UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Implementing and Evaluating Nonsingular Matrices Generators for the Hill Cipher
(Versão Final Após a Defesa Pública)

Moisés Alfredo da Cunha Ferreira

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Ricardo Morais Inácio
Co-orientadora: Profª Doutora Aleksandra Mileva

Covilhã, Dezembro de 2018

Dedications

To my family, specially my mother, who gave me everything in this life, the main responsible for all I achieve; and my father, who passed away too soon, to whom I cannot show how I am following all of his advice.

Acknowledgments

First of all, I thank God for everything He has done in my life. Secondly, I would like to thank all my teachers and colleagues in the master's course, without forgetting my dear supervisor, Professor Pedro Inácio, for making my master's degree a success. I would also like to thank the co-advisor for this project, Professor Aleksandra Mileva from Macedonia, and the doctoral student Bernardo Sequeiros, for all the support in all of the process, from the beginning until the end. I cannot forget Professor Mário Freire, who made some very important corrections as well. Finally, I thank my graduation faculty, ISP/UKB, for the scholarship provided by INAGBE.

Resumo

A Cifra de Hill (Hill Cipher) é um exemplo clássico de um sistema criptográfico com propriedades muito interessantes, nomeadamente a implementação dos conceitos de confusão e difusão apresentados por Shannon como propriedades essenciais para as cifras; no entanto, a sua forma básica é vulnerável a Known Plaintext Attacks (KPAs). Esta dissertação apresenta um método eficiente para gerar matrizes não singulares, baseado no método de eliminação de Gauss-Jordan, que permite gerar uma nova matriz para cada bloco a ser cifrado. A cifra Randomized Key Hill Cipher (RKHC), descrita nesta dissertação, usa o referido método e adiciona um passo, quer ao algorítmo de cifra, quer ao algoritmo de decifra para tratar mensagens que contenham padrões (ex: uma sequência de zeros), a fim de aumentar a segurança contra KPAs, Chosen Plaintext Attacks (CPAs) e Chosen Ciphertext Attacks (CCAs). Uma avaliação da performance, mediante uma implementação não optimizada do algorítmo RKHC na linguagem ANSI C, é incluída e comparada com implementações optimizadas do Advanced Encryption Standard (AES) e do Salsa20, juntamente com uma discussão acerca da segurança e das limitações em relação aos ataques de criptanálise mais relevantes. A prova de que o método proposto possui aleatoriedade com elevada entropia nas matrizes e criptogramas gerados é apresentada mediante resultados provenientes de testes estatísticos da biblioteca TestU01 para a avaliação rigorosa da aleatoriendade dos *outputs* gerados.

Palavras-chave

Cifra de Hill, chave aleatória, Gauss-Jordan, matrizes não singulares, cifra simétrica, cifra por blocos.

Resumo alargado

Este resumo alargado tem como objetivos apresentar, na Língua Portuguesa, o conteúdo desta dissertação mais detalhadamente em relação à seccão anterior, pois a maior parte do conteúdo desta dissertação está escrito na Língua Inglesa.

Introdução

O primeiro capítulo tem como objetivo enquadrar o trabalho descrito ao longo desta dissertação, introduzindo o tema geral e apresentando a motivação, bem como o enquadramento relativo ao mesmo. Neste capítulo também é apresentado o problema que este trabalho se propõe a resolver e os seus objetivos. É ainda apresentada a abordagem seguida para resolver o problema proposto e as principais contribuições do trabalho.

Enquadramento, Descrição do Problema e Objetivos

A Cifra de Hill - Hill Cipher (HC) é um exemplo clássico de um sistema criptográfico com propriedades interessantes, nomeadamente a implementação dos princípios de difusão e confusão, defendidos por Shannon como propriedades fundamentais para as cifras; a HC tem um grande nível de entropia, obtida mediante a multiplicação de cada texto limpo (bloco) por uma matriz invertível. A HC tem atraído a atenção de muitos investigadores nos últimos anos, com o intuito de corrigir as suas limitações que ainda não foram conseguidas até ao momento, nomeadamente a vulnerabilidade a KPA, pois, quando uma mesma matriz é usada para cifrar vários blocos de texto, entre outras limitações, tais como CPA e CCA.

A motivação deste trabalho provém do estudo da Cifra de Hill (HC) clássica, focando na sua componente principal, a matriz que é usada como chave da cifra. Esta matriz deve ser não singular para que possa ser invertida e usada no momento da decifra; a geração de tais matrizes de forma pseudo-aleatória tem sido alvo da atenção da comunidade investigativa. Portanto, este trabalho descreve um novo método para gerar tais matrizes e o analisa no âmbito de uma nova variante da HC. Neste caso, a HC é modificada, a fim de melhorar a sua segurança.

O principal problema que este trabalho se propõe resolver é a vulnerabilidade da HC em relação a KPA, pois, quando vários textos são cifrados com a mesma matriz chave, torna fácil a recuperação da mesma por meio da resolução de um sistema de equações lineares trivial. Outro problema que este trabalho pretende resolver é o facto de, até ao momento, não haverem métodos eficientes na literatura para gerar matrizes não singulares que possam ser usadas como chave de cifra para cada bloco de texto. Além da vulnerabilidade em relação a KPA, a HC também é vulnerável a CPA e CCA (ex: para uma entrada de blocos formados inteiramente por zeros, a HC sempre dá como saída o mesmo bloco de zeros).

O principal objetivo desta dissertação é, portanto, desenvolver uma variante segura da HC. Para alcançar este objetivo é necessário: a) apresentar um método eficiente para gerar matrizes não singulares pseudo-aleatórias; b) modificar o algorítmo de cifra e o de decifra da HC, a fim de corrigir as vulnerabilidades contra KPA, CPA e CCA; c) avaliar as referidas modificações e o método proposto (aqui denominado por RKHC).

A abordagem escolhida para resolver o referido problema começou com o estudo das modifi-

cações da HC encontradas na literatura. Essas modificações foram importantes para perceber o que já foi feito para resolver as vulnerabilidades em relação a KPA, CPA e CCA. Em seguida, como nenhuma dessas modificações apresenta um método eficiente para gerar matrizes não singulares pseudo-aleatórias, um método baseado na eliminação de Gauss-Jordan foi proposto, implementado e analisado. Este método é capaz de gerar todas as matrizes necessárias, ou seja, para uma determinada ordem n, o método permite gerar todas as matrizes não singulares contendo apenas 0s e 1s. O passo a seguir consistiu em analisar como as debilidades do HC, em relação a textos-limpos ou criptogramas específicos (tal como vetores inteiramente formados por zeros), podem ser revistas para a proposta das modificações que resolvem tais problemas, fazendo com que o algorítmo seja resistente a esses ataques. Para testar o método proposto, uma implementação não otimizada do RKHC na linguagem ANSI C foi levada a cabo e comparada com implementações otimizadas do AES e do Salsa20 em termos de benchmarking. A famosa biblioteca TestU01, para avaliação rigorosa de aleatoriedade em testes estatísticos, foi usada para avaliar o algorítmo proposto para geração de matrizes não-singulares, a fim de assegurar que os outputs têm entropia suficiente para ser usados em cenários criptográficos.

Principais Contribuições

As principais contribuições alcançadas ao longo da investigação e desenvolvimento deste projeto podem ser apresentadas da seguinte forma:

1. Um método eficiente para gerar uma matrix não singular dada uma determinada seed (semente), implementado na linguagem ANSI C e avaliado. A novidade deste método é que este gera todas as matrizes não singulares possíveis para qualquer ordem n, de forma eficiente e com um vasto espaço de chaves;
2. Um algorítmo eficiente para gerar diretamente uma matriz inversa, baseado especificamente no algorítmo referido na primeira contribuição, foi também proposto, implementado e avaliado;
3. A dissertação também descreve uma modificação da HC (RKHC), nomeadamente os algorítmos de cifra e de decifra, utilizando o método referido acima para cifrar/decifrar cada bloco de texto limpo/criptograma com uma matriz pseudo-aleatória diferente, com meios adicionais de prevenir ataques de KPA, CPA e CCA;
4. Também foi parte deste trabalho uma criptanálise inicial em relação a HC e a RKHC, utilizando vários modelos e ataque.

Fundamentação Teórica e Trabalhos Relacionados

O capítulo 2 descreve os conceitos relacionados com este trabalho: os fundamentos da criptografia, as cifras simétrica e assimétrica (de chave pública), os PRNG e CSPRNG, o conceito de testes estatísticos e a famosa biblioteca TestU01. Vários modelos de ataque são apresentados, uma vez que são ferramentas estudadas profundamente ao longo do trabalho para provar a segurança do sistema criptográfico apresentado.

Finalmente, é apresentada a HC clássica, juntamente com os seus pontos fortes e fracos, além disso, são apresentados vários trabalhos relacionados para melhorar a segurança da HC, tais trabalhos incluem múltiplas multiplicações de matrizes, geração de matrizes por tentativa e erro, uso de matrizes auto-invertíveis (matrizes iguais às suas respetivas inversas), entre outras tentativas.

Randomized Key Hill Cipher (RKHC)

Os capítulos 3 e 4 descrevem o sistema criptográfico proposto (RKHC) e sua avaliação.

No capítulo 3 é apresentada a principal contribuição desta dissertação que consiste no referido sistema criptográfico, uma variante segura do HC, RKHC, explorando o uso de CSPRNG para gerar uma matriz chave diferente para cada bloco submetido à cifra/decifra. O algorítmo proposto para gerar matrizes não singulares módulo 2 é baseado no método de eliminação de Gauss-Jordan, um algorítmo simples, mas não foi encontrado algum que use esta abordagem, muito menos um método eficiente que cumpra tal tarefa. Neste capítulo são ultrapassados os problemas encontrados nos outros trabalhos relacionados apresentados no capítulo anterior, pois estes apresentam três problemas fundamentais, nomeadamente: (i) a suscetibilidade à KPA; (ii) o tamanho e a transmissão da chave; (iii) a predição dos outputs gerados para blocos formados unicamente por zeros (são gerados zeros igualmente). Para tratar dos dois primeiros problemas, são apresentados algorítmos para gerar, de forma eficiente, matrizes pseudo-aleatórias módulo 2 e suas respetivas inversas para cada bloco. Para resolver o terceiro problema, um passo adicional é incluído nos procedimentos de cifra e decifra. Esses procedimentos são descritos detalhadamente. Aqui é assumido que o número de letras do alfabeto usado é igual a 2 ($n = 2$). Isto significa que as operações aritméticas são feitas módulo 2. Muitas vezes, *XOR* ($\oplus$) é usado para referir a soma módulo 2 e cada letra da mensagem é referida como *bit*. A cifra RKHC proposta é eficiente e segura em relação a outras variantes da HC. A resistência da RKHC contra os ataques básicos também foi analisada e o algorítmo mostra-se resistente a cada um deles com as modificações apresentadas.

Quanto ao capítulo 4, é reservado à uma análise prática de uma implementação não otimizada da RKHC em ANSI C, comparada com implementações de algorítmos conceituados, nomeadamente o AES e o Salsa20. Além do benchmarking, os testes também incluem avaliação da aleatoriedade das matrizes geradas pelo método associado ao RKHC, mediante a biblioteca TestU01, sendo que, a RKHC passou em todos os testes de aleatoriedade a que foi submetida e mostrou-se muito melhor que o único algorítmo que encontramos na literatura para gerar matrizes não singulares (método desenvolvido por [Ran93]).

Conclusões

O capítulo final desta dissertação apresenta as conclusões e o trabalho futuro que pode ser levado a cabo como continuidade e melhoramento deste trabalho.

Aqui foi salientado que o foco deste trabalho foi a pesquisa e desenvolvimento de uma variante do HC denominada RKHC com melhor segurança em relação à cifra original, bem como em relação às variantes apresentadas no capítulo 2. Foi frisado que o trabalho apresenta métodos eficientes e seguros para cifrar e decifrar, utilizando matrizes pseudo-aleatórias geradas através de algorítmos eficientes e criptograficamente seguros. O método (sistema criptográfico) proposto também apresenta resistência contra modelos de ataque como KPA, CPA e CCA; isto foi demonstrado para os ataques básicos e brevemente discutido em relação aos ataques mais poderosos, nomeadamente a criptanálise diferencial e a linear. Apesar da versão proposta estar concebida de forma a resistir a esses ataques todos, mais trabalho será necessário realizar para assegurar que não foram cometidos enganos ao demonstrar a segurança.

Outra contribuição forte referida neste capítulo tem a ver com o desenho, desenvolvimento,

implementação e testes, nomeadamente a grande superação do algorítmo proposto em relação ao único método que encontramos na literatura para geração de matrizes não singulares pseudo-aleatórias, principalmente na resistência aos testes estatísticos levados a cabo com a biblioteca TestU01, o que prova o seu valor para aplicações criptográficas. Outro facto importante é que foi provado que o método proposto gera todas as matrizes não singulares para um determinada ordem *n* dada. A performance do sistema criptográfico proposto foi avaliada cifrando e decifrando ficheiros de diferentes tamanhos e comparando com o AES e o Salsa20, tais testes mostram que o RKHC ainda é muito inferior à esses dois algorítmos em termos de banchmarking, talvez por se tratar de uma varinate não otimizada.

No que diz respeito ao trabalho futuro, muita coisa pode ser feita para melhorar o método proposto: uma criptanálise mais aprofundada será necessária, eventualmente verificada por outras pessoas. A cifra deve ser submetida a outros métodos de criptanálise tais como: meet-in-the-middle and bicliques rebound attacks, related-key attacks, invariant subspace attacks, algebraic attacks or integral attacks, entre outros; a fim de mostrar mais claramente os pontos fortes e fracos da cifra. Além disso, é necessário implementar uma versão otimizada da mesma, implementar também algumas das outras variantes mais recentes da HC e comparar com a RKHC em termos de benchmarks para provar melhor as vantagens do método proposto em relação aos outros. Finalmente, comparar a versão otimizada com os algorítmos mais conceituados como o AES, o Salsa20, entre outras cifras simétricas.

Abstract

Hill Cipher is a classical example of a cryptosystem with interesting properties, namely that it implements the diffusion and confusion concepts coined by Shannon as essential properties for ciphers; nonetheless, its basic form is vulnerable to KPAs. This dissertation presents an efficient method to generate nonsingular key matrices, based on the Gauss-Jordan elimination procedure, which provides means to generate a new matrix per each block submitted to encryption. RKHC, described along this dissertation, uses that method and adds a step to both the encryption and decryption algorithms to deal with messages containing patterns (e.g., a sequence of zeros), in order to increase their strength against KPAs, CPAs and CCAs. A performance evaluation of a non-optimized implementation in the C programming language of RKHC is also included, compared with those of optimized implementations of AES and Salsa20, along with a discussion of its security and limitations under the well-known cryptanalysis attacks. The claim that the proposed method embeds randomness with high entropy into the generated matrices and ciphertext is corroborated by results of the TestU01 library for stringent randomness statistical tests.

Keywords

Hill Cipher, randomized key, Gauss-Jordan, nonsingular matrices, symmetric cipher, block cipher.

