UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE SÃO JOÃO DA BOA VISTA

BRUNO HENRIQUE DE AGUIAR SILVA

Implementação e Análise do Algoritmo de Cifra Simétrica TEA

São João da Boa Vista
2020

BRUNO HENRIQUE DE AGUIAR SILVA

Implementação e Análise do Algoritmo de Cifra Simétrica TEA

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia de Telecomunicações do Campus de São João da Boa Vista, Universidade Estatual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do diploma de Graduação em Engenharia de Telecomunicações .

Orientador: Profa. Dra. Cintya Wink de Oliveira Benedito

São João da Boa Vista
2020

Silva, Bruno Henrique de Aguiar

Implementação e análise do algoritmo de cifra simétrica TEA / Bruno Henrique de Aguiar Silva -- São João da Boa Vista, 2020.
58 p. : il. color.

Trabalho de Conclusão de Curso (Bacharelado - Engenharia Eletrônica e de Telecomunicações) – Universidade Estadual Paulista (Unesp), Câmpus Experimental de São João da Boa Vista, São João da Boa Vista.
Orientador: Profa. Dra. Cintya Wink de Oliveira Benedito
Bibliografia
1. Algoritmos 2. Criptografia 3. Simulação (Computadores) 4. Telecomunicações

CDD 23. ed. – 621.382

Ficha catalográfica elaborada pela Biblioteca-BJB
Bibliotecário responsável: João Pedro Alves Cardoso – CRB-8/9717

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CÂMPUS EXPERIMENTAL DE SÃO JOÃO DA BOA VISTA
GRADUAÇÃO EM ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

TRABALHO DE CONCLUSÃO DE CURSO

IMPLEMENTAÇÃO E ANÁLISE DO ALGORITMO DE CIFRA SIMÉTRICA TEA

Aluno: Bruno Henrique de Aguiar Silva

Orientador: Profª. Drª. Cintya Wink de Oliveira Benedito

Banca Examinadora:
- Cintya Wink de Oliveira Benedito (Orientadora)
- Carina Alves Severo (Examinadora)
- Marcelo Luís Francisco Abbade (Examinador)

A ata da defesa com as respectivas assinaturas dos membros encontra-se no prontuário do aluno (Expediente nº 02/2019)

São João da Boa Vista, 31 de julho de 2020

AGRADECIMENTOS

À Deus, que é sinônimo de amor e compaixão, por me guiar durante todo o processo de graduação.

À minha família por todo o apoio e suporte, para que eu pudesse realizar os meus sonhos.

À minha orientadora, Profª. Dra. Cintya Wink de Oliveira Benedito, pela dedicação, auxílio, apoio, ensinamentos e conselhos durante todo o processo.

Aos membros da banca examinadora pela disponibilidade, atenção e conselhos na realização deste trabalho.

À todos os professores e funcionários da Unesp SJBV pelos anos que por aqui estive, me ajudando a evoluir como pessoa e profissional.

Aos meus amigos, pelos momentos inesquecíveis e também pelo apoio nos momentos difíceis.

Aos meus colegas que muito me auxiliaram nos momentos acadêmicos em que tive dificuldades, além do apoio e lembranças que jamais esquecerei.

Gratidão!

“Não viva para que a sua presença seja notada, mas para que a sua falta seja sentida.”
(BOB MARLEY)

RESUMO

A transmissão de dados confidenciais por algum canal necessita de um cenário altamente restrito à segurança, exigindo portanto, a aplicação de algum algoritmo de criptografia. Criptografia é uma técnica utilizada para proteger os dados de ataques maliciosos. Existem diferentes algoritmos utilizados em criptografia, escolhidos dependendo a aplicação desejada. Neste trabalho estamos interessados nos algoritmos TEA (*Tiny Encryption Algorithm*) e DES (*Data Encryption Standart*), ambos algoritmos são de chave simétrica e cifra de bloco e, baseados nas redes de Feistel. O TEA possui uma descrição e implementação bastante simples e se mostra eficiente para sistemas de segurança média. Além disso, o TEA alcança uma difusão e confusão completa após apenas seis rodadas. O DES apesar de atualmente ser considerado inseguro para diversas aplicações, é um algoritmo de criptografia bastante conhecido e utilizado. Neste trabalho apresentamos um estudo sobre o funcionamento do TEA, uma implementação deste algoritmo utilizando o software MATLAB e uma análise em relação a difusão, confusão e efeito avalanche para 5, 32 e 50 rodadas. Apresentamos também um estudo e uma implementação do algoritmo de criptografia DES.

PALAVRAS-CHAVE: Criptografia. Redes de Feistel. Cifra de Bloco. DES. TEA.

ABSTRACT

The transmission of confidential data through a channel requires a scenario highly restricted to security, therefore requiring the application of some encryption algorithm. Encryption is a technique used to protect data from malicious attacks. There are different algorithms used in cryptography, chosen depending on the desired application. In this work we are interested in the algorithms TEA (*Tiny Encryption Algorithm*) and DES (*Data Encryption Standart*), both algorithms are of symmetric key and block cipher and, based on Feistel networks. The TEA has a very simple description and implementation and is efficient for medium security systems. In addition, TEA achieves complete diffusion and confusion after just six rounds. DES, despite being currently considered insecure for several applications, is a well-known and used encryption algorithm. In this work we present a study on the functioning of the TEA, an implementation of this algorithm using the MATLAB software and an analysis in relation to diffusion, confusion and avalanche effect for 5, 32 and 50 rounds. We also present a study and an implementation of the DES encryption algorithm.

KEYWORDS: Cryptography. Feistel Networks. Block Cipher. DES. TEA.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AES	Advanced Encryption Standard
ASCII	Código Padrão Americano para o Intercâmbio de Informação
DES	Data Encryption Standard
FIPS	Federal Information Process Standard
FPGA	Field Programmable Gate Array
IBM	Empresa norte-americana de informática
IoT	Internet of Things
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NSA	Nacional Security Agency
RFID	Radio-Frequency IDentification
SAC	Strict Avalanche Criterion
TEA	Tiny Encryption Algorithm
XOR	Operação lógica "ou exclusivo"
XTEA	eXtended Tiny Encrypt Algorithm

LISTA DE SÍMBOLOS

«4	Deslocamento à esquerda em 4 bits
»5	Deslocamento à direita em 5 bits
C_n	Transformação da chave à esquerda do DES
D_n	Transformação da chave à direita do DES
DD	Deslocamento de bits à direita
DE	Deslocamento de bits à esquerda
delta	Constante utilizada no TEA
E	Expansão
EA	Efeito avalanche
f	Função F do modelo de Feistel
FP	Permutação Final
IP	Permutação Inicial
k	Chave Completa de um modelo de Feistel
k(0)	Primeira parte da chave K do TEA
k(1)	Segunda parte da chave K do TEA
k(2)	Terceira parte da chave K do TEA
k(3)	Quarta parte da chave K do TEA
k_n	Subchaves redes de Feistel e DES
L_n	32 primeiros bits da sequência à esquerda do DES
P	Permutação P
PC-1	Permutação do DES
PC-2	Permutação do DES
R_n	32 primeiros bits da sequência à direita do DES
x	Texto original
y	32 primeiros bits da sequência à esquerda do TEA
z	32 primeiros bits da sequência à direita do TEA

SUMÁRIO

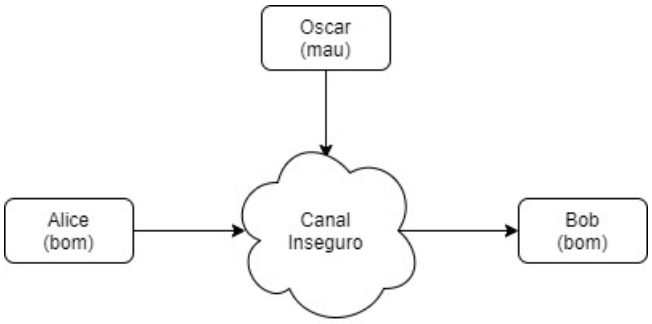
1 INTRODUÇÃO

O aumento da demanda por troca de informações com arquivos cada vez mais extensos e que necessitam de envio instantâneo, enfatizou a necessidade de sistemas de comunicação digitais cada vez mais rápidos e seguros, de modo a atingir requisitos de integridade, sigilo e não reprodução das informações transmitidas [CARPENTIERI 2018]. A criptografia fornece uma técnica para proteger e autenticar a transmissão de informações através de canais de comunicação inseguros. Com o uso de algoritmos criptográficos podemos armazenar informações confidenciais ou transmiti-las por sistemas de comunicação inseguros, de modo que pessoas não autorizadas não consigam interceptá-las [MATHIVANAN 2015].

