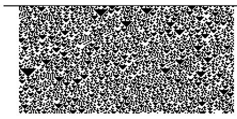
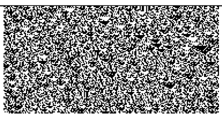
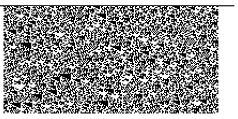
30-Like rules

Like-30 rules found by Jimmy Jose are presented in [95], we selected the best 4-neighborhood rules according to cryptographic properties in addition to space time diagram. 42390, 43350, and 51510 are the best ones (see[95]). Tables C.1 and C.2 provide the cryptographic properties and the space-time diagrams of the 4N rules.

Table C.1: The cryptographic properties of the best 30-like rules

Rule	NL	Bal	CI
42390	4	✓	1
43350	4	✓	1
51510	4	✓	1

Table C.2: The space time diagram of the best 30-like rules

Rule 42390	Rule 43350	Rule 51510
		

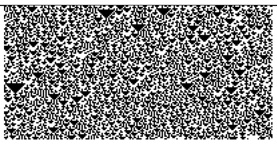
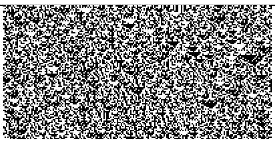
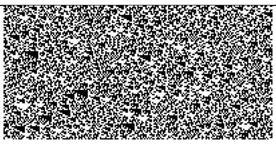
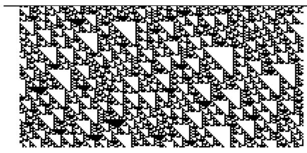
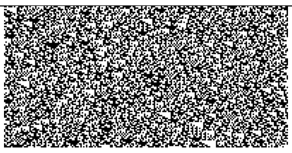
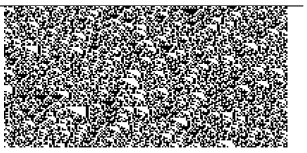
120-Like rules

120 like rules 4N :

Table C.3: Cryptograhic proerties of the best 120-like rules

Rules	NL	CI	RES	AD	Bal
38502	4	1	1	2	✓
38490	4	1	1	2	✓
38250	4	1	1	2	✓
38460	4	1	1	2	✓
37740	4	1	1	2	✓
34680	4	1	1	2	✓

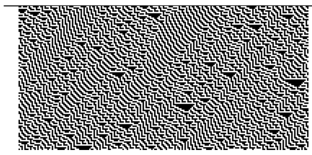
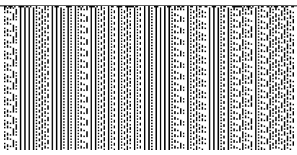
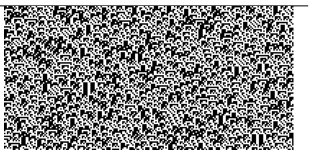
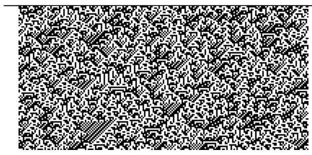
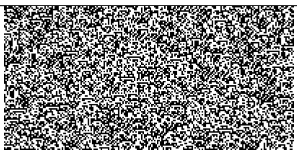
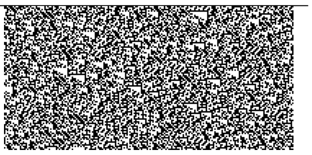
Table C.4: The space time diagram of the best 120-like rules

Rule 38502	Rule 38490	Rule 38250
		
Rule 38460	**Rule 37740**	**Rule 34680**
		

180-Like rules

Rules	NL	CI	RES	AD	Bal
38229	4	1	1	2	✓
8580	4	1	1	2	✓
27033	4	1	1	2	✓
27045	4	1	1	2	✓
27285	4	1	1	2	✓
25500	4	1	1	2	✓

Table C.5: The space time diagram of the best 180-like rules

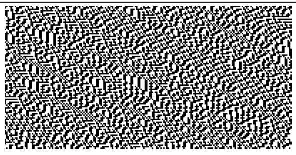
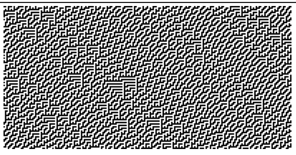
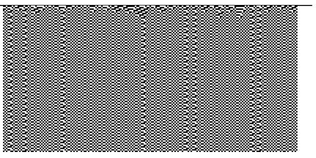
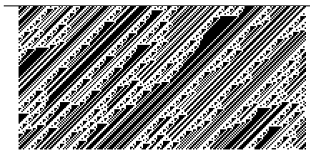
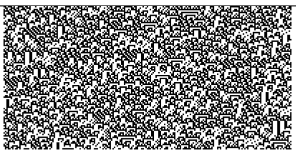
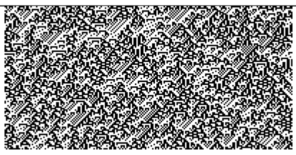
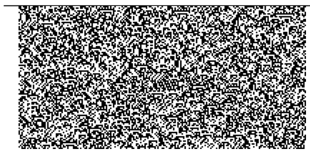
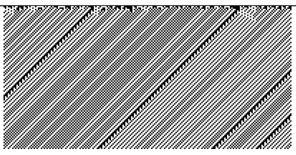
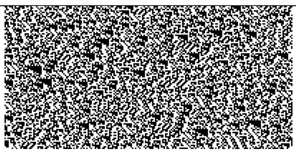
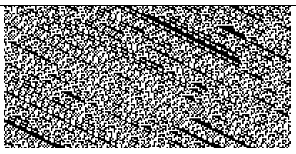
Rule 38229	Rule 8580	Rule 27033
		