Contents

1 Introduction **1**
- 1.1 Motivation and Scope . . . 1
- 1.2 Problem Statement and Objectives . . . 2
- 1.3 Adopted Approach for Solving the Problem . . . 2
- 1.4 Main Contributions . . . 2
- 1.5 Dissertation Organization . . . 3

2 Background and Related Work **5**
- 2.1 Introduction . . . 5
  - 2.1.1 Cryptography Fundamentals . . . 5
  - 2.1.2 Symmetric vs. Asymmetric Ciphers . . . 6
  - 2.1.3 Pseudo Random Number Generator (PRNG) . . . 14
  - 2.1.4 Statistical Tests and TestU01 Library . . . 17
  - 2.1.5 Attack Models . . . 18
- 2.2 Classical Hill Cipher . . . 22
- 2.3 Existing Hill Cipher Modifications . . . 23
  - 2.3.1 First Attempts on Correcting the Known Plaintext Attack . . . 23
  - 2.3.2 Iterated Hill Cipher . . . 24
  - 2.3.3 Hash Hill Cipher . . . 24
  - 2.3.4 Randomized Hill Cipher . . . 24
- 2.4 Conclusions . . . 25

3 Randomized Key Hill Cipher (RKHC) **27**
- 3.1 Introduction . . . 27
- 3.2 Encryption and Decryption Key Matrix Generation . . . 27
- 3.3 Number of Generated Matrices . . . 32
- 3.4 RKHC Encryption and Decryption . . . 33
- 3.5 Cryptanalysis on RKHC . . . 34
  - 3.5.1 Basic Attack Models . . . 34
  - 3.5.2 Differential Cryptanalysis . . . 36
  - 3.5.3 Linear Cryptanalysis . . . 37
- 3.6 Conclusions . . . 37

4 RKHC Implementation, Tests and Results **39**
- 4.1 Introduction . . . 39
- 4.2 Implementation . . . 39
- 4.3 Matrix Generator Randomness Tests . . . 39
- 4.4 RKHC Benchmarking . . . 40
- 4.5 Conclusions . . . 42

5 Main Conclusions and Future Work **43**
- 5.1 Main Conclusions . . . 43
- 5.2 Future Work . . . 43

References **45**

List of Figures

2.1 A communication using cipher to assure confidentiality over an insecure channel (adapted from Stinson [Sti05]). 5
2.2 The Offset Codebook (OCB) encryption and decryption algorithms (adapted from Rogaway et al. [RBB03]). 12
2.3 Tweakable Block Chaining (TBC): a mode of operation for a tweakable block cipher. Each ciphertext becomes the tweak for the next encryption. Note that to decrypt a block, one needs only the key, C_i, and C_{i-1}, so decryption can be done in parallel. 13
2.4 Tweak Incrementation Encryption (TIE): a mode of operation for tweakable block ciphers to produce secure symmetric encryption. The initial tweak Initialization Vector (IV) is chosen randomly (or sequentially) and incremented by 1 for each successive block. 13
2.5 Tweakable Authenticated Encryption (TAE): Authenticated encryption mode based on a tweakable block cipher. 14
2.6 A one-round characteristic with probability 1 is (for any L') for DES-like cryptosystems. 20

List of Tables

4.1 Summary of the results concerning tests with the TestU01 batteries of tests (version 1.2.3 of TestU01 was used in this work). 40
4.2 Processing time (s) for Salsa20, AES-CBC and RKHC during encryption. 41
4.3 Processing time (s) for Salsa20, AES-CBC and RKHC during decryption. 41
4.4 Memory (KB) usage for Salsa20, AES-CBC and RKHC during encryption. 41
4.5 Memory (KB) usage for Salsa20, AES-CBC and RKHC during decryption. 41

List of Algorithms

1 Encryption Key Matrix Generation Algorithm. 28
2 Algorithm for Inverting the Key Matrix. 30
3 Algorithm for Directly Generating the Inverted Key Matrix. 32
4 RKHC Encryption. 34
5 RKHC Decryption. 34

Acronyms

AAD Additional Authenticated Data

AES Advanced Encryption Standard

ACM Association for Computing Machinery

ANSI American National Standards Institute

CBC Cipher Block Chaining

CCA Chosen Ciphertext Attack

CIPH Cipher

CCA2 Adaptive Chosen Ciphertext Attack

CFB Cipher Feedback

CMAC Cipher-based Message Authentication Code

COA Ciphertext Only Attack

CPA Chosen Plaintext Attack

CPU Central Processing Unit

CSPRNG Cryptographically Secure Pseudo Random Number Generator

CCS Computing Classification System

CTR Counter

DDR3 Double Data Rate type 3

DES Data Encryption Standard

3DES Triple Data Encryption Standard

ECB Electronic Code Book

FIPS Federal Information Processing Standard

GB Gigabyte

GF(2) Binary Galois Field

GCM Galois Counter Mode

GCTR Galois Counter

GHz Gigahertz

GHASH Galois Hash (Function) - defined as $GHASH(H, A, C) = X_{m+n+1}$. H is the Hash Key (a string of 128 zero bits encrypted using the block cipher), A is data which is only authenticated (not encrypted), C is the ciphertext, m is the number of 128-bit blocks in A, n is the number of 128-bit blocks in C.

GMAC Galois Message Authentication Code

HC Hill Cipher

HRNG Hardware Random Number Generator

INAGBE Instituto Nacional de Bolsas de Estudo (Angola)

ISP Instituto Superior Politécnico (Angola)

IBM International Business Machines

IDEA International Data Encryption Algorithm

ISAAC Indirection, Shift, Accumulate, Add And Count

ISO International Organization for Standardization

IV Initialization Vector

KB Kilobyte

KPA Known Plaintext Attack

LSB Less Significant Bit

MB Megabyte

MDS Maximum Distance Separable

MAC Message Authentication Code

MSB Most Significant Bit

NIST National Institute of Standards and Technology

OFB Output Feedback

OCB Offset Codebook

OS Operating System

PCBC Propagating Cipher Block Chaining

PKCS5 Public Key Cryptography Standards 5

PKCS7 Public Key Cryptography Standards 7

PRNG Pseudo Random Number Generator

PRP Pseudo Random Permutation

RAM Random Access Memory

RC4 Rivest Cipher 4

RC5 Rivest Cipher 5

RC6 Rivest Cipher 6

RKHC Randomized Key Hill Cipher

RNG Random Number Generator

SSL Secure Sockets Layer

TAE Tweakable Authenticated Encryption

TBC Tweakable Block Chaining

TIE Tweak Incrementation Encryption

TLS Transport Layer Security

TRNG True Random Number Generator

UBI Universidade da Beira Interior

UKB Universidade Katyavala Bwila (Angola)

W3C World Wide Web Consortium

WEP Wired Equivalent Privacy

XOR exclusive OR

Chapter 1

Introduction

This document describes a project developed as part of the work for the attainment of a master degree in Computer Science and Engineering at Universidade da Beira Interior (UBI). The dissertation addresses the subject of Implementing and Evaluating Nonsingular Matrices Generators for the Hill Cipher. Section 1.1 presents the motivation for the project and its scope. The two subsequent sections present the adopted approach and the main contributions of this work, respectively. Finally, the last section describes the structure of the dissertation, composed of five chapters.

1.1 Motivation and Scope

Hill Cipher (HC) is a classical example of a cryptosystem with interesting properties, namely the implementation of the diffusion and confusion concepts, which Shannon determined as essential properties for ciphers; HC provides a very high level of entropy, obtained through matrix multiplications to which plaintexts are submitted to. Another HC advantage is the simplicity of the encrypting and decrypting processes. It has, thus, attracted the interest of many researchers along the years, in an attempt to deal with its limitations, which were not solved completely until now, namely the Known Plaintext Attack (KPA) vulnerability, when the same matrix is used to encrypt many plaintext blocks, and others, such as Chosen Plaintext Attack (CPA) and Chosen Ciphertext Attack (CCA).

The main motivation of this work comes from the study of the classical HC, though it mostly focuses on its most important component, the matrix that is used as encryption key of the cipher. This matrix has to be nonsingular, so that it can be inverted for the decryption process, and generating pseudo-random matrices with such property has received little attention from the research community. This work describes a new method to generate such matrices and analyses it in the scope of a modified HC. In this case, HC was modified to improve its security, and it is part of the contributions of this work.

The scope of this dissertation encompasses the areas of cryptography and computer security, as it aims to evaluate and explore the aforementioned new method applied on a modified HC that does not have its traditional vulnerabilities. Under the 2012 version of the ACM Computing Classification System (CCS), the topics that best describe this dissertation are the following:

- ***Security and privacy – Block and stream ciphers***;
- **Security and privacy – Cryptanalysis and other attacks**;
- *Theory of computation – Cryptographic protocols*.

1.2 Problem Statement and Objectives

The main problem addressed by this work is the Hill Cipher (HC) vulnerability against KPA, when many plaintext blocks are encrypted using the same key matrix, which makes it easy to recover the key matrix by solving a trivial system of linear equations. Other problem that this work attempts to solve is that, to the best of our knowledge, there are no efficient methods in the literature to generate nonsingular matrices that can be used as key matrices for each plaintext block. Besides the KPA vulnerability, HC is also vulnerable to CPA and CCA (e.g., for a block of zeroes as input, HC always outputs a block of zeros as well).

The main objective of this dissertation is, then, to design a secure HC variant. In order to reach this aim, it is necessary to: a) present an efficient method to generate pseudo-random nonsingular matrices; b) modify the HC encryption and decryption algorithms in order to prevent KPA, CPA and CCA; c) evaluate the proposed method and modifications.

1.3 Adopted Approach for Solving the Problem

The chosen approach to solve the aforementioned problem began with the study of existing HC modifications. These different modifications were important to ascertain what was already made against KPA, CPA and CCA vulnerabilities. Next, and as none of them presents an efficient method for generating pseudo-random nonsingular matrices, a method based on the Gauss-Jordan elimination procedure was proposed, implemented and analyzed. This method is capable of generating all needed key matrices. The next step consisted in thinking how the HC susceptibility to specific plaintexts or ciphertexts (such as 0 filled vectors) could be addressed and proposing modification that solve these problems, allowing the algorithm to be resistant to KPA, CPA and CCA. In order to test the proposed method, a non-optimized implementation in the ANSI-C programming language of Randomized Key Hill Cipher (RKHC) was, then, performed and compared to optimized implementations of Advanced Encryption Standard (AES) and Salsa20 in terms of benchmarking. The well-known TestU01 library for stringent randomness statistical tests was used against the proposed nonsingular matrix generation algorithm, as to ensure that its outputs have enough entropy to be used in cryptography scenarios.

1.4 Main Contributions

The main contributions achieved from the research and development of this project can be summarized as follows:

1. An efficient algorithm to generate a pseudo-random nonsingular matrix for a given seed was proposed, implemented in ANSI-C programming language and evaluated. The novelty of this method is that it generates all possible nonsingular matrices efficiently and with very good space coverage properties;

2. An efficient algorithm to invert a given nonsingular matrix was proposed, implemented and evaluated;

3. An efficient algorithm to directly generate an inverted matrix based specifically on the algorithm mentioned as first contributions was also proposed, implemented and evaluated;

4. The dissertation also describes a HC modification (RKHC), namely the algorithms for encryption and decryption using the method above to encrypt/decrypt each plaintext/ciphertext block with a different pseudo-random matrix with additional means to prevent KPA, CPA and CCA;

5. An initial cryptanalysis on HC and RKHC using several attack models was also part of this work.

The aforementioned contributions are planned to be part of the subject of an international journal paper. This publication will potentially include a lengthier version of the cryptanalysis and considerations on optimized implementations of the algorithms.

1.5 Dissertation Organization

This dissertation is organized as follows:

- Chapter 1 – Introduction – presents the main motivations of this work, its scope, the addressed problem and objectives, the adopted approach for solving the addressed problem, the main contributions and the structure of the document;
- Chapter 2 – Background and Related Work – presents concepts and definitions that are important to understand the work described herein, as well as previous works concerning the HC;
- Chapter 3 – Randomized Key Hill Cipher (RKHC) – discusses the proposed RKHC as a secure variant of HC, and includes an analysis of the most common attack models used against block ciphers;
- Chapter 4 – RKHC Implementation, Tests and Results – describes all of the tests performed on the RKHC implementation, and their results, and discusses as well a comparison with other state-of-the-art algorithms;
- Chapter 5 – Main Conclusions and Future Work – analyses the developed work, as well as its main results and limitations from a broader perspective, and identifies potential future work that can be pursued to improve it.

Chapter 2

Background and Related Work

2.1 Introduction

This chapter describes the background and main concepts related to this work, from cryptography fundamentals, symmetric and asymmetric ciphers, PRNG and CSPRNG, the concept of statistical tests and the well-known TestU01 library. The most common attack models are presented as well. Finally, the classical HC cryptosystem, its strengths and weaknesses and several works related to posterior improvements are described as well.

2.1.1 Cryptography Fundamentals

According to Stinson [Sti05], the fundamental objective of Cryptography is to enable two people, typically referred, nowadays, to as Alice and Bob, to communicate over an insecure channel in such a way that an opponent, Oscar, cannot understand what is being said. Alice encrypts the plaintext, using a predetermined key, and sends the resulting ciphertext over the channel. Oscar, upon seeing the ciphertext in the channel by eavesdropping, cannot determine what the plaintext was; but Bob, who knows the encryption key, can decrypt the ciphertext and reconstruct the plaintext. Figure 2.1 illustrates this description.

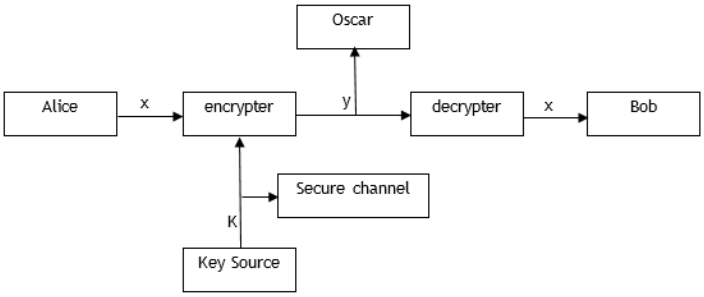

Figure 2.1: A communication using cipher to assure confidentiality over an insecure channel (adapted from Stinson [Sti05]).

Stinson refers to Cryptanalysis as the process of discovering the plaintext from the ciphertext without knowing the decryption key in an efficient manner, and a cryptographic algorithm or cipher as a mathematical function employed for the encryption and decryption of messages. Ciphers can be divided into two categories: symmetric key and public key ciphers. Both are defined by Laskari et al. [LMSV07]. In symmetric-key ciphers, the sender and the receiver of the message, secretly, choose and share the **same** key that will be used for encryption and decryption. A drawback of this type of cryptosystems is that it requires prior communication of the key between the sender and the receiver through a secure channel, before any message is sent. Public-key ciphers, on the other hand, are designed in such a way that the key used for encryption is publicly available and differs from the key used in decryption, which is secret. Although these two keys are functionally interrelated, the computation of the secret key from

the public key is computationally intractable. Thus, using the public key, anyone can send an encrypted message, but only the owner of the secret key can perform the decryption. These different categories of ciphers are briefly described with examples in the next subsection.

2.1.2 Symmetric vs. Asymmetric Ciphers

According to Goldreich [Gol09], a fundamental distinction between encryption schemes refers to the relation between the aforementioned pairs of the keys (i.e., the encryption-key and the decryption-key). The simpler (and older) notion assumes that the encryption-key equals the decryption-key; such schemes are called private-key (or symmetric).

2.1.2.1 Stream Ciphers vs. Block Ciphers

This dissertation is focused on symmetric cipher systems, which are generally classified into block and stream ciphers. According to Cusick [CDR04], the essential distinction between them is as follows:

A stream cipher specifies a device with internal memory that encrypts the j_{th} digit m_j of the message stream into the j_{th} digit c_j of the ciphertext stream by means of a function which depends on both the secret key k and the internal state of the stream cipher at time j. The sequence $z^{\infty} = z_0 z_1 ...$ which controls the encryption is called the *key stream* or the *running key*. The deterministic automaton which produces the key stream from the actual key k and the internal state is called the *key stream generator* or the *running key generator*.

A block cipher breaks each plaintext message into successive blocks and encrypts each block M under the control of a key k into a ciphertext block $C = (c_1, ..., c_n)$, where the plaintext and the ciphertext alphabet are usually identical. Each block is typically several characters long. Simple substitution and homophonic substitution ciphers are examples of block ciphers, even though the unit of encryption is a single character. This is because the same key is used for each character. A block cipher is, generally, created using a Pseudo Random Permutation (PRP) which, as described, e.g., by Luby and Rackoff [LR88], is a function that cannot be distinguished from a random permutation and can be built from a pseudo-random function using a Luby–Rackoff construction, which is built using a Feistel network, in order to be a strong PRP.

2.1.2.2 Stream Ciphers Examples

Some examples of well-known stream ciphers can be briefly described as follows:

- Rivest Cipher 4 (RC4) — designed and presented in 1987 by Ron Rivest [Riv87], RC4 is a cipher that uses a large internal state that is stored in an array of words. Because of a simplicity of design and a high speed offered by software implementation, the cipher has gained popularity in many internet applications such as TLS/SSL (Transport Layer Security/Secure Sockets Layer) and Wired Equivalent Privacy (WEP) [OPSS13]. More recently, its use has been decreasing.
- Salsa20 — according to Bernstein [Ber08b], Salsa20 is a family of 256-bit stream ciphers designed in 2005 and submitted to eSTREAM, the ECRYPT Stream Cipher Project. Salsa20 has progressed to the third round of eSTREAM without any changes. The 20-round stream

cipher Salsa20/20 is consistently faster than AES and is recommended by the designer for typical cryptographic applications. The reduced-round ciphers Salsa20/12 and Salsa20/8 are among the fastest 256-bit stream ciphers available and are recommended for applications where speed is more important than confidence. The fastest known attacks use ≈ 2^{153} simple operations against Salsa20/7, ≈ 2^{249} simple operations against Salsa20/8, and ≈ 2^{255} simple operations against Salsa20/9, Salsa20/10, etc.
According to its designer, Daniel J. Bernstein [Ber05], the core of Salsa20 is a hash function with 64-byte input and 64-byte output. The hash function is used in counter mode as a stream cipher: Salsa20 encrypts a 64-byte block of plaintext by hashing the key, nonce, and block number and XORing the result with the plaintext. The designer defines the Salsa20 hash function, the Salsa20 expansion function, and the Salsa20 encryption function.

In Ecrypt-Net research [Sal12], it is shown that the algorithm supports keys of 128 bits and 256 bits. During its operation, the key, a 64-bit nonce (unique message number), a 64-bit counter and four 32-bit constants are used to construct the 512-bit initial state. After r iterations of the Salsa20/r round function, the updated state is used as a 512-bit keystream output. Each output block is an independent combination of the key, nonce, and counter and, since there is no chaining between blocks, the operation of Salsa20/r resembles the operation of a block cipher in counter mode. Salsa20/r, therefore, shares the very same implementation advantages, in particular the ability to generate output blocks in any order and in parallel. The maximum length of the keystream produced by Salsa20/r is 2^{70} bits.

The round transformation of Salsa20 uses a combination of three simple operations: addition modulo 2^{32}, bit rotation and exclusive OR (XOR). The efficient implementation of these operations in software gives the good software performance of the cipher. Within eSTREAM, three main variants of Salsa20 – depending on the number of rounds r – were proposed: Salsa20/8, Salsa20/12 and Salsa20/20. Each provides a different security vs. performance trade-off. Salsa20/20 is recommended by the designer for "encryption in typical cryptographic applications". The eSTREAM committee suggested the use of Salsa20/12, as offering the best balance among the different versions, combining very good performance with a comfortable margin of security.

This stream cipher is of particular interest in this work, because it was chosen as the Cryptographically Secure Pseudo Random Number Generator (CSPRNG) that was used to generate the non-singular matrices in a real implementation of the proposed modified HC. Nonetheless, any CSPRNG could be used for that purpose.