A criptografia, ou o ato de mascarar uma mensagem, para que a mesma possa ser enviada sem que seja descoberta, exceto pelo receptor, pode ser considerada uma técnica contemporânea, ou mesmo recente. Entretanto, a ação de esconder a informação é uma técnica muito mais antiga, tanto que povos antigos, como as civilizações egípcias, gregas e romanas já utilizavam linguagens ocultas na troca de informação entre dois oficiais, por exemplo. O fato é que a criptografia está contida em uma área muito maior, a criptologia. Na verdade, da criptologia se derivam duas vertentes de pesquisa, a criptografia e a criptoanálise, ambas também com origem na língua grega [JONES H. S.; LIDDELL 1996], onde a criptoanálise é vista como a ciência que estuda formas de quebrar a codificação das cifras. Esse ato pode ser visto por alguns como algo ilícito ou mesmo criminoso. Porém, por outro lado, a criptoanálise ajuda também a testar o quão forte é esta criptografia [PAAR C.; PELZL 2009].

O exemplo clássico quando se fala em criptografia, é o exemplo da comunicação de Alice e Bob. Alice deseja se comunicar com Bob através de um canal de comunicação inseguro, então a criptografia permite que os dois se comuniquem, mascarando a mensagem a ser enviada por um canal para que um possível intruso, que chamaremos de Oscar, não possa descobrir o conteúdo da informação, chegando em Bob com segurança [PAAR C.; PELZL 2009].

Figura 1 – Bob e Alice

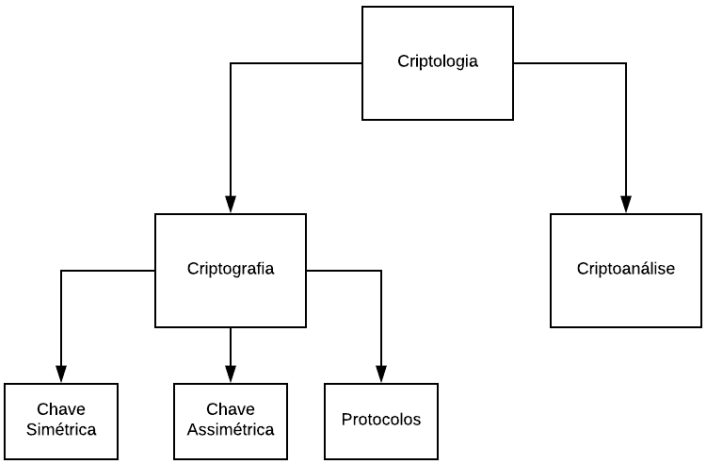


Fonte: Próprio Autor

A Figura 2 nos mostra um fluxograma da criptologia. Podemos observar que a criptografia pode ser dividida em três grupos principais, ou três tipos criptográficos [PAAR C.; PELZL 2009]. São eles:

- **Algoritmo de Chave Simétrica:** Até 1976, a criptografia era baseada em algoritmos simétricos. A principal característica destes algoritmos é que utilizam uma mesma chave privada para

Figura 2 – Fluxograma da Criptologia



Fonte: Próprio Autor

criptografia e decriptografia. Outro detalhe importante que caracteriza estes algoritmos é que a comunicação é realizada em dois canais separados, um seguro e outro inseguro. No inseguro será enviada a informação cifrada, e no seguro a chave será enviada de forma a decriptografar a informação [DELFS H.; KNEBL 2007]. Algoritmos de chave simétrica são amplamente utilizados, especialmente para criptografia de dados e verificação de integridade de mensagens.

- **Algoritmos de Chave Assimétrica:** Foram desenvolvidos a partir de 1976, por Whitfield Diffie, Martin Hellman e Ralf Merkle, tendo como principal característica uma chave pública. Foi um marco na história da criptografia, pois a chave não era mais secreta, e além disso, a chave de decriptografia deveria ser diferente da de criptografia, mas com alguma relação matemática pré-estabelecida entre elas [STALLINGS 1999].

- **Protocolos:** Os protocolos têm a função básica de trabalhar com os vários vários tipos de criptografia, servindo como uma ponte entre as aplicações [PAAR C.; PELZL 2009]. O TLS (*Transport Layer Security*), utilizado nos navegadores, é um exemplo de protocolo criptográfico.

A criptografia simétrica é dividida em dois grupos principais [MENEZES A. J.;VAN OORSCHOT 1996]:

- **Cifras de bloco:** Uma cifra de bloco transforma blocos de texto simples (*plaintext*) em texto cifrado (*ciphertext*) sob a ação de uma chave. Essa transformação é relativamente complicada, mas além da chave ser reutilizada, a criptografia de um bloco é independente de outro.

- **Cifras de fluxo:** as cifras de fluxo processam a criptografia bit a bit, utlizando para cada um deles uma fração da chave, e portanto, cada parte da sequência criptografada não possui uma relação com outra parte. Em geral, são mais rápidas em processamento do que as cifras de bloco.

Analisando as cifras de bloco, é importante salientar que muitas vezes devem ser feitas operações de adição entre duas sequências lógicas. Dessa forma, é necessário identificar o modelo que mais se

Tabela 1 – Tabela Verdade XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Fonte: Próprio Autor

adeque a aplicação, e neste caso a resposta sempre é a operação lógica XOR (*Exclusive or*) [JUNOD P.; CANTEAUT 2011]. Devido a sua natureza de sempre ter uma resposta de probabilidade cinquenta por cento 1 lógico, e de cinquenta por cento 0 lógico, ele se torna o mais interessante criptograficamente devido a sua imprevisibilidade. A Tabela Verdade para esta operação é dada na Tabela 1.

Algoritmos de criptografia simétrica com cifras de bloco serão utilizadas neste trabalho. O DES (*Data Encryption Standard*) e o TEA (*Tiny Encryption Standard*) são exemplos destes tipos de algoritmo, sendo que o TEA será o foco principal deste trabalho, e o DES é um dos algoritmos precursores para as redes de Feistel. Como vimos, os modelos criptográficos simétricos possuem como característica principal o uso de chaves privadas e a necessidade de um canal inseguro e outro seguro, para o texto criptografado e para a chave, respectivamente [PAAR C.; PELZL 2009]. A criptografia simétrica também permite que o modelo tanto de criptografia, quanto de decriptografia sejam semelhantes, permitindo sistemas mais simplificados, e ao mesmo tempo, seguros contra possíveis invasores.

A apresentação do modelo de Feistel foi o princípio de uma verdadeira revolução, quando se trata de modelos criptográficos simétricos em blocos. A estrutura de Feistel tem a vantagem de que as operações de criptografia e decriptografia são muito semelhantes, sendo idênticas em alguns casos, necessitando apenas da utilização das chaves na ordem inversa [FEISTEL 1973]. A teoria utilizada esteve presente em diversos diagramas criptográficos, começando pelo DES [SCHNEIER 2017], que foi um dos precursores neste tipo de criptografia [FEISTEL 1973]. O DES foi inicialmente apresentado em 1972. Neste ano, o então NBS (*National Bureau of Standards*), conhecido hoje como NIST (*National Institute of Standards and Technology*), necessitava de algum modelo que permitisse a troca de informações que não eram confidenciais, mas que deviam ser secretas. Em consulta à agência de segurança nacional norte-americana NSA (*National Security Agency*) em 1973, foi requisitada uma tecnologia de cifras que pudesse atingir os requisitos, entretanto, nenhum dos projetos apresentados se mostrou promissor. Em 1974, houve uma nova requisição, desta vez com uma resposta positiva, um projeto promissor da IBM (*International Business Machines*), uma cifra desenvolvida entre os anos de 1973 e 1974 a partir do modelo do algoritmo de Horst e Feistel. Já em 1975, o projeto foi finalmente inscrito no registro federal norte-americano e ocorreram subsequentes discussões, tanto que foram organizadas duas atividades para fomento do tema, contando com a presença de nomes como os dos pioneiros da chave-pública, Martin Hellman e Whitfield Diffie. Nesses encontros foram citados alguns temas importantes como o tamanho da chave e as ainda nebulosas S-boxes, que eram considerada por muitos um erro do NSA, criador desse instrumento, por considerar uma ferramenta de fácil detecção e compreensão, tornando a criptografia mais branda no sentido de quebra. Os engenheiros da IBM

alegavam que as mudanças no projeto feitas pela agência reduziram a força matemática do projeto por ter reduzido a chave e adicionado as S-boxes. Mesmo com todas as polêmicas e críticas feitas pelo meio acerca do projeto do DES, ele foi aprovado em 1976 como padrão federal, graças à cooperação entre os agentes da NSA e os engenheiros da IBM, idealizadores do projeto. Em 1977, o modelo foi catalogado para a utilização em dados não-classificados pelo FIBS PUB 46 (FIPS – Federal Information Process Standard, órgão do governo dos Estados Unidos responsável pela padronização em áreas da computação não-militares) [TUCHMAN 1997]. Já em 1994, durante o congresso científico *Fast Software Encryption*, em Leuven na Bélgica, foi apresentado ao mundo o TEA por David Wheeler e Roger Needham [WHEELER David J.; NEEDHAM 1994]. O modelo foi desenvolvido anos antes no laboratório de Computação de Cambridge, e se tratava de um modelo de criptografia, baseado em linguagem computacional C. O modelo apresentava um tipo de criptografia que, apesar de bastante simples (9 linhas de código), prometia ser bastante rápido, justamente pela sua lógica ser bastante simples. Com o passar do tempo, o sistema foi se mostrando bastante seguro, devido ao fato da criptografia ser processada em um tempo bastante reduzido, em comparação a outras do mesmo tipo, e assim, sendo quase que impossível que o código pudesse ser quebrado.