Rule 27045	Rule 27285	Rule 25500
		

45-Like rules

Table C.6: Cryptographic properties of the best 45-like rules

Rule	NL	CI	RES	AD	Bal
765	2	0	0	3	✓
3885	2	0	0	3	✓
13113	2	0	0	3	✓
43689	2	0	0	3	✓
26217	4	1	1	2	✓
23145	4	1	1	2	✓
22185	4	1	1	2	✓
50025	4	1	1	2	✓
50745	4	1	1	2	✓
53805	4	1	1	2	✓

Table C.7: The space time diagram of the best 45-like rules

Rule 765	Rule 3885	Rule 13113
		
Rule 43689	Rule 26217	Rule 23145
		
Rule 22185	Rule 50025	Rule 50745
		
	Rule 53805	
		

Appendix D

Rules Selection for the HCAHF hybrid CA R_T

The same process of A is followed to get the ruleset of HCAHF.

1. Pick a nonlinear rule with good cryptographic properties : $\{30\}$
2. Chose more rules to have similar number of linear and nonlinear rules in the rest of cells to find a compromise between algebraic degree and correlation immunity: $\{30,60, 135,90\}$
3. To increase the algebraic degree, two or three nonlinear rules should appear consecutively in the ruleset : $\{30,60,30,135,90\}$
4. To increase the algebraic degree, two or three nonlinear rules should appear consecutively in the ruleset : $\{30,60,90,30,135,30,60,90\}$
5. To make the period higher, pick rules having large period :$\{90,150\}$ $\rightarrow$ $\{30,90,150,30,135,30,90,150\}$

The resulting rulset to use in CFA having satisfying cryptographic properties is

$$R_T = \{30, 90, 150, 30, 135, 30, 90, 150\}$$

ROYAUME DU MAROC
Université Mohammed V جامعة محمد الخامس
- RABAT - – الرباط –
Faculté des sciences
كلية العلوم
CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

Résumé

Le recours de systèmes et de protocoles cryptographiques est devenue indispensable pour garantir les objectifs de la sécurité informatique, qui correspondent à la motivation de cette thèse. De nouveaux systèmes cryptographiques symétriques sont conçus pour assurer la confidentialité et l'intégrité des données en utilisant les automates cellulaires, qui fournissent un comportement complexe et des propriétés satisfaisantes à partir de calculs rapides, simples et en parallèle. Dans un premier temps, un système de chiffrement symétrique inspiré par le problème de la partition, chiffrement par partition (PCS), est proposé pour rendre la cryptanalyse des fréquences et les attaques par force brute plus complexes. Ensuite, la deuxième contribution, CA-PCS utilise un automate cellulaire non-uniform produisant plus de confusion et de diffusion, ainsi qu'une meilleure sécurité contre les attaques. La troisième contribution vise à produire des séquences de haute qualité d'aléatoire grâce à une famille de générateurs de nombres pseudo-aléatoires, CFA, basée sur des automates cellulaires et d'autres primitives cryptographiques. La quatrième contribution est un chiffrement par flux impliquant des automates cellulaires, avec l'objectif d'étudier plusieurs configurations pour atteindre un meilleur niveau de sécurité. Les deux dernières contributions concernent les primitives assurant l'intégrité des données, à savoir les fonctions de hachage. La cinquième contribution, LCAHASH1.1, est une famille de fonctions de hachage basées sur des automates cellulaires. La dernière fait appel à plusieurs automates cellulaires en plus de certaines mécanismes pour prévenir les attaques connues.

Mots-clefs (11) : Sécurité Informatique, Cryptographie Symètrique, Automate Cellulaire, Confidentialité, Integrité, Effet d'Avalanche, Confusion, Diffusion, Générateur Pseudo-Aléatoire, Chiffrement par flux, Fonction de Hachage.

Abstract

The use of cryptographic systems and protocols became indispensable to guarantee the information security goals, which fall in the purpose of this thesis. New symmetric cryptographic systems are designed to ensure data confidentiality and integrity using cellular automata, which provide complex behavior and good properties from fast, simple, and parallel computations. At first a symmetric partition problem inspired encryption scheme, Partition Ciphering System (PCS), is proposed to make frequency cryptanalysis and brute force attacks challenging. Then the second contribution introduces a hybrid cellular automaton producing more confusion and diffusion, as well as better security against attacks. The third contribution aims to produce sequences with high quality of randomness through a family of pseudo-random number generators, CFA, based on cellular automata and other cryptographic primitives. The fourth contribution is a stream cipher involving cellular automata, with the objective of investigating different configurations to have better security level. The last two contributions fall in the primitives providing data integrity, namely hash functions. The fifth contribution, LCAHASH1.1, is a cellular automaton-based hash functions family. The last one makes use of multiple cellular automata in addition to some features to prevent known attacks.

Key Words (11) : Information security, Symmetric Cryptography, Cellular Automata, Confidentiality, Integrity, Avalanche effect, Confusion, Diffusion, Pseudorandom Number Generator, Stream Cipher, Hash functions