- ChaCha8 — Bernstein [Ber08a] presented a 256-bit stream cipher based on the 8-round cipher Salsa20/8. The changes from Salsa20/8 to ChaCha8 improve diffusion per round, Conjecturally increasing resistance to cryptanalysis, while preserving (and often improving) time per round. ChaCha12 and ChaCha20 are analogous modifications of the 12-round and 20-round ciphers Salsa20/12 and Salsa20/20, respectively. Chacha8 was not chosen for the tests presented in this work because it did not increase the performance of the tests compared to Salsa8, curiously.

2.1.2.3 Block Ciphers Examples

Some examples of well-known block ciphers can be briefly described as follows:

- Data Encryption Standard (DES) — DES was the first encryption standard to be recommended by National Institute of Standards and Technology (NIST) (e.g., Singh and Main [SM11]). It is based on an algorithm proposed by International Business Machines (IBM) called Lucifer. DES became a standard in 1974. Since that time, many attacks and methods were discovered that exploit weaknesses of DES, which made it an insecure block cipher. Triple Data Encryption Standard (3DES) is an enhancement of DES. In this standard, the encryption method is similar to the one in original DES but applied 3 times to increase the encryption level. However, it is a known fact that 3DES is slower than other block cipher methods. DES uses 64-bit block size and 56-bit keys, which makes it vulnerable to nowadays brute force attacks [Mat94].

- International Data Encryption Algorithm (IDEA) — IDEA is a 128 bits key block cipher algorithm designed by Xuejia Lai and James L. Massey of ETH-Zürich, and was first described in 1991 [Bas11]. The original algorithm went through few modifications and was finally named as IDEA. The mentioned algorithm works on 64-bit plaintext and ciphertext blocks (at one time). For encryption, the 64- bit plain text is divided into four 16 bits sub-blocks. Each of these blocks goes through 8 rounds and one Output Transformation Phase.

- Rivest Cipher 5 (RC5) – according to Rivest [Riv94], RC5 has a variable word size, a variable number of rounds and a variable-length secret key. RC5 works with two-word input (plaintext) block size and a two-word output (ciphertext) block size. The nominal choice for word is 32 bits, and RC5 operates over 64-bit plaintext and ciphertext block sizes. Allowed values for words are 16, 32 and 64. the size of the plaintext and ciphertext blocks is always twice the value of the size of the words. The number of rounds and the number of bytes in the secret key is a value from 1 to 255.

- Blowfish — Blowfish was designed in 1993 by Bruce Schneier [Sch94, Sch95] working under a Feistel network with 16 rounds, 64-bits block size and a key from 32-bits to 448-bits (default key of 128 bits).

- Advanced Encryption Standard (AES) — AES was presented in 2001 by NIST as the USA Standard [Sta01, Pub01] for symmetric key cipher, and the project winner was Rijndael, developed by Vincent Rijmen and Joan Daemen, beating others like: MARS, Rivest Cipher 6 (RC6), Serpent and Twofish. AES operates over 128 bits sized blocks and with keys with sizes of 128 bits (10 rounds), 192 bits (12 rounds) and 256 bits (14 rounds). It is, currently, considered the *de facto* standard in terms of symmetric key cryptography, with many modern processors providing native optimized functions or implementations of this cipher.

2.1.2.4 Block Ciphers Modes of Operation

Some examples of well-known block ciphers modes of operation are presented below. This presentation was inspired by a Dworkin work [Dwo01]:

- Electronic Code Book (ECB) — ECB is the default mode of operation of block ciphers, and can be simply described as follows:
  ECB Encryption: $C_j = \mathsf{CIPH}_K(P_j)\ for\ j = 1...n$;
  ECB Decryption: $P_j = \mathsf{CIPH}_K^{-1}(C_j)\ for\ j = 1...n$.

- Cipher Block Chaining (CBC) — CBC is a mode whose encryption process features the combining (i.e., *chaining*) of the plaintext blocks with the previous ciphertext blocks to improve

the cascade effect. The CBC mode requires an Initialization Vector (IV), which is combined with the first block of plaintext (thus natively supports re-usage of the same key for different messages). The IV does not need to be secret, but it must be unpredictable. CBC is one of the most widely used cipher modes, specially for encryption of files.

CBC encryption (and decryption) can be described with recursive formulas:
$C_1 = \mathsf{CIPH}_K(P_1 \oplus \mathsf{IV})$;
$C_j = \mathsf{CIPH}_K(P_j \oplus C_{j-1})\ for\ j = 2...n$.

CBC Decryption:
$P_1 = \mathsf{CIPH}_K^{-1}(C_1) \oplus \mathsf{IV}$;
$P_j = \mathsf{CIPH}_K^{-1}(C_j) \oplus (C_{j-1})\ for\ j = 2...n$.

- Propagating Cipher Block Chaining (PCBC) — PCBC is a variant of CBC, in which each block of plaintext is XORed with both the previous plaintext block and the previous ciphertext block before being encrypted. As with CBC mode, an IV is used in the first block.

- Cipher Feedback (CFB) — CFB features the feedback of successive ciphertext segments into the input blocks of the forward cipher to generate output blocks that are XORed with the plaintext to produce the ciphertext, and vice versa. The CFB mode requires an IV as the initial input block. The CFB mode also requires an integer parameter, denoted s, such that $1 \leq s \leq b$, with b denoting the size of the block. In the specification of the CFB mode below, each plaintext segment ($P_j^{\#}$) and ciphertext segment ($C_j^{\#}$) consist of s bits. The value of s is sometimes incorporated into the name of the mode, e.g., the 1-bit CFB mode, the 8-bit CFB mode, the 64-bit CFB mode, or the 128-bit CFB mode.

  CFB Encryption:
  $I_1 = \mathsf{IV}$;
  $I_j = \mathsf{LSB}_{b-s}(I_{j-1}) | C_{j-1}^{\#}\ for\ j = 2...n$;
  $O_j = \mathsf{CIPH}_K(I_j)\ for\ j = 1, 2...n$;
  $C_j^{\#} = P_j^{\#} \oplus \mathsf{MSB}_s(O_j)\ for\ j = 1, 2...n$.

  CFB Decryption:
  $I_1 = \mathsf{IV}$;
  $I_j = \mathsf{LSB}_{b-s}(I_{j-1}) | C_{j-1}^{\#}\ for\ j = 2...n$;
  $O_j = \mathsf{CIPH}_K(I_j)\ for\ j = 1, 2...n$;
  $P_j^{\#} = C_j^{\#} \oplus \mathsf{MSB}_s(O_j)\ for\ j = 1, 2...n$.

- Output Feedback (OFB) — OFB features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are XORed with the plaintext to produce the ciphertext, and vice versa. The OFB mode requires that the IV is a nonce, i.e., the IV must be unique for each execution of the mode under the given key.

  OFB Encryption:
  $I_1 = \mathsf{IV}$;
  $I_j = O_{j-1}\ for\ j = 2...n$;
  $O_j = \mathsf{CIPH}_K(I_j)\ for\ j = 1, 2...n$;
  $C_j = P_j \oplus O_j\ for\ j = 1, 2...n-1$;
  $C_n^* = P_n^* \oplus \mathsf{MSB}_u(O_n)$.

  OFB Decryption:
  $I_1 = \mathsf{IV}$;
  $I_j = O_{j-1}\ for\ j = 2...n$;

$O_j = \mathsf{CIPH}_K(I_j)\ for\ j = 1, 2...n$;
$P_j = C_j \oplus O_j\ for\ j = 1, 2...n-1$;
$P_n^* = C_n^* \oplus \mathsf{MSB}_u(O_n)$.

- Counter (CTR) — CTR is a confidentiality mode that features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are XORed with the plaintext to produce the ciphertext, and vice-versa. The sequence of counters must have the property that each block in the sequence is different from every other block. This condition is not restricted to a single message: across all the messages that are encrypted under the same key, all the counters must be distinct. In this recommendation, the counters for a given message are denoted $T_1, T_2, ..., T_n$.

  CTR Encryption:
  $O_j = \mathsf{CIPH}_K(T_j)\ for\ j = 1, 2...n$;
  $C_j = P_j \oplus O_j\ for\ j = 1, 2...n-1$;
  $C_n^* = P_n^* \oplus \mathsf{MSB}_u(O_n)$.

  CTR Decryption:
  $O_j = \mathsf{CIPH}_K(T_j)\ for\ j = 1, 2...n$;
  $P_j = C_j \oplus O_j\ for\ j = 1, 2...n-1$;
  $P_n^* = C_n^* \oplus \mathsf{MSB}_u(O_n)$.

- Galois Counter Mode (GCM) — according to McGrew and Vieg [MV04], GCM is a block cipher mode of operation that uses universal hashing [1] over a GF(2) (Binary Galois Field) [2] to provide authenticated encryption [3]. It can be implemented in hardware to achieve high speeds with low cost and low latency. Software implementations can achieve excellent performance by using table-driven field operations. It uses mechanisms that are supported by a well-understood theoretical foundation, and its security follows from a single reasonable assumption about the security of the block cipher. This mode admits pipelined and parallelized implementations and have minimal computational latency in order to be useful at high data rates. CTR has emerged as the best method for high-speed encryption, but it provides no protection against bit-flipping attacks. GCM fills this need. Below is presented the definition of GCM for 128-bit block ciphers as the standard size. The mode is slightly different when applied to 64-bit block ciphers.
  GCM has two operations, authenticated encryption and authenticated decryption. The cryptosystem presented by Dworkin [Dwo07] uses the following prerequisites: approved block Cipher (CIPH) with a 128-bit block size; key K; definitions of supported input-output lengths; supported tag length t associated with the key.

  The authenticated encryption operation has the following inputs, each of which is a bit string:

  - IV (whose length is supported);
  - Plaintext P (whose length is supported);
  - Additional Authenticated Data (AAD) A (whose length is supported).

  There are two outputs:

[1] Hash function selected at random from a family of hash functions with a certain mathematical property.
[2] A finite field of elements, thus GF (2) is a Binary Galois Field (finite field with only 0 and 1).
[3] Encryption which simultaneously provides confidentiality, integrity, and authenticity on the data.

- A ciphertext C;
- an Authentication tag Y.

Encryption steps:

1. Let $H = \text{CIPH}_K(0^{128})$.
2. Define a block, J_0, as follows:
   If len (IV) = 96, then let J_0 = IV $||0^{31}||1$.
   If len (IV) $\neq$ 96, then let s = 128 . [len(IV)/128]-len(IV), and let
   J_0 = GHASH $_H$(IV$||0^{s+64}||[\text{len(IV)}]_{64}$).
3. Let C = GCTR $_K$(inc$_{32}$(J_0), P).
4. Let u = 128 . [len (C)/128] - len (C) and let v = 128 . [len (A)/128] - len (A).
5. Define a block, S, as follows:
   S = GHASH $_H$ (A || 0^v || C || 0^u || $[\text{len(A)}]_{64}$ || $[\text{len(C)}]_{64}$).
6. Let $T = \text{MSB}_t$ (GCTR $_K$(J_0, S))
7. Return (C, T).

The decryption operation has the following inputs:

- IV (whose length is supported);
- Ciphertext C;
- AAD A;
- Authentication tag T.

The output is either the plaintext P or indication of an authenticity FAIL.

Decryption steps:

1. If the bit lengths of IV, A or C are not supported, or if len(T) $\neq$ t, then return FAIL.
2. Let $H = \text{CIPH}_K(0^{128})$.
3. Define a block, J_0, as follows:
   If len (IV) = 96, then let J_0 = IV $||0^{31}||1$.
   If len (IV) $\neq$ 96, then let s = 128 . [len(IV)/128]-len(IV), and let
   J_0 = GHASH $_H$(IV$||0^{s+64}||[\text{len(IV)}]_{64}$).
4. Let $P = \text{GCTR}_K(\text{inc}_{32}(J_0), C)$.
5. Let u = 128 . [len (C)/128] - len (C) and let v = 128 . [len (A)/128] - len (A).
6. Define a block, S, as follows:
   S = GHASH $_H$ (A || 0^v || C || 0^u || $[\text{len(A)}]_{64}$ || $[\text{len(C)}]_{64}$).
7. Let $T' = \text{MSB}_t$ (GCTR $_K$(J_0, S))
8. If $T = T'$, then return P; then return FAIL.

According to McGrew and Vieg [MV04], the strength of the authentication of P, IV and A is determined by the length of the authentication tag. When the length of P is zero, GCM acts as a Message Authentication Code (MAC) on the input A. The mode of operation that uses GCM as a stand-alone message authentication code is denoted as Galois Message Authentication Code (GMAC). An alternative to the GMAC mode for Authentication is Cipher-based Message Authentication Code (CMAC) presented by [Dwo16], which is based on a symmetric key block cipher.

- Offset Codebook (OCB) — according to Rogaway et al. [RBB03], OCB is another mode of operation that simultaneously provides confidentiality and authenticity. OCB encrypts and authenticates a nonempty string $M\{0,1\}$ using $[|M|/n]+2$ block-cipher invocations, where n is the block length of the underlying block cipher. Additional over-head is small. Desirable properties of OCB include the ability to encrypt a bit string of arbitrary length into a ciphertext of minimal length, cheap offset calculations, cheap key setup, a single underlying cryptographic key, no extended-precision addition, a nearly optimal number of block-cipher calls, and no requirement for a random IV. The scheme is presented in Figure 2.2.

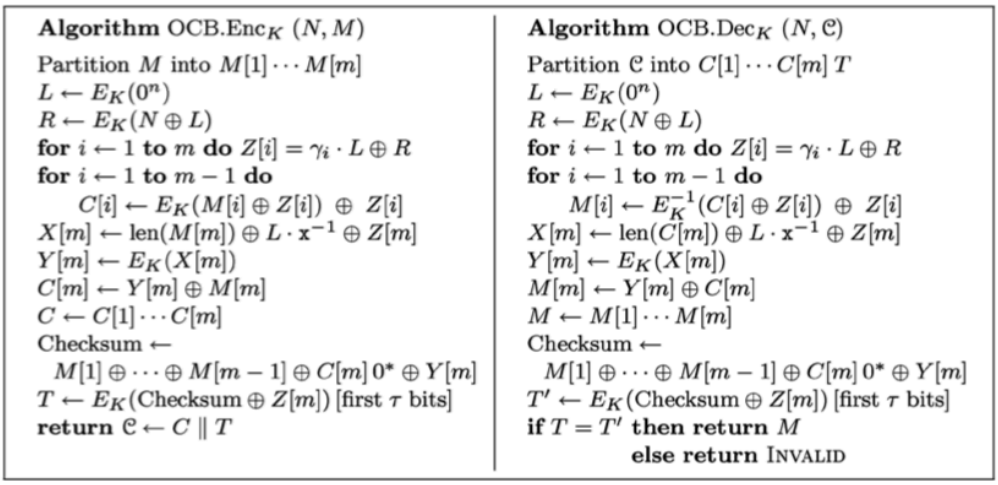

Figure 2.2: The OCB encryption and decryption algorithms (adapted from Rogaway et al. [RBB03]).

- Tweakable Block Ciphers — tweakable block ciphers were presented by Liskov et al. [LRW02] as a cryptographic primitive in which a cipher has not only the usual inputs — message and cryptographic key — but also a third input, the *tweak*. The *tweak* serves much the same purpose that an IV does for CBC mode or that of a nonce for OCB mode. Liskov et al. [LRW11] present a strong tweakable block cipher which uses a family H of functions with signature $\{0,1\}^t \longrightarrow \{0,1\}^n$ that is said to be an *ε-almost 2-xor-universal* hash function family if $Pr_h[h(x) \oplus h(y) = z] \leq \varepsilon$ holds for all x, y, z with $x \neq y$, where the probability is taken over h chosen uniformly at random from H (a hash function family). This strong tweakable block cipher uses a key (K, h) where $K \longrightarrow \{0,1\}^k$ and $h \longleftarrow H$, and is given by:
$E_{K,h}(T, M) = E_K(M \oplus h(T)) \oplus h(T)$,
$D_{K,h}(T, C) = D_K(C \oplus h(T)) \oplus h(T)$.

  Tweakable modes of operation are presented as well, namely the Tweakable Block Chaining (TBC) (schematized in Figure 2.3), Tweak Incrementation Encryption (TIE) (schematized in Figure 2.4) and Tweakable Authenticated Encryption (TAE) (schematized in Figure 2.5). Yan [Yan15] says the goals for tweakable block ciphers and modes of operations are: achieving efficiency (since changing the tweak should be less costly than changing the key) and security (since even if an adversary has control of the tweak input, the tweakable block cipher may remain secure). All of this is made by separating the roles of cryptographic key (uncertainty) from that of tweak (variability).

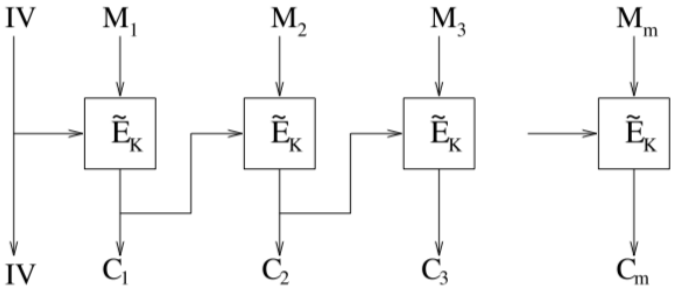

Figure 2.3: TBC: a mode of operation for a tweakable block cipher. Each ciphertext becomes the tweak for the next encryption. Note that to decrypt a block, one needs only the key, C_i, and C_{i-1}, so decryption can be done in parallel.