É importante identificar as fraquezas e forças do TEA em relação aos dois modelos de criptografia mais comuns, DES e AES (*Advanced Encryption Standard*) [DAOR J.; DAEMON 1999, BUDIYANTO D; PUTRO 2018], em que o AES apresenta um melhor desempenho que o DES porém neste trabalho apresentamos o DES por também utilizar as cifras de Feistel. Como dito anteriormente, o TEA é um modelo bastante simples que muitas vezes pode parecer ineficiente do ponto de vista de segurança, mas que por conta da sua simplicidade e rapidez pode ser utilizado em dispositivos menos robustos, que possuem capacidade de processamento mais reduzida. O modelo oferece um processo de criptografia completo em um tempo bastante reduzido, em comparação a outros tipos de criptografia mais comuns, fato que dificulta muito a possibilidade de quebra de segurança, pois o tempo hábil para que isso aconteça é mínimo [GILBERT 2009]. O DES, por sua vez, é um pouco diferente. Por ser mais complexo, requer uma maior capacidade de processamento e um hardware mais robusto. O processo de criptografia é mais lento, mas o nível de complexidade dos processos tornam a possibilidade de quebra uma tarefa mais difícil, mesmo que haja mais tempo para uma quebra em comparação com o TEA. O TEA também é considerado uma criptografia com menor segurança por conta de seu tamanho fixo de bloco de texto de 64 bits e de chave de 128 bits. Além disso, apesar da chave de 128 bits ser fragmentada em 4 partes de 32 bits, a mesma chave é utilizada em todas as rodadas, o que pode se tornar uma brecha para quebra de segurança. Uma condição necessária para uma cifragem ser considerada segura é satisfazer as propriedades de difusão e confusão. Claude Shannon propôs em 1949 estas técnicas para capturar os blocos fundamentais de uma função criptográfica, em vez de usar um método estatístico demorado. Shannon estava preocupado principalmente com a prevenção da criptoanálise com a ajuda da análise estatística [SHANNON 1949]. As três cifras citadas TEA, DES e AES possuem as propriedades de confusão e difusão, porém como veremos neste trabalho, a análise destas propriedades para o algoritmo TEA é bastante simples, sem precisar de tabelas como P-boxes e S-boxes. Além disso, ele alcança uma difusão e confusão total já a partir de seis rodadas do algoritmo [Ma e Obimbo 2011].

Usualmente as análises do TEA são realizadas através de implementações em *hardware*, em especial em FPGA (*Field Programmable Gate Array*) [DIKEVAR A. P.; ROUT 2015, Hussain e Badar 2015, Feldhofer e Wolkerstorfer 2008], porém neste trabalho iremos nos restringir a implementação deste algoritmo em *software*, o MATLAB. Pelo fato do TEA ser um algoritmo criptográfico leve, ele se torna adequado para sistemas embarcados que exigem alto desempenho, facilidade de implementação, alta velocidade, baixo consumo de energia e baixo custo, além da segurança [Damaj, Hamade e Diab 2008]. Neste sentido, aplicações em diversas áreas vêm sendo estudadas, como em sistemas sem fio com identificação por radiofrequência - RFID (*Radio-Frequency Identification*) [ISRASENA 2006, GILBERT 2009, ABDELHALIM M.; EL-MAHALLAWY 2013, ISRASENA 2005], em Internet das Coisas - IoT(*Internet of Things*) [Saurabh S.; SHARMA 2017] e em sistemas móveis que possuem recursos limitados e ainda se preocupam com velocidade e área [Hunn S.A.Y.; Naziri 2012].

Baseado no pressupostos acima, neste trabalho apresentamos um estudo sobre o TEA, apresentando seu algoritmo de encriptação e decriptação. Tais algoritmos serão implementados utilizando o *software* MATLAB e realizaremos algumas análises em relação a confusão, difusão e efeito avalanche, variando o número de rodadas do algoritmo. Para alcançar o foco de nosso trabalho, iremos realizar inicialmente um estudo sobre as redes de Feistel e o algoritmo DES que é o principal exemplo de algoritmo que utiliza tais redes. Iremos realizar também a implementação computacional utilizando o *software* MATLAB da criptografia e decriptografia do DES. Este trabalho está organizado da seguinte forma: No Capítulo 2 apresentamos o estudo sobre as redes de Feistel e o DES bem como as implementações computacionais da encriptação e decriptação do DES. No Capítulo 3 apresentamos um estudo do TEA, mostrando o seu funcionamento na encriptação e decriptação além de suas implementações computacionais. No Capítulo 4 apresentando as análises realizadas com o algoritmo TEA em relação a difusão, confusão e efeito avalanche. Por fim, no Capítulo 5 apresentamos as conclusões deste trabalho.

2 REDES DE FEISTEL E CRIPTOGRAFIA DES

Como vimos *Data Encryption Standard* - DES é um algoritmo de criptografia de chave simétrica em blocos, baseado nas redes de Feistel. Neste capítulo inicialmente apresentaremos um estudo sobre as redes de Feistel, que servirá de base para os capítulos deste trabalho e em seguida apresentamos um estudo sobre o DES. Mostraremos como são realizadas a encriptação e a decriptação deste algoritmo. Uma implememtação computacional em MATLAB também será apresentada para mostrar o seu funcionamento.

2.1 REDES DE FEISTEL

A rede de Feistel ou cifra de Feistel não é um esquema específico de cifra de bloco, mas sim um modelo do qual derivam-se muitas cifras de bloco, como o DES e o TEA. A rede de Feistel foi apresentada por Horst Feistel, durante o desenvolvimento de uma criptografia denominada Lucifer, em 1973, em conjunto com seu colega de IBM, Don Coppersmith [FEISTEL 1973]. Uma vantagem das redes Feistel é que um sistema criptográfico baseado nesta estrutura utiliza basicamente o mesmo algoritmo para criptografia e decriptografia. Uma característica das redes de Feistel é que elas possuem muitas rodadas de criptografia, o que ajuda aumentar a segurança. Em cada rodada, diferentes técnicas são aplicadas ao texto original para criptografá-lo. Após passar por todas essas rodadas o texto original é convertido no texto cifrado.

O funcionamento das redes de Feistel se baseia na ideia de que a informação é dividida em duas partes, sendo que as operações serão realizadas simultaneamente nas duas partes, em múltiplas vezes, e que durante cada rodada, a parte da direita e da esquerda podem ser trocadas [MENEZES A. J.;VAN OORSCHOT 1996, Biryukov 2011]. O funcionamento da encriptação é mostrado na Figura 3 e iremos detalhar a seguir. Também é importante salientar que todas as operações, tanto para as redes de Feistel, quanto para as criptografias DES e TEA, são realizadas operações na base binária.