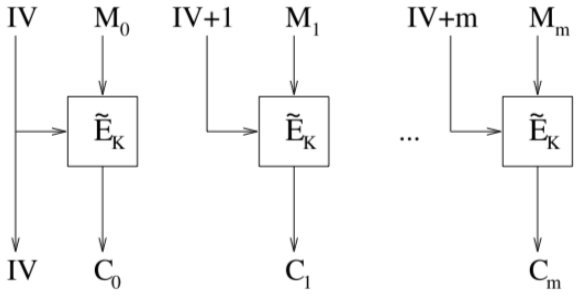

Figure 2.4: TIE: a mode of operation for tweakable block ciphers to produce secure symmetric encryption. The initial tweak IV is chosen randomly (or sequentially) and incremented by 1 for each successive block.

2.1.2.5 Padding

For the ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments) [Dwo01]. In other words, for these three modes, the total number of bits in the plaintext must be a (positive) multiple of the block (or segment) size. If the data string to be encrypted does not, initially, satisfy this property, then the formatting of the plaintext must entail an increase in the number of bits. A common way to achieve the necessary increase is to append some extra bits, called *padding*, to the trailing end of the data string as the last step in the formatting of the plaintext. An example of a padding method is to append a single `1` bit to the data string and, then, to pad the resulting string by as few `0` bits, possibly none, as necessary to complete the final block (segment). Other methods may be used; in general, the formatting of the plaintext is outside the scope of this recommendation. For the above padding method (bit padding), the padding bits can be removed unambiguously, the receiver can determine that the message is, indeed, padded. One way to ensure that the receiver does not mistakenly remove bits from an unpadded message is to require the sender to pad every message, including messages in which the final block (segment) is already complete. For such messages, an entire block (segment) of padding is appended. Alternatively, such messages can be sent without padding if, for every message, the existence of padding can be reliably inferred, e.g., from a message length indicator. A padding scheme similar to bit padding is the *Trailing Bit Complement* padding, in which, if the data ends in a `0` bit, all the padding bits will be ones (`1`s). If the data ends in a `1` bit, all the padding bits will be zeros (`0`s) [pada]. Another padding scheme similar to the bit padding is the ISO/IEC 7816-4 [ISO05]. While in Bit Padding we start with a `1` bit, in this case we start by a mandatory byte valued `0x80` followed by as many `00` as possible. This scheme is also known OneAndZeroes Padding [padb]

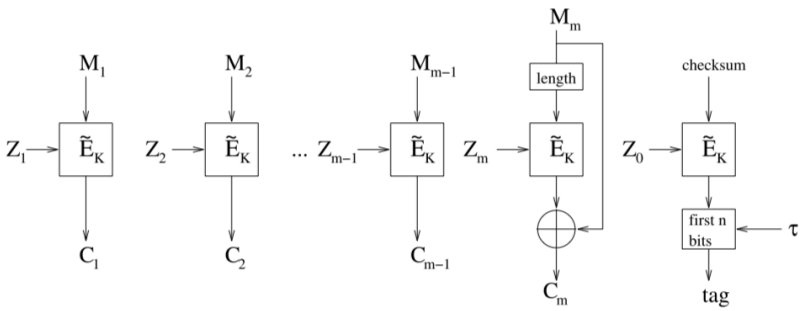

Figure 2.5: TAE: Authenticated encryption mode based on a tweakable block cipher.

or only ISO 7816-4 [pada]. Below, several other byte padding schemes used in symmetric block ciphers are briefly presented:

- Zero (Byte) Padding — all padding bytes are zeros. This type of padding is rather unreliable (what happens if the data ends with zeros?) and should be used only if necessary in legacy applications [pada];
- ANSI X.923 — in ANSI X.923, if N padding bytes are required ($1 < N \leq B$), set the last byte as N and all the preceding $N - 1$ padding bytes as zero [padb];
- ISO 10126 — in ISO 10126, the last byte of the padding (thus, the last byte of the block) is the number of pad bytes [pada]. All other bytes of the padding are some random data [pada]. W3C Padding is similar to ISO 10126 padding. This method is not recommended since only one padding byte is ever checked when decrypting and this opens up security vulnerabilities [padb];
- PKCS5 and PKCS7 — in PKCS5, if the block length is B, then add N padding bytes of value N to make the input length up to the next exact multiple of B. If the input length is already an exact multiple of B then add B bytes of value B. Thus padding of length N between one and B bytes is always added in an unambiguous manner. After decrypting, check that the last N bytes of the decrypted data all have value N with $1 < N \leq B$. If so, strip N bytes, otherwise throw a decryption error [padb]. PKCS7 is identical to PKCS5 [pada], but PKCS5 has only been defined for block ciphers that use a 64-bit (8-byte) block size.

2.1.3 Pseudo Random Number Generator (PRNG)

According to Rukhin et al. [RSN+01], a PRNG uses one or more inputs and generates multiple pseudo-random numbers. Inputs to PRNG are called seeds. In contexts in which unpredictability is needed, the seed itself must be random and unpredictable. Hence, by default, a PRNG should obtain its seeds from the outputs of a true Random Number Generator (RNG); i.e., a PRNG requires a RNG as a companion.

The outputs of a PRNG are deterministic functions of the seed; i.e., all true randomness is confined to the seed generation. The deterministic nature of the process leads to the pseudo-random term. Since each element of a pseudo-random sequence is reproducible from its seed, only the seed needs to be saved if reproduction or validation of the pseudo-random sequence is required. Ironically, pseudo-random numbers often appear to be more random than random numbers obtained from physical sources. If a pseudo-random sequence is properly constructed, each value in the sequence is produced from the previous value via transformations which appear

to introduce additional randomness. A series of such transformations can eliminate statistical auto-correlations between input and output. Thus, the outputs of a PRNG may have better statistical properties and be produced faster than a RNG. Random and pseudo-random numbers generated for cryptographic applications should be unpredictable. In the case of PRNGs, if the seed is unknown, the next output number in the sequence should be unpredictable in spite of any knowledge of previous random numbers in the sequence. This property is known as forward unpredictability. It should also not be feasible to determine the seed from knowledge of any generated values (i.e., backward unpredictability is also required). No correlation between a seed and any value generated from that seed should be evident. To ensure forward unpredictability, care must be exercised in obtaining seeds. The values produced by a PRNG are completely predictable if the seed and generation algorithm are known. Since in many cases the generation algorithm is publicly available, the seed must be kept secret and should not be derivable from the pseudo-random sequence it produces. In addition, the seed itself must be unpredictable.

2.1.3.1 Random Number

According to Kelsey et al. [KSF99], a random number is a number that cannot be predicted by an observer before it is generated. If the number is to be in the range $0 \ldots 2^n - 1$, an observer cannot predict that number with probability any better than $1/2^n$. If m random numbers are generated in a row, an observer, given any $m-1$ of them, still, cannot predict the mth with any better probability than $1/2^n$.

2.1.3.2 Hardware Random Number Generators (HRNGs)

An Hardware Random Number Generator (HRNG) or True Random Number Generator (TRNG) or RNG uses a non-deterministic source (i.e., the entropy source), along with some processing function (i.e., the entropy distillation process) to produce randomness [RSN+01]. The use of a distillation process is needed to overcome any weakness in the entropy source that results in the production of non-random numbers (e.g., the occurrence of long strings of zeros or ones). The entropy source typically consists of some physical quantity, such as the noise in an electrical circuit, the timing of user processes (e.g., key strokes or mouse movements), or the quantum effects in a semiconductor. Various combinations of these inputs may be used. The outputs of an HRNG may be used directly as a random number or may be fed into a PRNG. To be used directly (i.e., without further processing), the output of any TRNG needs to satisfy strict randomness criteria as measured by statistical tests in order to determine that the physical sources of the TRNG inputs appear random. For example, a physical source such as electronic noise may contain a superposition of regular structures, such as waves or other periodic phenomena, which may appear to be random, yet are determined to be non-random using statistical tests. For cryptographic purposes, the output of RNG needs to be unpredictable. However, some physical sources (e.g., date/time vectors) are quite predictable. These problems may be mitigated by combining outputs from different types of sources to use as the inputs for an RNG. However, the resulting outputs from the RNG may still be deficient when evaluated by statistical tests. In addition, the production of high-quality random numbers may be too-time consuming, making such production undesirable when a large quantity of random numbers is needed. To produce large quantities of random numbers, PRNGs may be preferable.

2.1.3.3 Cryptographically Secure Pseudo Random Number Generator (CSPRNG)

According to Rose et al. [RGX11], a CSPRNG is seeded with unpredictable inputs in a secure way so that it is unfeasible to distinguish its output from a sequence of random bits. As defined herein, a CSPRNG has all properties of a normal PRNG, and, in addition, at least two other properties. One of these properties, referred to as the *next bit test*, states that given a sequence of m bits generated from a generator, no feasible method can predict the $(m+1)_{th}$ bit with probability significantly higher than one half. The second property, referred to a *malicious seeding resistance*, states that even if an attack can gain full or partial control of the inputs to the CSPRNG for a period (time), it is still unfeasible to predict or reproduce any random output from the CSPRNG. A pseudo-random number generation scheme is relatively straightforward in a CSPRNG. It can be, for example, a block cipher running in counter mode or output feedback mode, a stream cipher using a seed as cipher key, or a nested structure of hashing. A complicated part in CSPRNG design is how to seed and reseed the CSPRNG. Reseeding is a process used to update the sequential logic of a CSPRNG, which has been previously seeded, with a new seed. Such reseeding makes it more difficult to break a deterministic number generation algorithm.

2.1.3.4 PRNG Examples

Good examples of PRNGs include the Mersenne Twister and the Xorshift generators:

- Mersenne Twister, developed in 1997 and published in 1998 by Matsumoto and Nishimura [MN98], is a very popular PRNG with a period of $2^{2199371}$ and 623-dimensional equidistribution up to 32-bit accuracy, while using a working area of only 624 words. In 2006, the WELL family of generators was developed. The WELL generators, in some ways, improve the quality of the Mersenne Twister, which has a too-large state space and a very slow recovery from state spaces with many zeros [PLM06].
- Xorshift generators were introduced in 2003 by George Marsaglia [M$^+$03] and comprise a family of very fast generators based on a linear recurrence. A xorshift PRNG produces a sequence of $2^{32}-1$ integers x, or a sequence of $2^{64}-1$ pairs (x,y), or a sequence of $2^{96}-1$ triples (x,y,z), etc., by means of repeated use of a simple computer construction: XOR a computer word with a shifted version of itself. There are versions of xorshift which combine it with a non-linear function (e.g., xorshift+ or xorshift*) and pass some strong statistical tests. Another successor of Xorshift is the xoroshiro128+.

2.1.3.5 CSPRNG Examples

There are a number of standardized CSPRNG designs, such as FIPS 186-2, ANSI X9.17-1985, ANSI X9.31-1998 and ANSI X9.62-1998. Unfortunately, many of these designs are not satisfactory under certain circumstances. For example, two design flaws of the ANSI X9.17 PRNG have been identified by J. Kelsey et al. at Fast Software Encryption, 5th International Workshop Proceedings, Springer-Verlag, 1998 [RGX11].
The CSPRNGs that deserve special attention in this section are the following:

- Both Yarrow and Fortuna CSPRNGs [KSF99] have reseeding controls with support from complicated schemes for entropy accumulation. Yarrow does not specify a concrete method to evaluate entropy for reseeding while Fortuna reseeds the system periodically when the fastest entropy pool source is ready. Both of them use block ciphers in counter mode for pseudo-random number generation and use hash algorithms extensively for reseeding.

Use of block ciphers in counter mode and hash algorithms for reseeding is computationally expensive and time-consuming [RGX11]. So Fortuna is a Yarrow refinement. There are four major components in Yarrows: 1. An Entropy Accumulator which collects samples from entropy sources, and collects them in the two pools; 2. A Reseed Mechanism which periodically reseeds the key with new entropy from the pools; 3. A Generation Mechanism which generates pseudo-random outputs from the key; 4. A Reseed control that determines when a reseed is to be performed [KSF99].

- The Blum–Micali algorithm is a CSPRNG presented in 1984 by Blum and Micali [BM84]. Blum-Micali gets its security from the difficulty of computing discrete logarithms.
- ISAAC - Indirection, Shift, Accumulate, Add And Count (ISAAC) is a well known CSPRNG, and a stream cipher designed by Robert J. Jenkins Jr. in 1993 [Pud01]. ISAAC shares some similarities with RC4.
- Finally, Salsa20 and ChaCha20 (the later is a variation of the first), as presented above, are both secure stream ciphers and may be used as CSPRNG by using their outputs as pseudo-random numbers.

2.1.4 Statistical Tests and TestU01 Library

Rukhin et al. [RSN+01] shows that various statistical tests can be applied to a PRNG sequence to compare and evaluate it to a truly random sequence. Randomness is a probabilistic property; that is, the properties of a random sequence can be characterized and described in terms of probability. The likely outcome of statistical tests, when applied to a truly random sequence, is known *a priori* and can be described in probabilistic terms. There are an infinite number of possible statistical tests, each assessing the presence or absence of a *pattern* which, if detected, would indicate that the sequence is nonrandom. Because there are so many tests for judging whether a sequence is random or not, no specific finite set of tests is deemed *complete*. In addition, the results of statistical testing must be interpreted with some care and caution to avoid incorrect conclusions about a specific generator. In a nutshell, within the context of PRNG, the purpose of a statistical test is to conclude about the matching of an theoretical hypothesis against estimations conducted over empirical data.

The **TestU01 Library** [LS07] is presented by Pierre L'Ecuyer and Richard Simard of the Université de Montréal, as a software library in ANSI C for empirical testing of random number generators. It provides general implementations of the classical statistical tests for random number generators, as well as several others proposed in the literature, and some original ones. These tests can be applied to the generators predefined in the library and to user-defined generators. Specific test suites for either sequences of uniform random numbers in [0, 1] or bit sequences are also available. Basic tools for plotting vectors of points produced by generators are provided as well. Additional software enables one to perform systematic studies of the interaction between a specific test and the structure of the point sets produced by a given family of RNG. That is, for a given kind of test and a given class of RNG, to determine how large should be the sample size of the test, as a function of the generator's period length, before the generator starts to fail the test systematically.

The software tools of TestU01 are organized in four classes of modules: those implementing RNG, those implementing statistical tests, those implementing predefined batteries of tests,

and those implementing tools for applying tests to entire families of generators [LS07]. The statistical tests are implemented in the s_modules, whose names start with `s`. They all test one of the two null hypotheses H_0 defined previously, using different test statistics. To apply a test to a specific generator, the generator must, first, be created by the appropriate `Create` function in a `u` module, then it must be passed as a parameter to the function implementing the appropriate test. The test results are printed automatically to the standard output, with a level of detail that can be selected by the user [LS07].

Batteries of tests are predefined suites with a number of standard statistical tests, with fixed parameters, that can be applied to a given RNG. Different types of tests should be included in such a battery, in order to detect different types of weaknesses in a given generator. A number of predefined batteries of tests, some oriented towards sequences of uniform floating-point numbers in the interval [0, 1], others towards sequences of bits, are available in TestU01. There are small batteries, that run quickly, and larger (more stringent) batteries that take longer to run. These batteries are implemented in the `b` modules [LS07].

2.1.5 Attack Models

The most universally accepted assumption in cryptography is that the attacker (an enemy cryptanalyst) has full access to the ciphertext transmitted over the insecure channel [Lai92], along with details about the utilized cipher.

2.1.5.1 Basic Attack Models

According to Lai [Lai92], Kerckhoffs assumption implies that the security of a secret-key cipher system rests entirely on the secret key. Under Kerckhoffs assumption, attacks are usually classified according to the knowledge of the cryptanalyst as follows:

- Ciphertext Only Attack (COA) — model in which it is assumed that the enemy cryptanalyst has no additional information to intercepted ciphertexts;
- Known Plaintext Attack (KPA) — model in which the enemy cryptanalyst knows some plaintext/ciphertext pairs for the current key;
- Chosen Plaintext Attack (CPA) — model in which the enemy cryptanalyst can obtain the ciphertexts for any specified plaintexts for the current key. CPA is more powerful than the above attacks. If a cipher is secure against CPA, then it is also secure against the other two attacks. In practice, one would like to use a cipher that is secure against a CPA even if the enemy cryptanalyst would rarely have the chance to mount more than a COA. For a block cipher used in a *non-one-time* mode, i.e., when one key is used to encrypt many plaintext blocks (without any nonce), perfect-secrecy as defined by Shannon can never be achieved because identical ciphertext blocks imply identical plaintext blocks. Thus, although this cipher is unconditionally secure against a COA, it can be easily broken in a KPA if the secret key is used more than once.
- Chosen Ciphertext Attack (CCA) — CCA is an attack model where the adversary has access to a *decryption oracle* only prior to obtaining the target ciphertext, and the goal of the adversary is to obtain partial information about the encrypted message or notable details about the functioning of the cipher [CS98].

- Adaptive Chosen Ciphertext Attack (CCA2) — a cryptosystem secure against CCA2 is a very powerful cryptographic primitive. It is essential in designing protocols that are secure against active adversaries. In this attack, an adversary can inject messages into a network. These messages can be ciphertexts, and the adversary may be able to extract partial information about the corresponding cleartexts through its interactions with the parties in the network. The adversary can obtain decryptions of its choice from the *oracle*, with the option of submitting any other ciphertext, including ciphertexts that are related to the target ciphertext [CS98].

The previous models are, often, described and used to prove security from a theoretical perspective. Nonetheless, it is also common nowadays to mention attacks that might not appear in the typical cryptanalyst list, due to the many advances and environments of the technology and usages of cryptographic primitive: side-channel attacks are amongst them. According to Cherednichenko et al. [CBM13], a side-channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms. For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information which can be exploited to break the system. Some side-channel attacks require technical knowledge of internal system operation on which the cryptography is based. Differential power analysis is effective as black-box attack. The most powerful side channel attacks are also based on statistical methods pioneered by Paul Kocher. General classes of side-channel attack include:

- Timing attack — attacks based on measuring time spent on various computations;
- Power monitoring attack — attacks which make use of power consumption variations by the hardware during computation;
- Electromagnetic attacks — attacks based on leaked electromagnetic radiation which can directly provide plaintexts and other information. Such measurements can be used to infer cryptographic keys using techniques equivalent in power analysis, or can be used in non-cryptographic attacks, e.g. TEMPEST (a.k.a. van Eck phreaking or radiation monitoring) attacks;
- Acoustic cryptanalysis — attacks which exploit sound produced during computation (rather than power analysis);
- Differential fault analysis — attacks in which secrets are discovered by introducing faults in computation;
- Data reminiscence — sensitive data are read after supposedly having been deleted.

Finally, brute force attacks are typically mentioned as one of the basic attacks to ciphers, but they are not really part of cryptanalysis. A practical block cipher is generally considered secure if none of the known attacks can do much better than the exhaustive key search attack (brute force attack). In an exhaustive key search ciphertext only attack on a block cipher, each of the possible keys is tried in turn to decrypt one (or more) intercepted ciphertext block(s) until one key results in readable plaintext block(s). The complexity of this attack can be described as the number of tried decryptions, which is, on average 2^{k_t-1} for a block cipher with true key size k_t. The true key size must be large enough to make the exhaustive key search attack unfeasible.

2.1.5.2 Differential Attacks

Differential Cryptanalysis was presented by Biham and Shamir in [BS91] as a method which analyses the effect of particular differences in plaintext pairs on the differences of the resulting ciphertext pairs. These differences can be used to assign probabilities to the possible keys and to locate the most probable key. This method usually works on many pairs of plaintexts with the same particular difference using only the resulting ciphertext pairs. E.g., for DES-like cryptosystems the difference is chosen as a fixed XORed value of the two plaintexts.

Associated with any pair of encryptions are the XOR value of its two plaintexts, the XOR of its ciphertexts, the XORs of the inputs of each round in the two executions, and the XORs of the outputs of each round in the two executions. These XOR values form an n-round characteristic. A characteristic has a probability, which is the probability that a random pair with the chosen plaintext XOR has the round and ciphertext XORs specified in the characteristic. Let us denote the plaintexts XOR of a characteristic by Ω_P and its ciphertexts XOR by Ω_T. The example in the Figure 2.6 illustrates a one-round characteristic with probability 1, in which the input of the round gives rise to exactly the same output when the right side half of the input is 0s. This characteristic was very useful in the analysis of any DES-like cryptosystem.

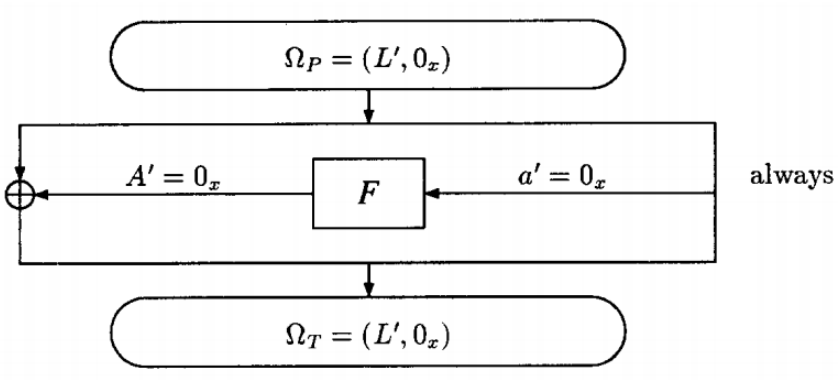

Figure 2.6: A one-round characteristic with probability 1 is (for any L') for DES-like cryptosystems.

Biham and Shamir define formally an n-round characteristic as a tuple $\Omega = (\Omega_P, \Omega_\Lambda, \Omega_T)$ where Ω_P and Ω_T are m-bit numbers and Ω_Λ is a list of n elements $\Omega_\Lambda = (\Lambda_1, \Lambda_2, ..., \Lambda_n)$, each of which is a pair of the form $\Lambda_i = (\lambda_I^i, \lambda_O^i)$ where λ_I^i and λ_O^i are $(m/2)$-bit numbers and m is the block size of the cryptosystem. A characteristic satisfies the following requirements:
λ_I^1 = the right half of Ω_P;
λ_I^2 = the left half of $\Omega_P \oplus \lambda_O^1$;
λ_I^n = the right half of Ω_T;
λ_I^{n-1} = the left half of $\Omega_T \oplus \lambda_O^n$;
and, for every i such that $2 \leq i \leq n-1$, $\lambda_O^i = \lambda_I^{i-1} \oplus \lambda_I^{i+1}$.
An n-round characteristic Ω has probability p^Ω if p^Ω is the product of the probabilities of its n rounds (which is probability of a pair to be a right pair):

$$p^\Omega = \prod_{i=1}^{n} p_i^\Omega \text{.Where } \lambda_I^i \rightarrow \lambda_o^i, \text{with probabilitity } p_i^\Omega.$$

For practical purposes, the significant probability with respect to a characteristic is the probability that a pair whose plaintext XOR equals the one of the right plaintexts pair XOR for a fixed

key (the one we try to find). This probability is not constant for all the keys. However, Biham and Shamir assume that the characteristic probability is a very good approximation of it, which is usually the case.

Differential cryptanalysis was used to show how DES could be broken and is considered a very powerful tool to understand the functioning of ciphers. Thought this tool was studied in depth during the course of this project, it was not possible to conduct an extensive differential analysis to the cryptosystem proposed herein yet. The next chapter contains nonetheless the intuition on how the analysis can be tackled and on how the proposed cryptosystem is resistant to this kind of attack.

2.1.5.3 Linear Attacks

Linear Cryptanalysis is a kind of KPA presented by Matsui [Mat93] as a method for cryptanalysis for DES-like ciphers. With this attack, the 8-round DES is breakable with 2^{21} known-plaintexts; the 12-round DES is breakable with 2^{33} known-plaintexts; and a 16-round DES is breakable with 2^{47} known-plaintexts, still faster than an exhaustive search for its 56 key bits.
The purpose of Linear Cryptanalysis is to find the following *effective* linear expression for a given cipher algorithm:

$$P[i_1, i_2, ..., i_a] \oplus C[j_1, j_2, ..., j_b] = K[k_1, k_2, ..., k_c], \quad \textbf{(2.1)}$$

where $i_1, i_2, ..., i_a, j_1, j_2, ..., j_b, k_1, k_2, ..., k_c$ denote fixed bit locations and $[a, b]$ denote the XOR between the inputs. Being *effective* means that equation (2.1) holds with probability $p \neq 1/2$ for randomly given plaintext P and the corresponding ciphertext C. The magnitude of $p - 1/2$ represents the effectiveness of equation (2.1). Once one succeeds in reaching an effective linear expression, it is possible to determine one key bit $K[k_1, k_2, ..., k_c]$ via the following algorithm based on the maximum likelihood method (**method 1**):

- Step1 $-$ Let T be the number of plaintexts such that the left side of equation (2.1) is equal to zero.
- Step2 $-$ If $T > N/2$ (N denotes the number of plaintexts),
  then guess $K[k_1, k_2, ..., k_c]$ = 0 (when $p > 1/2$) or 1 (when $p < 1/2$),
  else guess $K[k_1, k_2, ..., k_c]$ = 1 (when $p > 1/2$) or 0 (when $p < 1/2$).
  The success rate of this method clearly increases when N or $|p - 1/2|$ increases. We, now, refer to the most effective linear expression (i.e., to when $|p - 1/2|$ is maximal) as the best case and the probability p as the best probability. In this case, the main concern are the following:
  - P1 $-$ How to find effective linear expressions;
  - P2 $-$ An explicit description of the success rate by N and p;
  - P3 $-$ A search for the best expression and a calculation of the best probability.

For a practical KPA of an n-round cipher, the best expression of an $(n-1)$-round cipher is used; that is to say, assuming the final round as having been deciphered using K_n, it is acceptable to have a term of the F-function in the linear expression. Consequently, for ciphers using a Feistel network, the following type of expression is obtained:

$$P[i_1, i_2, ..., i_a] \oplus C[j_1, j_2, ..., j_b] \oplus F_n(C_L, K_n)[l_1, l_2, ..., l_d] = K[k_1, k_2, ..., k_c]. \quad \textbf{(2.2)}$$

If one substitutes an incorrect candidate for K in equation (2.2), the effectiveness of this equation clearly decreases. Therefore, the following maximum likelihood method can be applied to deduce K_n and $K[k_1, k_2, ..., k_c]$ (**method 2**):

- Step1 − For each candidate $K_n^{(i)}(i = 1, 2, ...)$ of K_n, let T_i be the number of plaintexts such that the left side of equation 2.2 is equal to zero.
- Step2 − Let T_{max} be the maximal value and T_{min} be the minimal value of all T_is;
  - If $|T_{max} - N/2| > |T_{min} - N/2|$, then adopt the key candidate corresponding to T_{max} and guess $K[k_1, k_2, ..., k_c]$ = 0 (when $p > 1/2$) or 1 (when $p < 1/2$);
  - If $|T_{max} - N/2| < |T_{min} - N/2|$, then adopt the key candidate corresponding to T_{min} and guess $K[k_1, k_2, ..., k_c]$ = 1 (when $p > 1/2$) or 0 (when $p < 1/2$).

In order to perform the attack on DES, Matsui [Mat93] performs, firstly, linear approximation of S-boxes investigating the probability that a value on an input bit coincides with a value on an output bit. More generally, it is useful to deal with not only one bit position but also a XORed value of several bit positions. With this approach, he was able to obtain the space search reduction mentioned at the beginning of this section.

2.1.5.4 Differential Linear Cryptanalysis

Differential-linear cryptanalysis is a CPA on iterated cryptosystems presented by Langford and E Hellman [LH94]. Based on both differential and linear cryptanalysis, this approach enabled recovering 10 bits of the key with 80% probability of success using only 512 chosen plaintexts for an 8-round DES. The probability of success would increase to 95% using 768 chosen plaintexts. A larger part of the key can be recovered with a smaller probability of success. While comparable in speed to existing attacks, this 8-round attack represents an order of magnitude improvement in the amount of required plaintexts.

2.2 Classical Hill Cipher

Hill Cipher (HC) is nowadays considered one of the classical examples of cryptosystems. This substitution cipher was defined in 1929 by Lester S. Hill [Hil29], to who owns its name but, interestingly, it has been the subject of several recent publications, which aim to solve its weaknesses. Compared to modern substitution ciphers, the encryption and decryption algorithms are simpler, but the construction *per se* has some interesting properties, discussed below.

HC can be defined as follows. Let $\mathcal{M}$ and $\mathcal{C}$ denote the messages and ciphertexts spaces, respectively, written in an alphabet of $n \in \mathbb{N}$ letters, and consider that $\mathcal{S}_k$ symbolizes the set of all squared $k \times k$ matrices ($k \in \mathbb{N}$), which are nonsingular modulo n. Herein, this set is called the set of *key matrices*. Ideally, when encrypting a message m, the key matrix should be randomly chosen from $\mathcal{S}_k$, with k coinciding with the size of m. The last prerequisite guarantees perfect secrecy, but it is impractical since the key is n times larger than the message. Alternatively, the cipher can be used as a block cipher, in which the size of the block is predefined and known. When encrypting, m needs to be padded up to a multiple of k, which defines the block size, and divided into blocks $m[0], m[1], ..., m[l]$. The encryption algorithm consists on multiplying the randomly selected key matrix $S \in \mathcal{S}_k$ by each one of the blocks using matrix multiplication

modulo n. Perfect secrecy would also be achieved if a key matrix was generated for each one of the blocks, but that would also be less practical than using the *one time pad*. A more practical implementation requires the same key matrix to be used to encrypt arbitrarily large messages or, at least to describe an efficient and secure means to change the key matrix without having to transmit the resulting matrices from the encrypting to the decrypting part. Resorting to Alice (A) and Bob (B) as the encrypting and decrypting parts, respectively, classical HC encryption can be formalized as follows:

1. A: $S \xleftarrow{r} \mathcal{S}_k$
   Alice randomly chooses a key matrix from $\mathcal{A}_k$;
2. A $\rightarrow$ B: S
   Alice sends the key to Bob using a secure channel;
3. A: $c[i] = S \times m[i] (\mathrm{mod}\ n)$, $\forall i = 0, ..., l$
   Alice multiplies each block of the message by the key matrix (modulo n) to obtain c.

Notice that the previous procedure indicates the key matrix selection and transmission steps, though it might be assumed that the key was randomly chosen and exchanged using a secure channel previously to the encryption procedure. Additionally, the concern on generating non-singular matrices is also not depicted because it is implicit to the definition of $\mathcal{S}_k$. Decryption is similar to encryption, but requires Bob to first calculate the inverse key matrix modulo n before performing the matrix multiplication:

1. B: calculates S^{-1}, such that $S \times S^{-1} = I \bmod n$, where I denotes the identity matrix, i.e., Bob calculates the inverse of $S \bmod n$;
2. B: $m[i] = S^{-1} \times c[i] (\mathrm{mod}\ n)$, $\forall i = 0, ..., l$
   Bob multiplies each block of the ciphertext by the inverse of the key matrix (modulo n) to obtain m.

HC exhibits some interesting properties, which justify some of the attention it received along the years. For example, it implements the diffusion and confusion concepts, that Shannon [Sha01] identified as essential properties for ciphers, even though his theory was published later than the proposal of the cipher. The major flaw of this cipher is that it is vulnerable to KPA, since given the knowledge of enough plaintext/ciphertext pairs (or enough message/ciphertext block pairs), it is possible to construct a system of linear equations that enables the recovery of the key matrix.

2.3 Existing Hill Cipher Modifications

Some of the modifications or improvement (attempts) on HC, proposed in the specialized literature, are reviewed in this section.

2.3.1 First Attempts on Correcting the Known Plaintext Attack

Several attempts to improve HC were made in past. Most of the improvements use small subsets of S_k ($k \times k$ matrices, nonsingular modulo n), usually related or generated from a master key matrix, and encrypt each plaintext with different key matrix from this subset.

Saeednia [Sae00] uses randomized permutations t of rows and columns of a master key matrix K to produce different matrices. The used permutation is transferred as HC-encrypted ($u = Kt$), which makes this method vulnerable to KPA, according to the proof presented by Li et al. [LLL04].

Chefranov et al. [CEOP08] try to correct the aforementioned issue using a PRNG for generating the permutation t as t =PRNG(seed, s), where the $seed$ is a shared secret and s is transferred as $s \oplus seed$. In both cases, the size of the subset of used matrices is $k!$.
In order to enlarge the previous key space, one can use Maximum Distance Separable (MDS) master key matrices [MKK12] or generalized permutation matrices, in which each row and column has only one special non-zero element, co-prime with n [MC12]. In the second case, the size of the subset of used matrices is $k!mo^k$, where $mo = 2^n - 2$ is the maximal order of the set of units Z_n^x.

Another method that did not survive to KPA [LZC08] [RRVGMM+08] is given by Ismail et al. [IAD06] for image encryption. It changes the matrix for each plaintext row by row, by multiplying it with a given IV. This scheme has an additional flaw also: it does not guarantee that the produced matrices are nonsingular.

2.3.2 Iterated Hill Cipher

Another group of improvements can be named as iterated HC, because they use rounds. Sastry et al. [SMB09] created three-staged rounds, each consisting of left and right multiplications by the key matrix K, application of a bitwise permutation and XORing with K. Keliher [Kel10] showed that this scheme is vulnerable to both KPA and CPA. The vulnerability to KPA/CPA [KT13] also holds for modifications presented by Sastry and Shank [SS07] and Sastry et al. [SSB10].

2.3.3 Hash Hill Cipher

Toorani and Falahati introduced two variants of HC [TF09, TF11] which use hash functions to modify the encryption process for each plaintext. Both variants are Affine HC, where the new matrix K and vector V are derived for each plaintext, using pseudo-random numbers generated by a one-way hash function [TF09] or a keyed hash function [TF11] from one starting random integer a_0. The value of a_0 is passed indirectly with the ciphertext. This idea comes originally from Lin et al. [LLL04], where a_0 is transferred in clear form. Keliher and Delaney [KD13] presented a CPA on these schemes, which bypasses the hash function.

2.3.4 Randomized Hill Cipher

On the subject of the generation of nonsingular matrices, the works from Dana Randall [Ran93] and Tran and Nguyen [TN08] need to be emphasized. Tran presented a theorem and some propositions for choosing a modularly nonsingular matrix.
Dana Randall presented an algorithm for generating $n \times n$ nonsingular matrices uniformly over a finite field which runs in expected time $M(n) + O(n^2)$ over $GF(2)$, where $M(n)$ is the time needed to multiply two $n \times n$ matrices and the expected number of random bits it uses is $n^2 + 3$. An implementation made in the scope of the work presented herein shows that the nonsingular matrices outputted by the algorithm developed by Dana Randall are always upper triangular-like matrices, which negatively impacts its usefulness within the scope of a HC. Other disadvantage

of this method concerns the fact that not all matrices generated with the method are modularly nonsingular (all are nonsingular, but not all modulo n). This is the only other algorithm we found in literature to generate invertible matrix without having to use another master matrix. Nonetheless, it should be mentioned that this algorithm was not designed to be used in ciphers specifically. It is a general method to generate nonsingular matrices and it works better when n is a big number, so it can have some entropy, but not quite enough.