Inicialmente, o texto original é dividido em duas partes iguais (L_0, R_0), sendo L_0 a parte da esquerda e R_0 a parte direita. Além disso, temos as subchaves k_0 até k_n geradas por algum algoritmo de geração de chave. No caso do TEA, a chave inicial é dividida em 4 partes e no DES a chave é modificada em cada rodada. Uma função f, chamada de função rodada, que é uma função característica em cada um dos modelos baseados em Feistel, será utilizada em cada rodada. Para a primeira rodada R_0 e k_0 são as entradas da função f, produzindo a saída $f(R_0, k_0)$ que fará uma operação XOR com L_0. Já a entrada R_0 permanece inalterada, completando assim a primeira rodada da criptografia e produzindo

$$(L_1, R_1) = (R_0, L_0 \oplus f(R_0, k_0)). \tag{2.1}$$

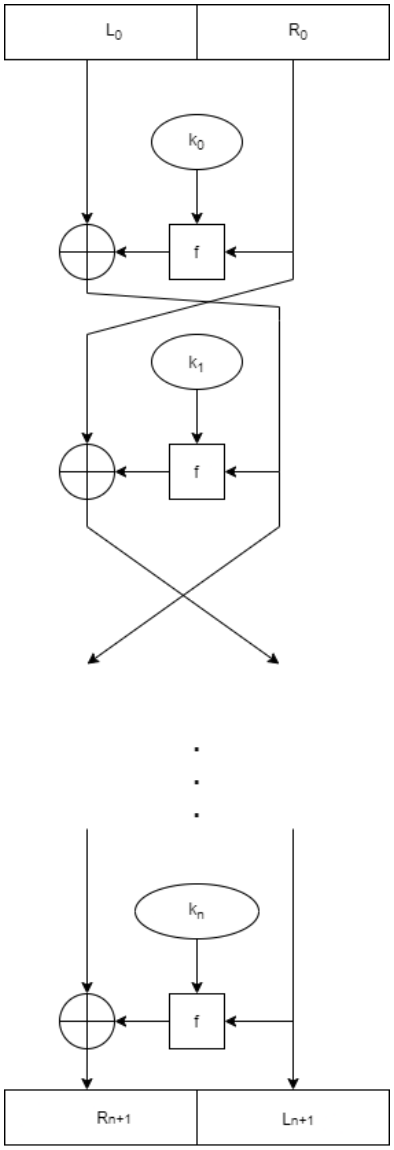
O processo continua se repetindo a cada rodada e após $n + 1$ rodadas teremos

$$(L_{n+1}, R_{n+1}) = (R_n, L_n \oplus f(R_n, k_n)), \tag{2.2}$$

que juntas, representam o texto cifrado. O número de rodadas depende do algoritmo e é diretamente

proporcional a segurança do algoritmo. Mas, simultaneamente, diminui a velocidade da criptografia e decriptografia.

Figura 3 – Exemplo de Cifra de Feistel para Criptografia



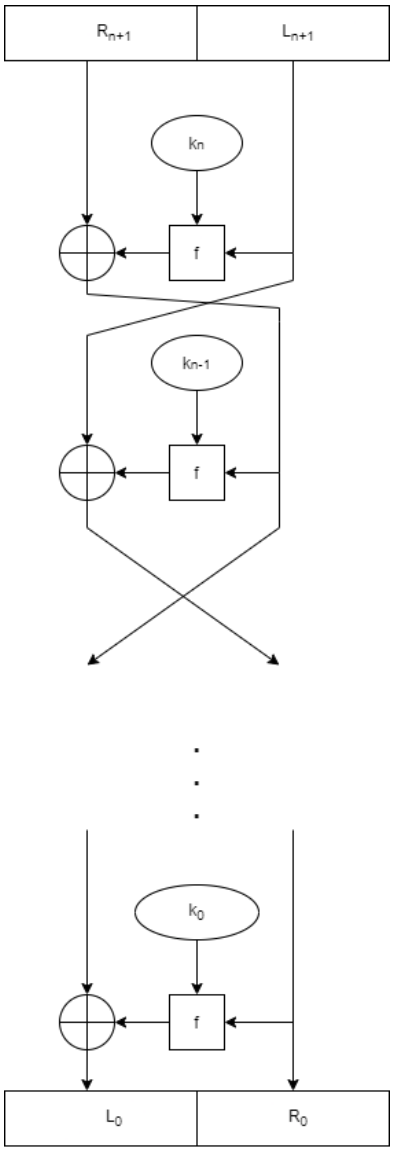
fonte: Próprio Autor

Para realizar a decriptografia, representada na Figura 4, ocorre o processo inverso, da última rodada para a primeira rodada. Iniciamos com o texto cifrado dividido em L_{n+1} e R_{n+1}, porém no sentido inverso, ou seja, (R_{n+1}, L_{n+1}), e a chave k_n. A funcão rodada f é aplicada agora L_{n+1} e k_n, e na saída $f(L_{n+1}, k_n)$ é realizada uma operação XOR com R_{n+1}. A parte L_{n+1} permanece inalterada nesta rodada. Desta forma

$$(R_n, L_n) = (L_{n+1}, R_{n+1} \oplus f(L_{n+1}, k_n)). \tag{2.3}$$

Da mesma forma, após $n + 1$ rodadas, chega-se em L_0 e R_0, que concatenados, representam o texto original.

Figura 4 – Exemplo de Cifra de Feistel para Decriptografia

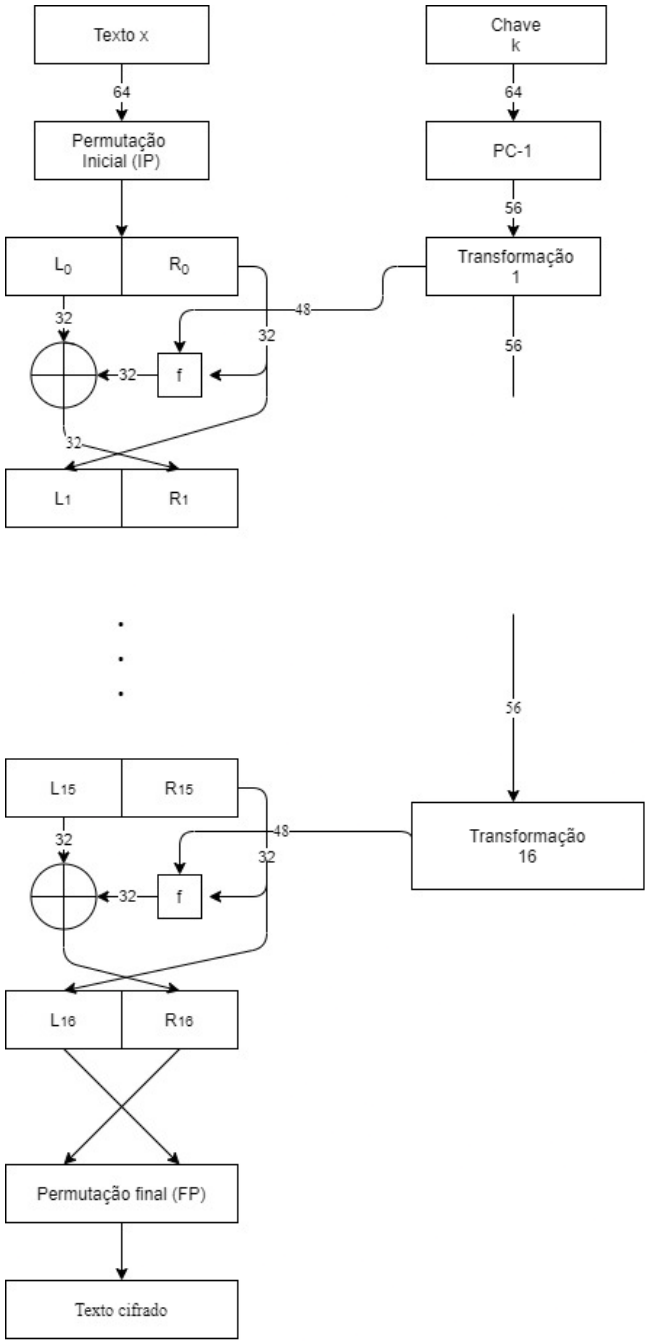


fonte: Próprio Autor

2.2 CRIPTOGRAFIA DES

Nesta seção apresentamos a criptografia do DES. O diagrama representado pela Figura 5 sintetiza o funcionamento do modelo criptográfico em diagrama de blocos [SCHNEIER 2017], o qual explicaremos a seguir. O algoritmo possui duas entradas básicas, a informação (texto original) x de 64 bits a ser criptografada, e a chave-privada k também de 64 bits que será a mesma para a criptografia e para a decriptografia.