Acharya [ARPP07] proposed a self-invertible matrix generation method, in which the matrix used for the encryption is its own inverse. At the time of decryption, it is not necessary to find the inverse of the matrix. In order to generate a different key matrix each time, the encryption algorithm randomly generates the seed number and, from this, the key matrix is generated. The key matrix is generated via the formula $K_{12} = (seednumber \times m) \ mod \ n$; $K_{ij} = (K_{i(j-1)} \times m) \ mod \ n$; $K_{i1} = (K_{(i-1)m} \times m) \ mod \ n$. One year later, the same authors [APP08] tried to improve the method in a novel cryptosystem which uses a randomly generated self-invertible matrix as an encryption key for each block of plaintext. Another variant [AJPP09] uses Permutation Matrix Generation, which makes use of *random* permutations of columns and rows of a matrix to form a *different* key for each data encryption, by performing the operation PkP, where P is a permutation matrix and k is an involutory key matrix.

Different nonsingular matrices can be obtained statically, as shown by Mahmoud and Chefranov [MC12], derived from other matrices, as they show in [MC09], or using pseudo-randomly generated eigenvalues as they propose in [MC14] for matrix exponentiation to a pseudo-random power generated by a PRNG, for each plaintext block. The size of the subset of used matrices is at most $\varphi(n)k!$ in the first case, and $min(\varphi(n)^k, \frac{Period(PRNG)}{k})$ in the second case. The designers claim this method to be more efficient and secure than other similar methods.

Another iteration of HC, termed Hill++ and based on the Affine HC, was proposed by Rahman et al. [Rah13]. In Hill++, the ciphertext is obtained from $C = PK + V(\mathrm{mod}\ m)$, where V represents a constant in form of matrix. Hill++ introduces a random matrix key which is computed based on the previous ciphertext blocks and a multiplying factor in order to increase the resistance of the algorithm to the KPA. Hill++ also implements an involutory key generation algorithm where the same key matrix can be used for both encryption and decryption as proposed by Acharya et al. [ARPP07]. In Hill++, the key depends on three parameters (α, β and γ) shared between the sender and receiver. RMK is a random involution $n \times n$ key matrix generated for each block using the parameters above. $\alpha = 1^{st}$ seed number; $\beta = 2^{nd}$ seed number; γ = multiplying factor; where $\alpha_i, \beta_i, \gamma_i \in Z_p$ and $i \ mod \ 2 = 0$.

2.4 Conclusions

This chapter presented all of the main cryptography-related concepts required to understand the remaining part of the document. It included also a greater insight to the main motivation behind this work. From these, and the different works already developed with the intent of improving HC, which include multiple matrix multiplications, generation of the key matrices by trial and error methods, or the use of self-invertible matrices, it is possible to iterate and develop the work that this document presents. The next chapter will define the different developed algorithms, the logic behind them, and the analysis done to them to ensure their

correctness.

In an intermediate part of the chapter, several attacks and attack models typically employed to prove the security of cryptosystems were discussed. Thought these tools were studied in depth along the project, there was no time to include an extensive study for some of them, specially differential and linear cryptanalysis, since these are typically very demanding tasks. The next chapter contains a discussion regarding those attacks and, for the most demanding ones, the intuition on how the analysis can be tackled and on how the proposed system is resistant on those cases.

Chapter 3

Randomized Key Hill Cipher (RKHC)

3.1 Introduction

The main contribution of this dissertation consists on proposing a more secure variant of HC, herein named RKHC, by exploiting the usage of a CSPRNG to generate a different key matrix for each plaintext block submitted to encryption/decryption. The proposed algorithm for generating nonsingular matrices modulo 2 is based on the Gauss-Jordan elimination procedure. It is a simple algorithm, but no previous work has been found using such approach, neither another efficient method to do this.

The modifications and algorithms described in chapter 2 address three fundamental problems of the classical HC, namely (i) the susceptibility to KPA; (ii) the size (and transmission) of the key; and (iii) the predictability of its outputs to specific messages. Regarding the problem mentioned in last, it should be noted that, for example, the classical form of HC outputs a block of zeros for the input blocks formed entirely by zeros as well. To tackle the first and second problems, algorithms to efficiently and securely generate pseudo-random nonsingular matrices modulo 2 and their inverses for each block of the message are proposed herein. To address the third problem, an additional step is included in the encryption and decryption procedures. These procedures are described in the next subsection, while the aforementioned algorithms are described in the third subsection.
Notice that it is assumed that the number of letters of the alphabet used is two (*i.e.*, $n = 2$). This also means that arithmetic operations are always performed modulo 2. Sometimes, *XOR* ($\oplus$) is used to refer to sum modulo 2 and each letter of the message referred to as *bit*.

3.2 Encryption and Decryption Key Matrix Generation

The size of the key matrices used to encrypt and decrypt will define the size of the blocks. Matrices with sizes 8×8, 16×16, 32×32 and 64×64 were tested during this project, corresponding to 64, 256, 1024 and 4096 bits, respectively; in these cases, the block size of the messages must be 8, 16, 32 and 64 bits, respectively. The algorithm proposed for generating nonsingular matrices modulo 2 is based on the Gauss-Jordan elimination procedure. Algorithm 1 outputs the encryption key matrix (a nonsingular matrix). Algorithm 2 is used to invert the key matrix. Algorithm 3 generates the inverse matrix directly through the same method used in Algorithm 1, but in the reverse order. Algorithm 3 is, thus, a shortcut algorithm that can be used directly to obtain the inverse matrix instead of combining Algorithm 1 and Algorithm 2. Notice that it is assumed that both Alice and Bob use the same generation algorithm $G(seed)$ initialized with the same seed (which is actually the cryptographic secret of this system) to generate equal matrices in both sides. In the explanation below, consider that the first position in a row or column is 0 and $n-1$ corresponds the last position.

Algorithm 1 Encryption Key Matrix Generation Algorithm.

GenMatrix (n, seed)
begin
Let I be a random upper triangular matrix with 1s at the main diagonal
Use each element on the main diagonal from the bottom to the top to alter the ones below
for all $0 \leq i < n-1$ **do**
  $r \leftarrow G(seed)$, i.e., assign a pseudo-random (n-1)-i bits sequence to r
  for all $i < j < n$ **do**
    if $(r_j bit = 1)$ **then**
      I_j = $I_j \bigoplus I_i$, i.e., XOR line j with line i in case the random bit is 1
choose the randomness of I_0 element that was not XORed
$r \leftarrow G(seed)$, let r be a number $0 \leq r < n$
Switch I_0 with I_r, i.e., change the 0_{th} with the r_{th} row of I
return (I)

Notice that Algorithm 1 defines $bitsNumber = [\sum_{i=1}^{n-1}(n-i)] \times 2 + log_2(n)$ as the number of bits to be generated per each matrix; $56 + log_2(8)$ for 8×8; $240 + log_2(16)$ for 16×16; $992 + log_2(32)$ for 32×32 and $4032 + log_2(64)$ for 64×64 matrices. $log_2(n)$ is the number of bits needed to generate r and may be rounded for excess. This last operation (generation of r) corresponds to the inverse first step of a Gauss Elimination process, in which one replaces the first row of the matrix when it starts with a `0` with a row starting with a number different from 0 (in this case `1`).

According to a theorem presented by Eves [Eve80], if a square matrix A is reducible to an identity of order n, $I_{(n)}$, by a sequence of elementary (linear) row operations, then the same sequence of elementary row operations performed on $I_{(n)}$ produces A^{-1} and this is precisely what is being exploited to obtain the generation algorithm. Thus, in Algorithm 1 we start with the identity matrix $I_{(n)}$, which is invertible. The first elementary row operation produces an elementary matrix which is nonsigular as well, such as the remaining elementary operations until the end of the algorithm, so all the sub-matrices generated along the algorithm are nonsingular, supported by the theorem presented above. Notice that Algorithm 1 needs at most $\sum_{i=1}^{n-1}(n-i)$ elementary row operations (XOR in this case), corresponding to the number of generated bits. The elementary operations on the upper triangle of the matrix are implicit (i.e., one just generates `0`s and `1`s and place them there directly) in order to get better performance.

Consider that rp denotes the number of row operations. The time complexity of Algorithm 1 (in the worst case) is $T(rp) = O(rp) + n \times T(\text{CSPRNG})$, where T (CSPRNG) is the complexity of the CSPRNG used to generate the pseudo-random bits.

For example, let us generate one 4×4 nonsingular matrix, using an hypothetical PRNG, supposing it generates a sequence of interleaved `1`s and `0`s (i.e., $10101010...$):

- Step 1 − Get a pseudo-random upper triangular matrix with 1 at the main diagonal, using the selected generator (notice the bits are used from right to the left):

$$M = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

- Step 2 — For the first row we generate a sequence of 3 bits = 101, which means that the algorithm will XOR the first row with the second row (bit = 1); not XOR with the third row (bit = 0) and, then, it will XOR with the fourth row (bit = 1), resulting in the following matrix:

$$M = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \quad (3.2)$$

- Step 3 — For the second row we need a sequence of 2 bits = 01, meaning, in this case, that the algorithm will not XOR the second row with the third row (bit = 0), but will XOR with the fourth row (bit = 1), resulting in the following matrix:

$$M = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (3.3)$$

- Step 4 — For the third row, only one bit = 0 is needed, meaning that the algorithm will not XOR the third row with the fourth row (bit = 0), ending up with the following matrix:

$$M = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (3.4)$$

- Step 5 — Let r be a pseudo-random bit sequence such that $0 \leq r < 4$ in decimal (4 is the number of rows). Using our hypothetical PRNG, we need $log_2(4) = 2$ bits, thus $r = 10$, which is 2 in decimal; then we change the row 0 (first) with the row 2 (third row). The resulting nonsingular matrix will be:

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (3.5)$$

Algorithm 2 Algorithm for Inverting the Key Matrix.

InvertMatrix (M, n)
begin
for all $0 \leq i < n - 1$ **do**
  Assure there is a 1 in the main diagonal
  if ($M_{i,i} \neq 1$) **then**
    for all $i < j < n$ **do**
      if ($M_{j,i} = 1$) **then**
        change the row M_j with M_i
        change the row I_j with I_i
        break for
  Once the element in the main diagonal is 1, clean the bits below
  for all $i < j < n$ **do**
    if ($M_{j,i} = 1$) **then**
      $M_j = M_j \bigoplus M_i$, i.e., XOR line j with line i in case the bit $M_{j,i} = 1$
      $I_j = I_j \bigoplus I_i$, i.e., XOR line j with line i in case the bit $M_{j,i} = 1$
Once the element in the main diagonal is 1, clean the bits above
for all $n > i > 0$ **do**
  for all $i > j \geq 0$ **do**
    if ($M_{i,j} = 1$) **then**
      $M_j = M_j \bigoplus M_i$, i.e., XOR line j with line i in case the bit $M_{j,i} = 1$
      $I_j = I_j \bigoplus I_i$, i.e., XOR line j with line i in case the bit $M_{j,i} = 1$
return (I)

Considering the matrix generated above as an example to explain Algorithm 1. This matrix will, now, be inverted following the steps of Algorithm 2. The explanation will, thus, start from the following 2 matrices (and operations are being performed in both M and I matrices in this example):

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

The steps are as follows:

- Step 1 − as $M_{0,0} \neq 1$, the first and the second rows are exchanged to comply with that requirement:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} I = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7)$$

- Step 2 − As $M_{0,0} = 1$, the algorithm performs the XOR of the first row with the following where there is a 1 in the first column, resulting in:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} I = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.8)$$

- Step 3 − as $M_{1,1} \neq 1$, the second and the third rows are exchanged to comply with that

requirement:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} I = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.9)$$

- Step 4 – As $M_{1,1} = 1$, the algorithm performs the XOR of the second row with the following where there is a `1` in the second column, resulting in:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} I = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad (3.10)$$

- Step 5 – As soon as the performed operations transform M into an upper triangular matrix, we perform the operations upwards. Since $M_{3,3} = 1$, the algorithm proceeds with the XOR of the last row with the rows containing a 1 on M at the corresponding column), and so on. This will eventually lead to M becoming the identity and I becomes the inverted matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} I = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad (3.11)$$

An encryption and decryption operation for the classical HC using these generated matrices can, now, be performed, for exemplification purposes. Consider that one want to encrypt the plaintext $P = [1110]$. Encryption corresponds to the following matrices multiplication, resulting in ciphertext $C = [0111]$:

$$[1110] \times \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} = [0111]. \quad (3.12)$$

Decryption of $C = [0111]$ is performed using the inverted matrix, obtaining the original plaintext:

$$[0111] \times \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} = [1110]. \quad (3.13)$$

Algorithm 3 Algorithm for Directly Generating the Inverted Key Matrix.

InvertMatrix (n, seed)
begin
Generate all random bits used in the encryption key matrix generation (Algorithm 1), and put them into an auxiliary matrix M
Let I be the Identity on which working in reverse order
Let $usedBits = (bitsNumber - 512 \times integerOf(bitsNumber/512)) - log_2(n)$
Let $matrixPosition = integerOf(bitsNumber/512)$
$r \leftarrow G(seed)$ Let r be a sequence of $log_2(n)$ bits from M in the matrixPosition
Change the 0_{th} row to the r_{th} row
Get a matrix with a random lower triangle
for all $0 \leq i < (n-1)$ **do**
  if $(usedBits - (n-(i+1)) < 0)$ **then**
    $matrixPosition = matrixPosition - 1;$
    $usedBits = 512 + (usedBits - (n-(i+1))$
  else
    $usedBits = usedBits - (n-(i+1))$
  $r \leftarrow G(seed)$ Let r be a sequence of $n-(i+1)$ bits from M in the matrixPosition
  for all $(n-2-i) \geq j \geq 0$ **do**
    if $(r_j bit = 1)$ **then**
      $I_{(n-1-j)} = I_{(n-1-j)} \bigoplus I_i$
Get a matrix with a random upper triangle
for all $(n-2) \geq i \geq 0$ **do**
  if $(usedBits - (n-(i+1)) < 0)$ **then**
    $matrixPosition = matrixPosition - 1$
    $usedBits = 512 + (usedBits - (n-(i+1))$
  else
    $usedBits = usedBits - (n-(i+1))$
  $r \leftarrow G(seed)$ Let r be a sequence of $n-(i+1)$ bits from M in the matrixPosition
  for all $0 \leq j < (n-1-i)$ **do**
    if $(r_j bit = 1)$ **then**
      $I_i = I_i \bigoplus I_{(n-1-j)}$
return (I)

Notice, in Algorithm 3, 512 is used as an implementation detail to save half of the bits for $32 \times 32 = 1024$. For $64 \times 64 = 4096$ that can be adapted as well.
In order to improve the performance, Algorithms 1 and 3 will have to be preferable implemented in order the r bits are generated outside the loops, since we already presented the exact number of bits to be generated by the deterministic CSPRNG given the seed.

3.3 Number of Generated Matrices

The number of nonsingular $n \times n$ matrices containing only 0s or 1s modulo 2 is given by $M_n = \prod_{i=0}^{n-1}(2^n - 2^i)$, defined by a general linear group of degree n [Wei55], which can be obtained by counting the number of possible independent rows (vectors) for a $n \times n$ matrix. This means that there will be 6 nonsingular 2×2 matrices, 168 nonsingular 3×3 matrices, etc. The proof that Algorithm 1 outputs all nonsingular matrices for a given order n can be achieved by induction as follows:

1. For $n = 2$, one can enumerate all $1 \times 2 + log_2(2)$ = 3 bits (2^3 possible matrices) easily by simulating Algorithm 1 by hand.

2. For the induction step, consider that $n = k$ and the algorithm, indeed, generates all

$\prod_{i=0}^{k-1}(2^n - 2^i)$ matrices. For $n = k+1$, Algorithm 1 will, then, need to generate an additional row and column. As for the row, it is known from the definition of the algorithm that all possible combinations of 2^{k-1} will be generated for that row (since the first value has to be 1). For each one of these combinations, $k-1$ random bits will be generated to decide if the first row will be XORed with the $k-1$ rows or not. All these $2^{k-1} \times 2^{k-1}$ combinations will give rise to different nonsingular matrices.
Finally, the last step of the algorithm consists in generating a number r between 0 and $k-1$ to decide if the first row is going to be shifted with row r, which will again produce different matrices half of the times (i.e., in the times that the row is interchanged with another one starting with a 0. The total number of outputs of the algorithm is given by:

$$M'_k = 2^{k-1} \times 2^{k-1} \times k \times M_{k-1}. \quad (3.14)$$

Notice that the actual number of nonsingular $k \times k$ matrices M_k given M_{k-1} is actually smaller than M'_k (notice the $1/2$ term in the following equation), given by

$$M_k = 2^{k-1} \times 2^{k-1} \times k/2 \times M_{k-1}, \quad (3.15)$$

because the final step (of choosing r) sometimes (half of the times, on average) changes row 1 with another row which already contains a 1 in the first position, being useless. This needs to be improved in future.
This proves that the entire space of matrices is actually outputted by the algorithm, one of its best advantages. ■

3.4 RKHC Encryption and Decryption

Encryption and decryption (Algorithm 4 and 5) in RKHC differs from the classical HC in two main aspects: (i) for each block of the message or ciphertext, a pseudo-random mask with k bits is generated for each block and XORed to it prior to its multiplication by the key matrix (during encryption) or after multiplying by the inverse of the matrix (during decryption); and (ii) a key matrix is generated for each plaintext or ciphertext block by an efficient algorithm. The generation of the pseudo-random mask and key matrix should use a CSPRNG each, denoted herein by $G_1(t)$ and $G_2(t)$, both defined from the set of $\{0,1\}^k$ to $\{0,1\}^N$, where $N >> k$, $N \in N$. The concatenation of the seeds, sk_1 and sk_2, used to initialize the generators, will be the actual symmetric key of the resulting cryptosystem. This key is herein denoted by $sk(= sk_1||sk_2)$ and it is assumed that it is established using a secure channel between Alice and Bob prior to encryption or decryption. The fact of using k as the size of the seeds is not that important, as long as sk is large enough to be secure. Below, the notation $G_1(sk_1)|_i$ is used to denote the sequence of k bits taken from the generated pseudo-random sequence, starting from bit $i \times k$. $Alg1|_i$ is used to denote iteration i of the Algorithm 1 used to output the encryption key matrices. The iteration number determines the block of bits that are taken from the pseudo-random sequence generated by $G_2(sk_2)$. The encryption procedure can, then, be formalized as presented in Algorithm 4.

Algorithm 4 RKHC Encryption.

RKHCEncrypt (plaintext, sk)
begin
Initialize $G()$ with sk_1
Apply padding and divide m into blocks $m_0, m_1, ..., m_l$
for $i = 0, 1, ..., l$ **do**
$r_i \leftarrow G(sk_1)|_i$, i.e., generate the next block of k bits from $G(sk_1)$ and assign it to r_i
$S_i \leftarrow Alg1|_i$, i.e., use ***Algorithm*** 1 to output a new key matrix S_i for the given block (notice that $sk2$ is used in $Alg1$).
Calculate $c_i = S_i \times (m_i \oplus r_i) \pmod 2$
$c = c_1||c_2||...||c_l$
return (c)
end

Notice that a single CSPRNG could be used instead of two, as long as the bits generated for the mask and for the construction of the key matrices were taken in the correct order for the encryption and decryption. In such case, a single key or seed would be involved. Nonetheless, using two CSPRNG eases the explanation. Decryption is similar to encryption, but uses the inverse modulo 2 of the key matrix used to encrypt, and the pseudo-random mask is applied after matrix multiplication. The algorithm for generating the i-th key matrix and inverting it is herein denoted by $Alg2|_i$ (iteration i of the Algorithm 2, alternatively the Algorithm 3 is used instead to output directly the decryption key matrices). The decryption procedure can be formalized as presented in Algorithm 5.

Algorithm 5 RKHC Decryption.

RKHCDencrypt (ciphertext, sk)
begin
Initialize $G()$ with sk_1
Divide c into blocks $c_0, c_1, ..., c_l$
for $i = 0, 1, ..., l$ **do**
$r_i \leftarrow G(sk_1)|_i$, i.e., generate the next block of k bits from $G(sk_1)$ and assign it to r_i
$S_i^{-1} \leftarrow Alg2_i$, i.e., use ***Algorithm*** 2 or 3 to output the inverse of the key matrix S_i for the given block (notice that $sk2$ is used in $Alg2$).
Calculate $m_i = (S_i^{-1} \times c_i) \oplus r_i \pmod 2$
Remove padding
$m = m_1||m_2||...||m_l$
return (m)
end

3.5 Cryptanalysis on RKHC

In this section, RKHC is analyzed from the perspective of basic (yet typical) attack models, and also briefly analyzed for important attack models used for block ciphers, such as Differential and Linear Cryptanalysis, based on the attacks performed against DES.

3.5.1 Basic Attack Models

This section analyses typical attack models (and also brute force).

3.5.1.1 Ciphertext Only Attack and Brute Force

RKHC is secure under COA. Since the attacker has to work against ciphertext only, it is not possible to get any information about the key or plaintext because the confusion and diffusion are assured by the HC construction, and on the proposed modified version of the cipher (RKHC), each block is encrypted with a different (pseudo-random) key matrix, obtained using a CSPRNG with very high entropy (eg., Salsa20 which is Cryptographically secure and relatively fast).

RKHC with 8×8 = 64 bits is not recommended because it is possible to try out all $2^{(8\times8)}$ key matrices by brute force. Notice that the subset of nonsingular matrices is even lower: $M_n = \prod_{i=0}^{n-1}(2^n - 2^i)$. This is a problem if an attacker can directly compute the nonsingular matrices without trying each to verify the determinant (singularity). From 16×16 = 256 bits on, each key matrix has a strength which is not breakable in useful time. Even if an attacker could perform $2^{(n\times n)}$ for one block, (s)he would then need to do the same for each block, since RKHC is assuming the usage of a CSPRNG, which assures forward and backward secrecy. In order to break RKHC by brute force it is possibly preferable to try all secret seeds instead. In this case, the security against brute force will depend on the quality of the underlying CSPRNG. As such, a state-of-the-art CSPRNG such as Salsa20, Chacha, combined with adequate key generation algorithms (eg., /dev/random, /dev/urandom or /dev/arandom directories in Unix-like OS) would completely guarantee that no brute force was possible.

3.5.1.2 Known Plaintext Attack

The classical HC is insecure in the KPA model because the attacker needs only n (linearly independent) plaintext-ciphertext block pairs to construct a system of linear equations to discover the key matrix. That can be easly proven: let M be a $n \times n$ nonsingular key matrix used to encrypt n known plaintexts with length n. Let p_i be a plaintext, c_i being its corresponding ciphertext. The attacker has only to perform n multiplications $M \times p_i = c_i$. Notice that each multiplication $M \times p_i = c_i$ results on n equations with different variables. Now the attacker organizes the n^2 equations in groups of n equations with the same variables. Finally, it is only necessary to solve the n systems of linear equations by a trivial method performed in few seconds by a common computer to get all elements of the M matrix (one key).

RKHC corrects the KPA vulnerability by generating a different key matrix for each block and XORing the plaintext with a pseudo-random mask before its multiplication by the key matrix (thus avoiding weak inputs). Notice that the fact that RKHC keeps generating different matrices for different blocks (while keeping the state of the CSPRNG) must be assured by the implementation. In section 3.4.3, RKHC will be, briefly, analyzed against linear cryptanalysis, a strong kind of KPA wherewith the DES standard was broken.

3.5.1.3 Chosen Plaintext Attack

Since each plaintext block is XORed with a mask previous to its multiplication, RKHC is secure against CPA. The most successful attack would be performed by giving a sequence of zeros to the algorithm, in which case the output will always be the multiplication of the secret mask to the secret corresponding matrix. In adaptive CPA, the attacker can adapt or modify the chosen plaintext as needed, based on results of the previous encryptions. In this situation, the attacker cannot do much. In case of the correct use of the secret seed, RKHC will be more

resistant to CPA because it never outputs a ciphertext with patterns due to the pseudo-random masks, however standardized may be the chosen plaintext. Even in ECB mode, the key would not be compromised, so RKHC is resistant to this kind of attack.

3.5.1.4 Chosen Ciphertext Attack and Adaptive Chosen Ciphertext Attack

RKHC may be vulnerable to CCA combined with CPA if not prevented against it. In order to recover one block mask and the corresponding key matrix, the attacker may try the following:

1) CCA: The attacker submits a sequence of zeros ($c_1 = 0_x$) to be decrypted by an oracle (a RKHC server). In this case, the attacker is deceiving the server with a false, very well-chosen, ciphertext. The multiplication of any block of c_1 by its corresponding key matrix will always result in a block of zeros, which will be XORed to the corresponding mask. Finally, the output p_1 will be the concatenation of as many masks according to the length of c_1;

2) CPA: Now the attacker has only to submit n different plaintext blocks previously XORed to its corresponding mask (according to the block position) — in this case the mask has no effect in the input;

3) Now the attacker has n plaintext-ciphertext block pairs to perform the KPA presented above to recover the key matrix of any block.

This attack can be solved if RKHC is further improved to XOR each block by the corresponding mask before and after its multiplication by the corresponding matrix, instead of just one mask. In this case, the mask is protecting RKHC against CPA in one side and against CCA in the other side. If we use two different masks the algorithm will be stronger, but the performance decreases due to generating the other mask.

3.5.1.5 Side Channel Attacks

The standard RKHC is not resistant to side channel attacks, since the algorithm for generating matrices is unbalanced at certain points (e.g., certain XORs only happen when random bits are equal to 1, which would leak that information via timing attacks). In order to prevent those attacks, dummy operations would have to be inserted at those points, but this is implementation dependant.

3.5.2 Differential Cryptanalysis

In this and the following subsections, RKHC will be, briefly, analyzed under the main methods to break weak block ciphers, namely differential cryptanalysis and linear cryptanalysis. As defined by Lai et al. [LMM91], secret-key block ciphers are cryptosystems based on iterating a cryptographically weak function several times. Each iteration is called a round.
In the case of RKHC we do not iterate a weak function. Basically, RKHC has only two or three steps: 1) XOR between the block to be encrypted and a pseudo-random mask; 2) the multiplication of the key matrix by the block; 3) XOR between the encrypted block and a second pseudo-random mask. As can be seen, another particularity is that these steps may not be called rounds, because they work differently to the ciphers based on Feistel Networks. So, these traditional attacks have to be adapted to the RKHC cryptosystem.

Consider a wrong implementation of RKHC in which the two fixed masks, r_1 and r_2, are both XORed to the plaintext prior to its multiplication by the corresponding key matrix ($r_1 \oplus p_i \oplus r_2$). The attacker cannot perform the adaptive CCA presented in the subsection 3.2.4 due to the second XOR. However, in order to be able to construct the differential cryptanalysis characteristic, consider the adaptation of CCA2 as follows:

1) CCA: The attacker submits a sequence of zeros ($c_1 = 0_x$) to be decrypted by the oracle. Each block of zeros will be multiplied by the corresponding decryption key matrix (with no effect). Then, the resulting zeros are submitted to the two masks r_1 and r_2, thus the attacker will get $p_1 = r_1 \oplus r_2$;

2) CPA: Now the attacker gives p_1 to the oracle to be encrypted. The result will always be a sequence of zeros (0_x) due to $p_1 = r_1 \oplus r_2 \longrightarrow p_1 \oplus r_1 \oplus r_2 = 0_x$;

3) Notice that p_1 enables the attacker to introduce any bit to be encrypted by RKHC, but (s)he cannot control exactly if the multiplication of his bit(s) by the key matrix will be performed, because (s)he does not know the masks r_1, r_2. If it could be possible, then the attacker would be able to perform CCA2 from now on;

4) Now, the attacker has to study how to introduce bits into c_1 in order to control RKHC. Here the power of diffusion and confusion of the matrix multiplication are the worst problem for the attacker.

Recall that the secure RKHC against all of these attacks is defined with two masks as follows: $c_i = [S_i \times (r_{i_1} \oplus p_i) \pmod 2] \oplus r_{i_2}$.

3.5.3 Linear Cryptanalysis

As the purpose of Linear Cryptanalysis is to find the following "effective" linear expression $P[i_1, i_2, ..., i_a] \oplus C[j_1, j_2, ..., j_b] = K[k_1, k_2, ..., k_c]$, let us compute the probability for this expression for RKHC, using the scheme with two masks. Notice that there are three operations ($[S_i \times (r_{i_1} \oplus p_i)] \oplus r_{i_2}$) and it is considered that they contain a very high entropy, so the probability to get any ciphertext c_i is $p_1 = \frac{1}{2^n} \times \frac{1}{2^n} \times \frac{1}{2^n} = \frac{1}{2^{3n}}$, where n is the block size (32 or 64 bits). So, the probability to get the linear expression will be $p_1 \times p_2$, where p_2 is the probability of a specific XOR $P[i_1, i_2, ..., i_a] \oplus C[j_1, j_2, ..., j_b]$ being equal to $K[k_1, k_2, ..., k_c]$.
The final probability is $p = p_1 \times p_2 = \frac{1}{2^{4n}}$. For the case of 32 bit matrices, $p = \frac{1}{2^{128}}$, which may be considered negligible.

3.6 Conclusions

This chapter presented the proposed HC modification as an alternative to the Classical HC, which is vulnerable to KPA. The proposed cryptosystem, RKHC, is efficient and more secure compared to other HC modifications in literature. Efficient methods to generate encryption and decryption key matrices, in order to encrypt and decrypt each block of the message with its key, were also proposed and explained. An efficient method to perform the encryption process and another one to perform decryption were presented as well.

The resistance of RKHC against the main basic attacks was also analyzed and a brief discussion on the security of the cipher against more complex models was also performed, thought in a brief manner. A more detailed discussion is left as future work. In those cases, nonetheless, hints on strong suspicions that the cipher should resistant to those attack models were provided, as well as how the proofs should proceed. The proposed version resists to all of these attacks, but some modifications on the algorithm would make it vulnerable to some or combinations of them.

Chapter 4

RKHC Implementation, Tests and Results

4.1 Introduction

This chapter presents all the tests performed on the non-optimized implementation of the proposed RKHC and on other algorithms, on the one hand to provide a clear idea of the performance of the approach and, on the other, to provide a baseline (and practical) comparison with other state-of-the-art cryptographic ciphers. The tests include randomness tests on the matrix generation algorithm as well, apart from the performance benchmarking on RKHC and other ciphers (third and fourth sections, respectively). The next section comments briefly on the specific implementation performed along this project.

4.2 Implementation

The proposed RKHC was implemented in ANSI C programming language. Two variants were implemented: one using 32×32 matrices and other with 64×64 bits matrices to favor performance (`unsigned int` and `unsigned long` can be used to save exactly 32 and 64 bit integers in C). Each matrix is saved in an array of n positions, which is to say, an array of n integers of 32 or 64 bits, in which each position is a row of n positions.

The implemented version of RKHC and used in the tests described below was not optimized. The implementation of the encryption and decryptions functions was performed in CBC mode, since it favors security against differential and linear cryptanalysis attacks, in case the implementation is used in future. It should be mentioned that the implementation includes additional dummy operations, which were introduced to prevent side-channel attacks. These dummy functions are simple operations performed in conditional branches in cases where the pseudo-random bit states XOR was not required. As such, this implementation is secure against all of the cryptanalysis discussed in chapter 3. The complete RKHC library may be found in the following github repository: `https://github.com/mafone/RKHC`.

4.3 Matrix Generator Randomness Tests

In order to compare the randomness of the nonsingular matrices generators mentioned in section 2.3.4, both were implemented in C programming language, and the results of their encryption using the generated matrices were submitted to the well-known *SmallCrush*, *Crush* and *BigCrush* TestU01 stringent batteries of statistical tests. As TestU01 works only with integers, we generated very long patterns of n bits (e.g., 11111111..., 11001100..., 10101010...) or pseudo-random numbers and multiplied them by the generated matrices; the resulting outputted sequences of n bits were then fed to TestU01. The main idea was to test if the matrices generated were able to transform a predictable (and weak) input into an apparently random

sequence of bits. Notice that this does not attest the security of the ciphers built upon these methods to generate matrices in case the tests are passed, but if a method fails any of these tests, one can immediately conclude that it cannot be used for cryptographic purposes.

We compared RKHC to the Dana Randall method, due to be the only method found in literature to generate pseudo-random nonsingular matrices, which failed in 14 of the 15 tests of the simplest battery known as *SmallCrush*. RKHC passed all three batteries of tests (a total number of 319 tests). A summary of the output of the batteries is included in Table 4.1. It can be safely concluded that proposed method fulfills the purpose of generating nonsingular matrices in a random manner very successfully. It can also be concluded that the method from Dana is not suitable for the purpose at hands.

Table 4.1: Summary of the results concerning tests with the TestU01 batteries of tests (version 1.2.3 of TestU01 was used in this work).

Generator	Battery	Number of Statistics	Total CPU Time	Tests passed
Dana	SmallCrush	15	05:36:03.20	1 out of 15
RKHC	SmallCrush	15	00:59:54.15	All tests were passed
RKHC	Crush	144	21:39:27.39	All tests were passed
RKHC	BigCrush	160	40:48:07.33	All tests were passed

4.4 RKHC Benchmarking

A series of simple benchmark experiments were performed, resorting to well-known state-of-the-art symmetric-key ciphers (Salsa20, AES128, AES192 and AES256), as well as to RKHC. Benchmarking was done by asking each cipher to encrypt and decrypt different sized files multiple times (eg., 33 times). Files with sizes 1 MB, 10 MB, 100 MB and 1 GB were randomly generated for these tests. For each experiment, the average ($\overline{X}$) and standard deviation (σ) of the processing times (measured in seconds) and memory usage (measured in KB) were calculated. AES and RKHC were set up to operate in CBC mode, while salsa20 was used as a stream cipher. All tests were performed in a Macbook Pro 15 Mid-2010 (Intel Quad-core, i7-620m 2.66 GHz, 8 GB DDR3 RAM, MacOS 10.13.5). The results of these experiments are summarized in Tables 4.2 to 4.5, which basically show that, for now, RKHC is almost five times as slow as AES256, which hints that an optimized implementation is required before further comparisons. As expected, Salsa 20 was the most efficient algorithm, both in terms of processing and memory requirements. Interestingly, the gap between RKHC and the other ciphers is not as big as expected for the decryption operation, when compared to encryption (here RKHC is only twice as slow as AES). In terms of memory footprint, all ciphers behaved similarly. With some implementation modifications, RKHC may be at least three times faster, because at least 66% of the consumed time is being lost in bits generation (the bits generation outside of the loops may improve this). Other optimization processes can be performed in order to make RKHC very fast.

Table 4.2: Processing time (s) for Salsa20, AES-CBC and RKHC during encryption.