Figura 5 – Criptografia DES



fonte: Próprio Autor

Inicialmente transformamos os vetores x e k que representam respectivamente o texto original e a chave, em matrizes 8×8. Ao texto original será aplicado uma matriz de permutação, chamada de

permutação inicial IP dada por

$$IP = \begin{pmatrix} 58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 \\ 60 & 52 & 44 & 36 & 28 & 20 & 12 & 4 \\ 62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 \\ 64 & 56 & 48 & 40 & 32 & 24 & 16 & 8 \\ 57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 \\ 59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 \\ 61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 \\ 63 & 55 & 47 & 39 & 31 & 23 & 15 & 7 \end{pmatrix}. \tag{2.4}$$

A matriz de permutação inicial, assim como todas as outras amtrizes, troca a posição dos bits, e estas são padronizadas pelo modelo, então, cada uma delas contida aqui, no caso do DES, foi retirada de Figura 5. Após a realização da permutação inicial, o resultado que permanece com 64 bits, deve ser dividido em L_0 e R_0, ambos com 32 bits cada. Essas duas metades de 32 bits juntamente com as subchaves serão as entradas para a rede Feistel, que no caso do DES consiste de 16 rodadas. Agora, vejamos o que acontece com a chave até este momento. Primeiramente, uma permutação é aplicada na matriz 8×8 que representa a chave, chamada de permutação $PC - 1$ dada por

$$PC - 1 = \begin{pmatrix} 57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 \\ 58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 \\ 59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 \\ 60 & 52 & 44 & 36 & 63 & 55 & 47 & 39 \\ 31 & 23 & 15 & 7 & 62 & 54 & 46 & 38 \\ 30 & 22 & 14 & 6 & 61 & 53 & 45 & 37 \\ 29 & 21 & 13 & 5 & 28 & 20 & 12 & 4 \end{pmatrix}, \tag{2.5}$$

que a transforma em uma matriz 7×8, sendo que os bits descartados são os bits múltiplos de 8, como podemos observar em (2.5).

Em seguida, é realizada uma transormação na chave, descrita na Figura 6. Nesta transformação, dividimos os 56 bits em C_0 e D_0, ambos com 28 bits. Após esta divisão, ocorre o chamado deslocamento à esquerda. Nesta primeira rodada, ocorre o deslocamento de um bit, o qual chamamos de DE_1 e com isso obtemos as matrizes C_1 e D_1. Após o deslocamento de bits, C_1 e D_1 são agrupados novamente em uma matriz com 56 bits, os quais serão transformados em 48 bits segundo uma matriz

de permutação 8×6 chamada de $PC-2$ dada por

$$PC-2 = \begin{pmatrix} 14 & 17 & 11 & 24 & 1 & 5 \\ 3 & 28 & 15 & 6 & 21 & 10 \\ 23 & 19 & 12 & 4 & 26 & 8 \\ 16 & 7 & 27 & 20 & 13 & 2 \\ 41 & 52 & 31 & 37 & 47 & 55 \\ 30 & 40 & 51 & 45 & 33 & 48 \\ 44 & 49 & 39 & 56 & 34 & 53 \\ 46 & 42 & 50 & 36 & 29 & 32 \end{pmatrix}, \tag{2.6}$$

onde os bits descartados nesta etapa são os das posições 9, 18, 22, 25, 35, 38, 43 e 54.

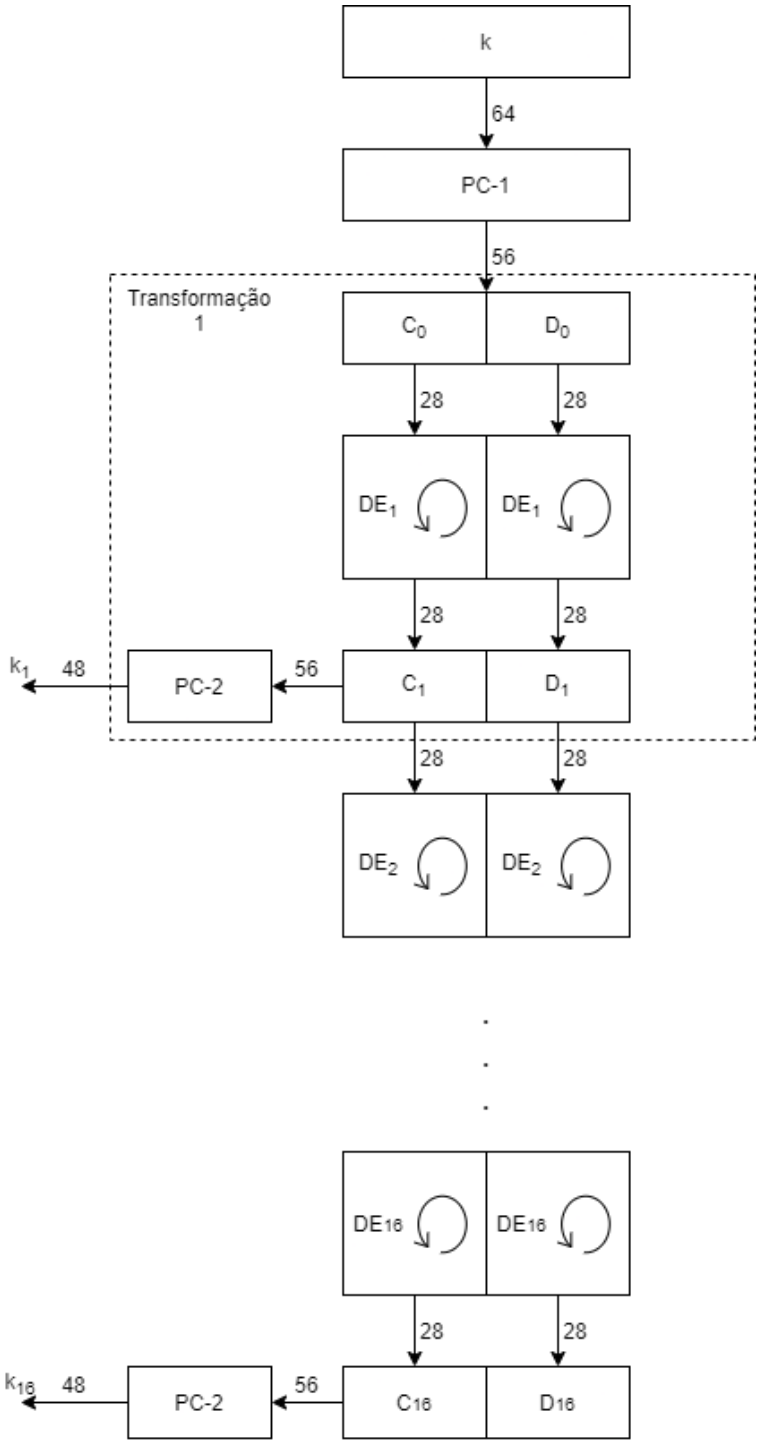
Observamos que o deslocamento à esquerda DE_i, para $i = 1, \cdots, 16$, ocorre nas dezesseis rodadas do algoritmo e a quantidade de bits rotacionados em cada rodada é dada na Tabela 2.

Tabela 2 – Bits de deslocamento

Rodada	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotacionados	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

fonte: Próprio Autor

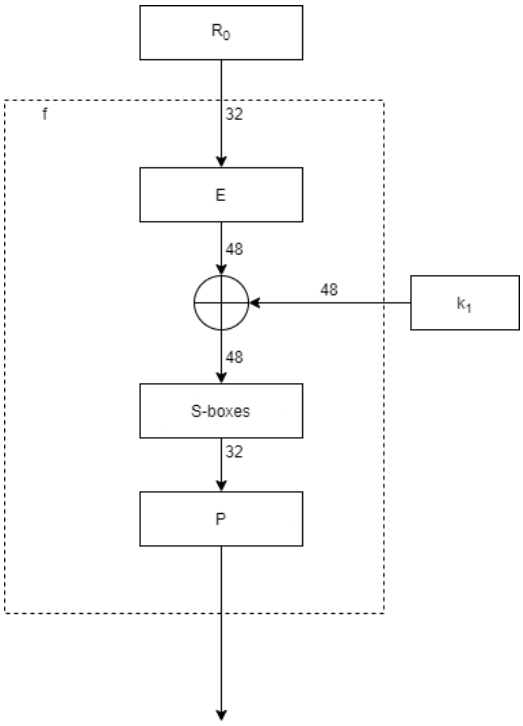
Figura 6 – Transformação



fonte: Próprio Autor

A matriz k_1 de 48 bits obtida após a transformação 1 será uma das entradas da função rodada f juntamente com a matriz R_0 advinda do texto original após aplicar a matriz 2.4. O funcionamento da função rodada f é mostrado na Figura 7 e detalhado a seguir. R_0 que possui 32 bits sofre uma

Figura 7 – Função f



fonte: Próprio Autor

expansão segundo a matriz E dada por

$$E = \begin{pmatrix} 32 & 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 10 & 11 & 12 & 13 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 16 & 17 & 18 & 19 & 20 & 21 \\ 20 & 21 & 22 & 23 & 24 & 25 \\ 24 & 25 & 26 & 27 & 28 & 29 \\ 28 & 29 & 30 & 31 & 32 & 1 \end{pmatrix}. \tag{2.7}$$

Com a matriz expandida é então realizada uma operação XOR com a matriz k_1. Após isso, é realizada uma operação utilizando uma Tabela S-box. S-boxes são operações utilizando tabelas que reduzem o número de bits [PAAR C.; PELZL 2009]. Neste caso, queremos transformar 48 bits em 32 bits. Para isso, os 48 bits são divididos em 8 partes de 6 bits, os quais serão transformados em 4 bits de acordo com a S-box correpondente. Existem 8 tabelas S-boxes diferentes, uma para cada parte, ou seja, a primeira parte será submetida à primeira S-box e assim por diante. A forma da obtenção do resultado segue algumas regras e as S-boxes têm um padrão, as linhas possuem entradas de 0 a 3 e as colunas de 0 a 15. A combinação entre o valor das duas representa um número em decimal que irá ser a resposta final em binário, que logicamente estará entre 0 e 15 para que represente 4 números binários. Para encontrar o resultado, o número da coluna e da linha são encontrados a partir da entrada, utilizando o primeiro e último bit para a linha, e os outros 4 bits centrais para a coluna, já transformados em decimais. Os resultados são novamente unidos, resultando nos 32 bits desejados.

Para um melhor entendimento, vamos utilizar um exemplo com a S-box 1 dada pela Tabela 3. Se uma entrada em binário é 010011, os bits da linha são o primeiro e o último, ou seja, 01 em binário, que representa 1 em decimal. Os bits da coluna serão 1001 em binário, ou 9 em decimal. Dessa forma, utilizaremos a linha 1 e a coluna 9 da tabela. O resultado é 06 em decimal, ou 0110 em binário.

Tabela 3 – S-box 1

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

fonte: Próprio Autor

Na matriz resultante com 32 bits é novamente aplicada uma matriz de permutação P dada por

$$P = \begin{pmatrix} 16 & 7 & 20 & 21 & 29 & 12 & 28 & 17 \\ 1 & 15 & 23 & 26 & 5 & 18 & 31 & 10 \\ 2 & 8 & 24 & 14 & 32 & 27 & 3 & 9 \\ 19 & 13 & 30 & 6 & 22 & 11 & 4 & 25 \end{pmatrix}. \tag{2.8}$$

Em seguida, uma operação XOR do resultado da permutação com L_0 é realizada. O resultado será transportado para R_1, e o valor L_1 será o mesmo de R_0. Sintetizando a rodada utilizando a notação da rede de Feistel, temos

$$(L_1, R_1) = (R_0, L_0 \oplus f(R_0, k_1)). \tag{2.9}$$

A partir das matrizes L_1 e R_1, repetimos os mesmos passos de L_0 e R_0, e prosseguimos até obter L_{16} e R_{16}. O mesmo ocorre para as subchaves de k_1 a k_{16}. Teremos,

$$(L_i, R_i) = (R_{i-1}, L_{i-1} \oplus f(R_{i-1}, k_i)), \tag{2.10}$$

para $i = 1, \ldots, 16$. Para finalizar o algoritmo, após a obtenção de L_{16} e R_{16} invertemos a posição iniciando com R_{16} e depois L_{16}. E, a esta matriz resultante aplicamos uma última permutação FP dada por

$$FP = \begin{pmatrix} 40 & 8 & 48 & 16 & 56 & 24 & 64 & 32 \\ 39 & 7 & 47 & 15 & 55 & 23 & 63 & 31 \\ 38 & 6 & 46 & 14 & 54 & 22 & 62 & 30 \\ 37 & 5 & 45 & 13 & 53 & 21 & 61 & 29 \\ 36 & 4 & 44 & 12 & 52 & 20 & 60 & 28 \\ 35 & 3 & 43 & 11 & 51 & 19 & 59 & 27 \\ 34 & 2 & 42 & 10 & 50 & 18 & 58 & 26 \\ 33 & 1 & 41 & 9 & 49 & 17 & 57 & 25 \end{pmatrix}. \tag{2.11}$$

A matriz resultante será o texto cifrado.

2.3 DECRIPTOGRAFIA DES

Com base nos modelos de redes de Feistel, podemos considerar que a decriptografia é relativamente parecida com a criptografia. Como vimos, nesses modelos a decriptografia atua como uma reconstituição, seguindo os passos da criptografia, da última rodada para a primeira. A primeira rodada da decriptografia reverte a última rodada da criptografia e assim por diante [SCHNEIER 2017]. Como podemos observar na Figura 8, no DES não é diferente, mas com um detalhe importante. Após o texto cifrado de 64 bits passar pela permutação inicial IP dada em (2.4), na primeira rodada ele será dividido em L_0^d e R_0^d (as primeiras rodadas de decriptografia, em que o índice "d"representa o processo de decriptografia) , os quais devem ser iguais a R_{16} e L_{16}. Este processo iterativo segue em todas as rodadas de modo que

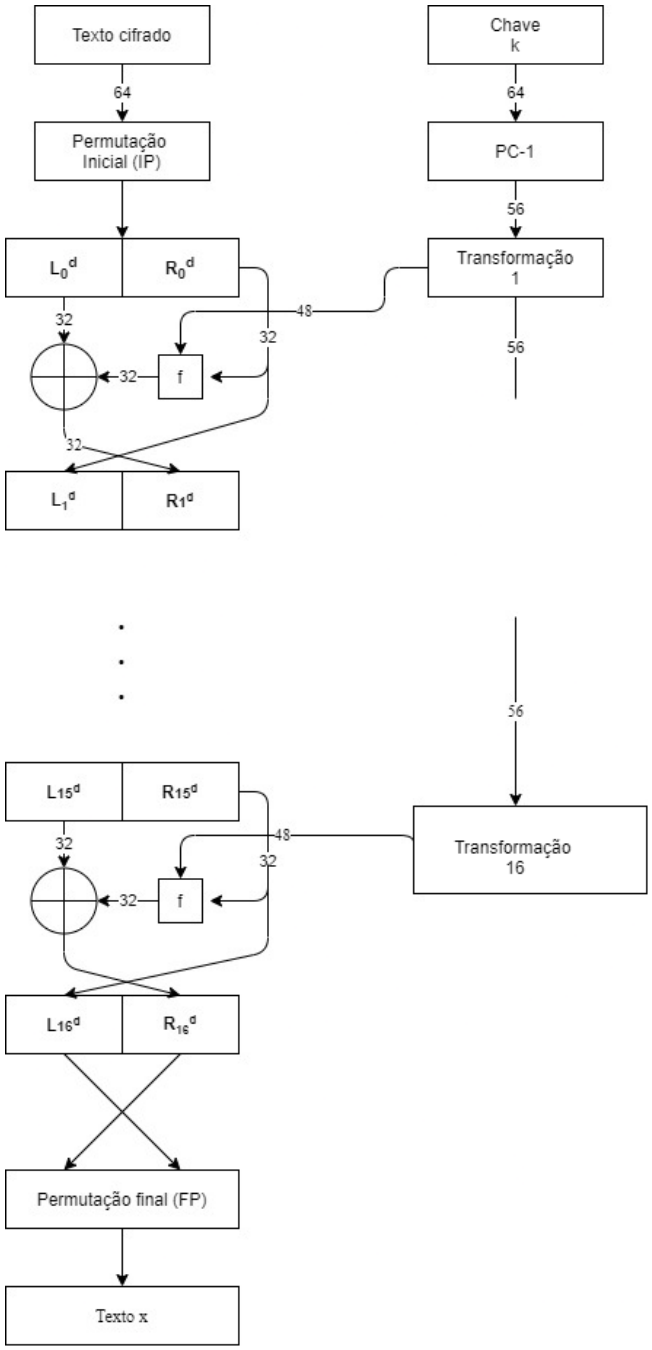
$$L_i^d = R_{16-i} \quad \text{e} \quad R_i^d = L_{16-i}, \tag{2.12}$$

para todo $i = 0, \ldots, 16$. Em particular, após a última entrada da decriptografia devemos ter

$$L_{16}^d = R_0 \quad \text{e} \quad R_{16}^d = L_0. \tag{2.13}$$

A transformação da chave k também deve ser modificada de maneira que a última alteração realizada na criptografia seja a primeira agora na decriptografia. Desta forma, a tranformação representada pela Figura 9 é realizada não mais com um deslocamento à esquerda DE e sim à direita DD, porém com uma estrutura análoga à da criptografia, sendo essa a única modificação estrutural para o blocos de decriptografia. Assim como na criptografia, o modelo é bastante complexo, e estas transformações seguem de entrada para a mesma rodada função f cujo esquema é representado na Figura 7.