Cipher	Salsa 20	AES 128	AES 192	AES 256	RKHC 32
1 MB $\overline{X}$	0.24	0.36	0.43	0.42	2.73
1 MB σ	0.012	0.03	0.03	0.012	0.07
10 MB $\overline{X}$	2.49	3.58	4.22	4.15	26.65
10 MB σ	0.30	0.20	0.29	0.06	1.88
100 MB $\overline{X}$	22.23	33.95	42.26	44.74	257.16
100 MB σ	0.48	2.06	1.71	1.50	23.94
1 GB $\overline{X}$	213.31	311.79	412.68	447.78	2551.27
1 GB σ	1.92	16.07	25.25	35.72	173.56

Table 4.3: Processing time (s) for Salsa20, AES-CBC and RKHC during decryption.

Cipher	Salsa 20	AES 128	AES 192	AES 256	RKHC 32
1 MB $\overline{X}$	0.24	1.65	2.02	2.02	4.00
1 MB σ	0.012	0.08	0.09	0.04	0.13
10 MB $\overline{X}$	2.49	16.33	19.55	20.00	38.67
10 MB σ	0.30	0.68	0.67	0.08	2.09
100 MB $\overline{X}$	22.23	155.28	196.22	213.68	376.35
100 MB σ	0.48	9.36	5.79	4.41	35.74
1 GB $\overline{X}$	213.31	1440.32	1902.11	2188.33	3718.94
1 GB σ	1.92	65.0004	130.15	236.05	237.51

Table 4.4: Memory (KB) usage for Salsa20, AES-CBC and RKHC during encryption.

Cipher	Salsa 20	AES 128	AES 192	AES 256	RKHC 32
1 MB $\overline{X}$	934.42	945.21	945.33	949.21	952.61
1 MB σ	8.32	5.56	4.74	8.24	3.32
10 MB $\overline{X}$	937.70	947.15	948.06	949.70	955.03
10 MB σ	15.67	253.86	8.72	9.10	11.43
100 MB $\overline{X}$	937.97	951.03	955.06	956.61	956.48
100 MB σ	9.86	14.88	9.88	13.08	10.03
1 GB $\overline{X}$	940.06	955.64	956.15	959.24	956.73
1 GB σ	7.54	11.08	14.14	17.63	10.83

Table 4.5: Memory (KB) usage for Salsa20, AES-CBC and RKHC during decryption.

Cipher	Salsa 20	AES 128	AES 192	AES 256	RKHC 32
1 MB $\overline{X}$	934.42	946.00	947.61	948.48	954.30
1 MB σ	8.32	10.85	15.49	5.36	11.08
10 MB $\overline{X}$	937.70	948.00	950.27	952.91	954.79
10 MB σ	15.67	18.18	16.04	9.21	12.38
100 MB $\overline{X}$	937.97	948.97	954.55	954.91	955.15
100 MB σ	9.86	13.19	19.17	12.99	13.44
1 GB $\overline{X}$	940.06	950.30	955.03	956.61	955.70
1 GB σ	7.54	16.72	14.43	15.33	6.55

4.5 Conclusions

This chapter presented the implementation of RKHC in ANSI C programming language and discussed its performance compared to well-known algorithms. The randomness of the RKHC key matrix generation algorithm was tested using the well-known library of statistical tests, TestU01, corroborating the statement that the proposed method does, indeed, generate random matrices from the subset of nonsingular matrices. It clearly outperformed the method for generating nonsingular matrices described by Dana Randall [Ran93]. The performance of a rough implementation of RKHC was tested against the ones of state-of-the-art symmetric ciphers, such as AES and Salsa20, whose experiments clearly shown the weight of optimization and CPU support on the latter; besides that, we could realize which points on RKHC are consuming more time: about 66% of the time are being consuming for the bits generation, this is an implementation detail and must be improved (the generation of all bits should be performed outside the algorithm, as we already know the number of needed bits. The dummy operations are not taking much time, but may also be improved. These issues and others should be corrected in future work.

Chapter 5

Main Conclusions and Future Work

This final chapter of this dissertation presents the main conclusions of the work described herein, and gives some insight on potential future work that can be iterated upon it to improve it.

5.1 Main Conclusions

The main purpose of this work was the research and development of a HC variant, termed herein RKHC, with better security. In chapter 2, several modifications of the classical HC were presented, but with a common ground between all of them: none were able to fully overcome the security issues that have been identified in the specific publications and along this document.

The work performed subsequently comprised the development of efficient methods to perform the encryption and decryption processes, all the while showing resistance to several attack models such as KPA, CPA or CCA. This was, then, demonstrated for the main basic attacks, and also briefly discussed for the most powerful attacks for block ciphers, namely the differential and linear attacks, of which some variants were performed as well. The proposed version was designed to resist all of these attacks, but more work is needed to assure that no mistakes were committed while performing these demonstrations. RKHC is nonetheless indubitably considerably more secure than the previously proposed methods.

Another strong contribution of the work was the design, development, implementation and testing of a method for generating pseudo-random nonsingular key matrices and their inverse which, when compared to the only other method found in the literature, show much better performance in statistical tests, showing its value for cryptographic applications. An implementation in ANSI C programming language was also developed, and the randomness of the full RKHC key matrix generation algorithm was tested using a well-known library of statistical tests (TestU01), as previously mentioned, corroborating the statement that the proposed method, indeed, generates random matrices of the subset of nonsingular matrices, at least from the practical perspective. The performance of RKHC was tested by encrypting and decrypting files of different sizes and, then, compared to AES and Salsa20. These tests show a rather large gap in performance, though it can be partially attributed to the non-optimized nature of the implementation and the optimization of modern processors for the other commonly used ciphers.

5.2 Future Work

Several iterations can be made upon this work to improve it and ascertain its value. Mainly, a deeper cryptanalysis is needed and cross verified by other peers. The usage of other cryptanalysis methods, such as meet-in-the-middle and bicliques rebound attacks, related-key attacks, invariant subspace attacks, algebraic attacks or integral attacks, among others, is also needed to fully understand the strong and weak points of the cipher.

Implementing an optimized version of RKHC, and implementing and evaluating some other variants of HC to compare to RKHC in benchmarks would also further prove the advantages of the proposed method. Finally, a new comparison of a highly optimized version of RKHC with state-of-the-art algorithms would allow for a better sense on the value of the method when compared to them.

References

[AJPP09] Bibhudendra Acharya, Debasish Jena, Sarat Kumar Patra, and Ganapati Panda. Invertible, Involutory and Permutation Matrix Generation Methods for Hill Cipher System. In *Advanced Computer Control, 2009. ICACC'09. International Conference on*, pages 410–414. IEEE, 2009. 25

[APP08] Bibhudendra Acharya, Sarat Kumar Patra, and Ganapati Panda. Image encryption by novel cryptosystem using matrix transformation. In *Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on*, pages 77–81. IEEE, 2008. 25

[ARPP07] Bibhudendra Acharya, Girija Sankar Rath, Sarat Kumar Patra, and Saroj Kumar Panigrahy. Novel methods of generating self-invertible matrix for Hill Cipher algorithm. 2007. 25

[Bas11] Sandipan Basu. International Data Encryption Algorithm (Idea)–A Typical Illustration. *Journal of global research in Computer Science*, 2(7):116–118, 2011. 8

[Ber05] Daniel J Bernstein. Salsa20 specification. *eSTREAM Project algorithm description, http://www. ecrypt. eu. org/stream/salsa20pf. html*, 2005. 7

[Ber08a] Daniel J Bernstein. ChaCha, a variant of Salsa20. In *Workshop Record of SASC*, volume 8, pages 3–5, 2008. 7

[Ber08b] Daniel J Bernstein. The Salsa20 family of stream ciphers. In *New stream cipher designs*, pages 84–97. Springer, 2008. 6

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing*, 13(4):850–864, 1984. 17

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991. 20

[CBM13] Oksana Cherednichenko, AA Baranov, and TI Morozova. Side-channel Attack. 2013. 19

[CDR04] Thomas W Cusick, Cunsheng Ding, and Ari R Renvall. *Stream ciphers and number theory*, volume 66. Elsevier, 2004. 6

[CEOP08] AG Chefranov, A Elci, B Ors, and B Preneel. Secure Hill Cipher modification shcm. In *Proc. Of the First International Conference on Security of Information and Network*, pages 34–37, 2008. 24

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Annual International Cryptology Conference*, pages 13–25. Springer, 1998. 18, 19

[Dwo01] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV, 2001. 8, 13

[Dwo07] Morris J Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. Technical report, 2007. 10

[Dwo16] Morris J Dworkin. Recommendation for block cipher modes of operation: The CMAC mode for authentication. Technical report, 2016. 11

[Eve80] Howard Whitley Eves. *Elementary matrix theory*. Courier Corporation, 1980. 28

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009. 6

[Hil29] Lester S Hill. Cryptography in an algebraic alphabet. *The American Mathematical Monthly*, 36(6):306–312, 1929. 22

[IAD06] IA Ismail, Mohammed Amin, and Hossam Diab. How to repair the Hill Cipher. *Journal of Zhejiang University-Science A*, 7(12):2022–2030, 2006. 24

[ISO05] ISO/IEC. *International Standard ISO/IEC 7816-4*. ISO, 2005. 13

[KD13] Liam Keliher and Anthony Z Delaney. Cryptanalysis of the toorani-falahati Hill Ciphers. In *Computers and Communications (ISCC), 2013 IEEE Symposium on*, pages 000436–000440. IEEE, 2013. 24

[Kel10] Liam Keliher. Cryptanalysis of a modified Hill Cipher. *International Journal of Computer and Network Security*, 2(7):122–126, 2010. 24

[KSF99] John Kelsey, Bruce Schneier, and Niels Ferguson. Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator. In *International Workshop on Selected Areas in Cryptography*, pages 13–33. Springer, 1999. 15, 16, 17

[KT13] Liam Keliher and Samuel Thibodeau. Slide Attacks against Iterated Hill Ciphers. In *International Symposium on Security in Computing and Communication*, pages 179–190. Springer, 2013. 24

[Lai92] Xuejia Lai. *On the design and security of block ciphers*. PhD thesis, ETH Zurich, 1992. 18

[LH94] Susan K Langford and Martin E Hellman. Differential-linear cryptanalysis. In *Annual International Cryptology Conference*, pages 17–25. Springer, 1994. 22

[LLL04] Chu-Hsing Lin, Chia-Yin Lee, and Chen-Yu Lee. Comments on Saeednia's improved scheme for the Hill Cipher. *Journal of the Chinese institute of engineers*, 27(5):743–746, 2004. 24

[LMM91] Xuejia Lai, James L Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 17–38. Springer, 1991. 36

[LMSV07] Elena C Laskari, Gerasimos C Meletiou, Yannis C Stamatiou, and Michael N Vrahatis. Cryptography and cryptanalysis through computational intelligence. In *Computational Intelligence in Information Assurance and Security*, pages 1–49. Springer, 2007. 5

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988. 6

[LRW02] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable block ciphers. In *Annual International Cryptology Conference*, pages 31–46. Springer, 2002. 12

[LRW11] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable block ciphers. *Journal of cryptology*, 24(3):588–613, 2011. 12

[LS07] Pierre L'Ecuyer and Richard Simard. TestU01: A software library in ANSI C for empirical testing of random number generators. 2007. 17, 18

[LZC08] Cheng-qing Li, Dan Zhang, and Guan-rong Chen. Cryptanalysis of an image encryption scheme based on the Hill Cipher. *Journal of Zhejiang University-Science A*, 9(8):1118–1123, 2008. 24

[M+03] George Marsaglia et al. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003. 16

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993. 21, 22

[Mat94] Mitsuru Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In *Annual International Cryptology Conference*, pages 1–11. Springer, 1994. 8

[MC09] Ahmed Y Mahmoud and Alexander G Chefranov. Hill cipher modification based on eigenvalues hcm-EE. In *Proceedings of the 2nd international conference on Security of information and networks*, pages 164–167. ACM, 2009. 25

[MC12] AY Mahmoud and Alexander G Chefranov. Secure Hill Cipher modification based on generalized permutation matrix SHC-GPM. *Information Sciences Letters*, pages 91–102, 2012. 24, 25

[MC14] Ahmed Mahmoud and Alexander Chefranov. Hill cipher modification based on pseudo-random eigenvalues. *Applied Mathematics & Information Sciences*, 8(2):505, 2014. 25

[MKK12] Kondwani Magamba, Solomon Kadaleka, and Ansley Kasambara. Variable-length Hill Cipher with MDS Key Matrix. *arXiv preprint arXiv:1210.1940*, 2012. 24

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. 16

[MV04] David McGrew and John Viega. The Galois/counter mode of operation (GCM). *Submission to NIST Modes of Operation Process*, 20, 2004. 10, 11

[OPSS13] Mohammad Ali Orumiehchiha, Josef Pieprzyk, Elham Shakour, and Ron Steinfeld. Cryptanalysis of RC4 (n, m) Stream Cipher. In *Proceedings of the 6th International Conference on Security of Information and Networks*, pages 165–172. ACM, 2013. 6

[pada] Padding mechanisms [online]. Available from: http://www.crypto-it.net/eng/theory/padding.html [cited July 2018]. 13, 14

[padb] Padding schemes for block ciphers [online]. Available from: https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html [cited July 2018]. 13, 14

[PLM06] François Panneton, Pierre L'ecuyer, and Makoto Matsumoto. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):1–16, 2006. 16

[Pub01] NIST FIPS Pub. 197: Advanced encryption standard (AES). *Federal information processing standards publication*, 197(441):0311, 2001. 8

[Pud01] Marina Pudovkina. A known plaintext attack on the ISAAC keystream generator. *IACR Cryptology ePrint Archive*, 2001:49, 2001. 17

[Rah13] Rahman, M Nordin A and Abidin, AFA and Yusof, Mohd Kamir and Usop, NSM. Cryptography: A new approach of classical hill cipher. *International Journal of Security and Its Applications*, 7(2):179–190, 2013. 25

[Ran93] Dana Randall. Efficient generation of random nonsingular matrices. *Random Structures & Algorithms*, 4(1):111–118, 1993. xi, 24, 42

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):365–403, 2003. xvii, 12

[RGX11] Gregory Gordon Rose, Alexander Gantman, and Lu Xiao. Cryptographically secure pseudo-random number generator, September 13 2011. US Patent 8,019,802. 16, 17

[Riv87] R Rivest. Rivest cipher 4 (RC4), 1987. 6

[Riv94] Ronald L Rivest. The RC5 encryption algorithm. In *International Workshop on Fast Software Encryption*, pages 86–96. Springer, 1994. 8

[RRVGMM+08] Y Rangel-Romero, R Vega-García, A Menchaca-Méndez, D Acoltzi-Cervantes, L Martínez-Ramos, M Mecate-Zambrano, F Montalvo-Lezama, J Barrón-Vidales, N Cortez-Duarte, and F Rodríguez-Henríquez. Comments on "How to repair the Hill Cipher". *Journal of Zhejiang University Science A*, 9(2):211–214, 2008. 24

[RSN+01] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Booz-Allen and Hamilton Inc Mclean Va, 2001. 14, 15, 17

[Sae00] Shahrokh Saeednia. How to make the Hill Cipher secure. *Cryptologia*, 24(4):353–360, 2000. 24

[Sal12] Salsa20/12 [online]. 2012. Available from: http://www.ecrypt.eu.org/stream/e2-salsa20.html [cited March 2012]. 7

[Sch94] Bruce Schneier. The Blowfish encryption algorithm. *Dr Dobb's Journal-Software Tools for the Professional Programmer*, 19(4):38–43, 1994. 8

[Sch95] Bruce Schneier. The Blowfish Encryption Algorithm-One-Year Later, 1995. 8

[Sha01] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. 23

[SM11] Simar Preet Singh and Raman Maini. Comparison of data encryption algorithms. *International Journal of Computer Science and Communication*, 2(1):125–127, 2011. 8

[SMB09] VUK Sastry, DSR Murthy, and S Durga Bhavani. A Block Cipher Involving a Key Applied on Both the Sides of the Plain Text. *International journal of computer and network security (IJCNS)*, 1(1):27–30, 2009. 24

[SS07] VUK Sastry and N Ravi Shankar. Modified Hill Cipher with Interlacing and Iteration 1. 2007. 24

[SSB10] V Umakanta Sastry, N Ravi Shankar, and S Durga Bhavani. A Modified Hill Cipher Involving Interweaving and Iteration. *IJ Network Security*, 10(3):210–215, 2010. 24

[Sta01] NIST-FIPS Standard. Announcing the advanced encryption standard (AES). *Federal Information Processing Standards Publication*, 197:1–51, 2001. 8

[Sti05] Douglas R Stinson. *Cryptography: theory and practice*. CRC press, 2005. xvii, 5

[TF09] Mohsen Toorani and Abolfazl Falahati. A secure variant of the Hill Cipher. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 313–316. IEEE, 2009. 24

[TF11] Mohsen Toorani and Abolfazl Falahati. A secure cryptosystem based on affine transformation. *Security and Communication Networks*, 4(2):207–215, 2011. 24

[TN08] Bao Ngoc Tran and Thuc Dinh Nguyen. Modular matrix cipher and its application in authentication protocol. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD'08. Ninth ACIS International Conference on*, pages 318–323. IEEE, 2008. 24

[Wei55] AJ Weir. Sylow p-subgroups of the general linear group over finite fields of characteristic p. *Proceedings of the American Mathematical Society*, 6(3):454–464, 1955. 32

[Yan15] Helen Yan. Tweakable Block Cipher. 2015. 12