Figura 8 – Decriptografia DES



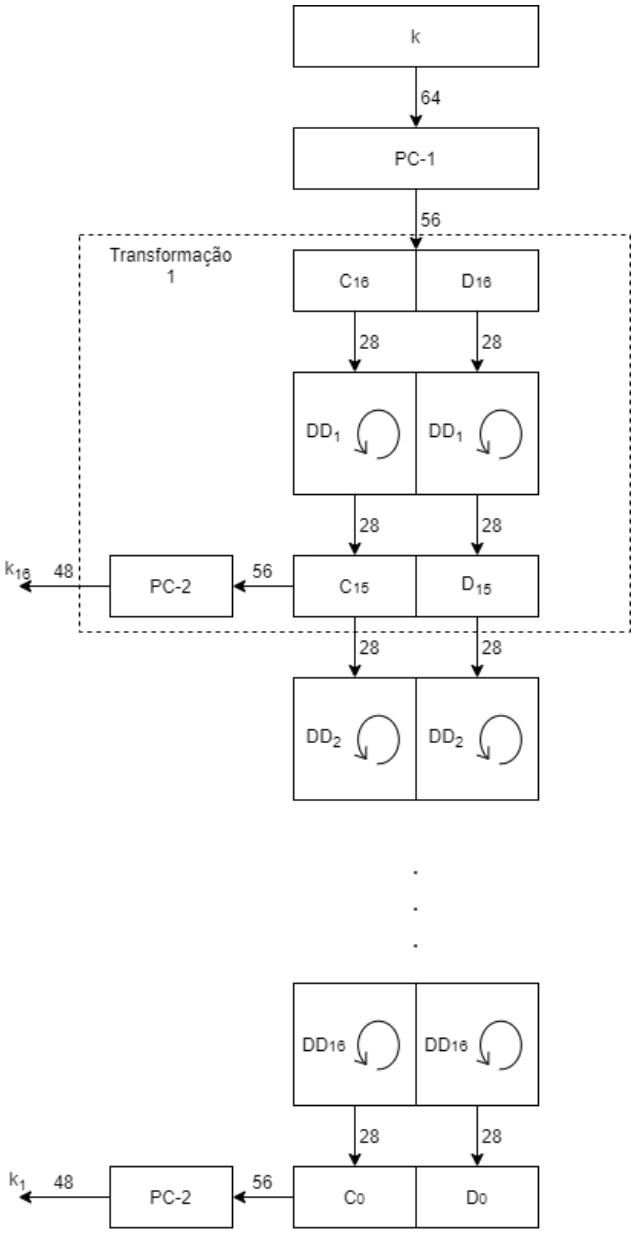
fonte: Próprio autor

2.4 SIMULAÇÃO COMPUTACIONAL

Utilizando o conhecimento adquirido com os estudos acerca da criptografia e decriptografia do DES, foi proposta uma simulação, utilizando a linguagem computacional MATLAB, tanto para criptografia, quanto para decriptografia. Os processos desenvolvidos com a ajuda da linguagem computacional estão apresentados abaixo.

Inicialmente, o DES necessita de duas entradas, que se transformarão no texto criptografado, são elas o texto original, que deve ser enviado por um canal inseguro e necessita ser “mascarado” e a chave-privada, que pode ser enviada por um canal seguro ou então já ser de conhecimento do

Figura 9 – Transformação para Decriptografia



fonte: Próprio Autor

receptor para que o mesmo possa decifrar a informação, através do processo de decriptografia. Para o algoritmo implementado, o texto pode ser de qualquer tipo, desde que esteja contido no Código Padrão Americano para o Intercâmbio de Informação, mais conhecido como Tabela ASCII (*American Standard Code for Information Interchange*). Já a chave é gerada como um vetor aleatório de 64 elementos binários. Outro detalhe importante é que como o DES opera com entradas de 64 bits, para que a simulação funcione como o esperado é necessário que o texto se transforme em vetores de 64 bits. Para tanto, o primeiro passo é a transformação da informação alfanumérica em bits, por intermédio da Tabela ASCII, gerando uma sequência de vetores binários de 64 bits equivalentes à informação. Caso seja necessário, bits 0 são adicionados para completar o último bloco de 64 bits. Apos isso, é necessário transformar essa sequência de elementos binários em matrizes e vetores para que o programa possa ser executado.

Abaixo, estão inseridas as lógicas utilizadas, tanto para criptografia, quanto para decriptografia.

Estas lógicas foram realizadas baseada em [FELIX 2020].

- Criptografia:

```
clc;
clear all;
plain_text = 'Vamos iniciar a criptografia agora';
key = randi([0, 1], 1, 64);
PT_0=dec2bin(plain_text,8)';
PT_1=logical(PT_0=='1');
PT_2=reshape(PT_1,1,numel(PT_1));
PT_3=repmat([0 0 0 0 0 0 0 0],1,(numel(PT_2)+mod(64-mod(numel(PT_2),64),64))/8);
PT_3(1:numel(PT_2))=PT_2;
quant_blocos = length(PT_3)/64;
ct_total = [];
IP    = [ 58    50    42    34    26    18    10     2 ...
          60    52    44    36    28    20    12     4 ...
          62    54    46    38    30    22    14     6 ...
          64    56    48    40    32    24    16     8 ...
          57    49    41    33    25    17     9     1 ...
          59    51    43    35    27    19    11     3 ...
          61    53    45    37    29    21    13     5 ...
          63    55    47    39    31    23    15     7 ];
for i=1:quant_blocos
    pt_round = PT_3(((i-1)*64)+1:i*64);
    IPST = pt_round(IP);



InvIP = [ 40     8    48    16    56    24    64    32 ...
          39     7    47    15    55    23    63    31 ...
          38     6    46    14    54    22    62    30 ...
          37     5    45    13    53    21    61    29 ...
          36     4    44    12    52    20    60    28 ...
          35     3    43    11    51    19    59    27 ...
          34     2    42    10    50    18    58    26 ...
          33     1    41     9    49    17    57    25 ];
PC1C  =   [ 57 49 41 33 25 17  9 ...
             1 58 50 42 34 26 18 ...
            10  2 59 51 43 35 27 ...
            19 11  3 60 52 44 36 ];

  PC1D  =   [ 63 55 47 39 31 23 15 ...
             7 62 54 46 38 30 22 ...
            14  6 61 53 45 37 29 ...
            21 13  5 28 20 12  4 ];
  PC2   =   [ 14 17 11 24  1  5 ...
             3 28 15  6 21 10 ...
            23 19 12  4 26  8 ...
```

```
           16   7 27 20 13  2 ...
           41 52 31 37 47 55 ...
           30 40 51 45 33 48 ...
           44 49 39 56 34 53 ...
           46 42 50 36 29 32  ];

   P32 = [ 16  7 20 21 ...
       29 12 28 17 ...
        1 15 23 26 ...
        5 18 31 10 ...
        2  8 24 14 ...
       32 27  3  9 ...
       19 13 30  6 ...
       22 11  4 25 ];

Left_Shift=[1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1];
C_0 = key(PC1C);
D_0 = key(PC1D);

K = cell(1,16);
for k =1:16
  C=circshift(C_0,-sum(Left_Shift(1:k)));
  D=circshift(D_0,-sum(Left_Shift(1:k)));
  CD=[C D];
  K{k}=CD(PC2);
end

Expn = [ 32  1  2  3  4  5 ...
         4  5  6  7  8  9 ...
         8  9 10 11 12 13 ...
        12 13 14 15 16 17 ...
        16 17 18 19 20 21 ...
        20 21 22 23 24 25 ...
        24 25 26 27 28 29 ...
        28 29 30 31 32  1 ];
 S{1}= [ 14  4 13  1  2 15 11  8  3 10  6 12  5  9  0  7 ...
        0 15  7  4 14  2 13  1 10  6 12 11  9  5  3  8 ...
        4  1 14  8 13  6  2 11 15 12  9  7  3 10  5  0 ...
       15 12  8  2  4  9  1  7  5 11  3 14 10  0  6 13 ];

S{2}= [ 15  1  8 14  6 11  3  4  9  7  2 13 12  0  5 10 ...
        3 13  4  7 15  2  8 14 12  0  1 10  6  9 11  5 ...
        0 14  7 11 10  4 13  1  5  8 12  6  9  3  2 15 ...
       13  8 10  1  3 15  4  2 11  6  7 12  0  5 14  9 ];

S{3}= [ 10  0  9 14  6  3 15  5  1 13 12  7 11  4  2  8 ...
       13  7  0  9  3  4  6 10  2  8  5 14 12 11 15  1 ...
       13  6  4  9  8 15  3  0 11  1  2 12  5 10 14  7 ...
        1 10 13  0  6  9  8  7  4 15 14  3 11  5  2 12 ];
```

```
S{4}= [   7  13  14   3   0   6   9  10   1   2   8   5  11  12   4  15 ...
        13   8  11   5   6  15   0   3   4   7   2  12   1  10  14   9 ...
        10   6   9   0  12  11   7  13  15   1   3  14   5   2   8   4 ...
         3  15   0   6  10   1  13   8   9   4   5  11  12   7   2  14 ];
S{5}= [   2  12   4   1   7  10  11   6   8   5   3  15  13   0  14   9 ...
        14  11   2  12   4   7  13   1   5   0  15  10   3   9   8   6 ...
         4   2   1  11  10  13   7   8  15   9  12   5   6   3   0  14 ...
        11   8  12   7   1  14   2  13   6  15   0   9  10   4   5   3 ];
S{6}= [  12   1  10  15   9   2   6   8   0  13   3   4  14   7   5  11 ...
        10  15   4   2   7  12   9   5   6   1  13  14   0  11   3   8 ...
         9  14  15   5   2   8  12   3   7   0   4  10   1  13  11   6 ...
         4   3   2  12   9   5  15  10  11  14   1   7   6   0   8  13 ];
S{7}= [   4  11   2  14  15   0   8  13   3  12   9   7   5  10   6   1 ...
        13   0  11   7   4   9   1  10  14   3   5  12   2  15   8   6 ...
         1   4  11  13  12   3   7  14  10  15   6   8   0   5   9   2 ...
         6  11  13   8   1   4  10   7   9   5   0  15  14   2   3  12 ];
S{8}= [  13   2   8   4   6  15  11   1  10   9   3  14   5   0  12   7 ...
         1  15  13   8  10   3   7   4  12   5   6  11   0  14   9   2 ...
         7  11   4   1   9  12  14   2   0   6  10  13  15   3   5   8 ...
         2   1  14   7   4  10   8  13  15  12   9   0   3   5   6  11 ];

row=@(plain_text,x) bin2dec(num2str([plain_text(6*x-5) plain_text(6*x)]));
col=@(plain_text,x) bin2dec(num2str([plain_text(6*x-4) plain_text(6*x-3) ...
                                plain_text(6*x-2) plain_text(6*x-1)]));



for n=1:16;
  L = cell(1,16);
  R = cell(1,16);
  L_0 = IPST(1:32);
  R_0 = IPST(33:64);
  L{1} = R_0;


end

  for n=1:16

      if n == 1
          ER=R_0(Expn);
      else
          ER=R{n-1}(Expn);
      end
      KER = xor(K{n},ER);
      SKER_0=ones(1,8);
      for j=1:8
          SKER_0(j)=S{j}(row(KER,j)*16+col(KER,j)+1);
```

```
        end
        SKER_1=dec2bin(SKER_0,4)';
        SKER=logical(reshape(SKER_1,1,32)=='1');
        PSKER=SKER(P32);
        if n==1
            R{1}=xor(L_0,PSKER);
        else
            R{n}=xor(L{n-1},PSKER);
            L{n}=R{n-1};

        end
    end
    assignin('base','Rn',R)
assignin('base','Rn',R)
Pre=[R{16} L{16}];
DESbinary=Pre(InvIP);



    ct_round = DESbinary;

    ct_total = [ct_total ct_round];



end

ct_total_1=bin2dec(ct_total);
ct_total_2=dec2hex(ct_total_1);
```

- Decriptografia:

```
quant_blocos = length(ct_total)/64;
IP    = [ 58    50    42    34    26    18    10     2 ...
          60    52    44    36    28    20    12     4 ...
          62    54    46    38    30    22    14     6 ...
          64    56    48    40    32    24    16     8 ...
          57    49    41    33    25    17     9     1 ...
          59    51    43    35    27    19    11     3 ...
          61    53    45    37    29    21    13     5 ...
          63    55    47    39    31    23    15     7  ];
     ct_total_decrypt= [];
for i=1:quant_blocos
    pt_round = ct_total(((i-1)*64)+1:i*64);
    IPST = pt_round(IP);
    InvIP = [ 40     8    48    16    56    24    64    32 ...
          39     7    47    15    55    23    63    31 ...
          38     6    46    14    54    22    62    30 ...
```

```
          37     5    45    13    53    21    61    29 ...
          36     4    44    12    52    20    60    28 ...
          35     3    43    11    51    19    59    27 ...
          34     2    42    10    50    18    58    26 ...
          33     1    41     9    49    17    57    25 ];
     PC1C = [ 57 49 41 33 25 17  9 ...
               1 58 50 42 34 26 18 ...
              10  2 59 51 43 35 27 ...
              19 11  3 60 52 44 36 ];

PC1D = [ 63 55 47 39 31 23 15 ...
          7 62 54 46 38 30 22 ...
         14  6 61 53 45 37 29 ...
         21 13  5 28 20 12  4 ];
PC2  = [ 14 17 11 24  1  5 ...
          3 28 15  6 21 10 ...
         23 19 12  4 26  8 ...
         16  7 27 20 13  2 ...
         41 52 31 37 47 55 ...
         30 40 51 45 33 48 ...
         44 49 39 56 34 53 ...
         46 42 50 36 29 32 ];

   P32 = [ 16  7 20 21 ...
       29 12 28 17 ...
        1 15 23 26 ...
        5 18 31 10 ...
        2  8 24 14 ...
       32 27  3  9 ...
       19 13 30  6 ...
       22 11  4 25 ];

  Left_Shift=[1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1];
C_0 = key(PC1C);
D_0 = key(PC1D);

K = cell(1,16);
for k =1:16
  C=circshift(C_0,-sum(Left_Shift(1:k)));
  D=circshift(D_0,-sum(Left_Shift(1:k)));
  CD=[C D];
  K{k}=CD(PC2);
end

Expn = [ 32  1  2  3  4  5 ...
         4  5  6  7  8  9 ...
         8  9 10 11 12 13 ...
        12 13 14 15 16 17 ...
        16 17 18 19 20 21 ...
```

```
          20  21  22  23  24  25  ...
          24  25  26  27  28  29  ...
          28  29  30  31  32   1 ];
    S{1}= [ 14   4  13   1   2  15  11   8   3  10   6  12   5   9   0   7 ...
          0  15   7   4  14   2  13   1  10   6  12  11   9   5   3   8 ...
          4   1  14   8  13   6   2  11  15  12   9   7   3  10   5   0 ...
         15  12   8   2   4   9   1   7   5  11   3  14  10   0   6  13 ];

   S{2}= [ 15   1   8  14   6  11   3   4   9   7   2  13  12   0   5  10 ...
          3  13   4   7  15   2   8  14  12   0   1  10   6   9  11   5 ...
          0  14   7  11  10   4  13   1   5   8  12   6   9   3   2  15 ...
         13   8  10   1   3  15   4   2  11   6   7  12   0   5  14   9 ];

   S{3}= [ 10   0   9  14   6   3  15   5   1  13  12   7  11   4   2   8 ...
         13   7   0   9   3   4   6  10   2   8   5  14  12  11  15   1 ...
         13   6   4   9   8  15   3   0  11   1   2  12   5  10  14   7 ...
          1  10  13   0   6   9   8   7   4  15  14   3  11   5   2  12 ];
   S{4}= [  7  13  14   3   0   6   9  10   1   2   8   5  11  12   4  15 ...
         13   8  11   5   6  15   0   3   4   7   2  12   1  10  14   9 ...
         10   6   9   0  12  11   7  13  15   1   3  14   5   2   8   4 ...
          3  15   0   6  10   1  13   8   9   4   5  11  12   7   2  14 ];
   S{5}= [  2  12   4   1   7  10  11   6   8   5   3  15  13   0  14   9 ...
         14  11   2  12   4   7  13   1   5   0  15  10   3   9   8   6 ...
          4   2   1  11  10  13   7   8  15   9  12   5   6   3   0  14 ...
         11   8  12   7   1  14   2  13   6  15   0   9  10   4   5   3 ];
   S{6}= [ 12   1  10  15   9   2   6   8   0  13   3   4  14   7   5  11 ...
         10  15   4   2   7  12   9   5   6   1  13  14   0  11   3   8 ...
          9  14  15   5   2   8  12   3   7   0   4  10   1  13  11   6 ...
          4   3   2  12   9   5  15  10  11  14   1   7   6   0   8  13 ];
   S{7}= [  4  11   2  14  15   0   8  13   3  12   9   7   5  10   6   1 ...
         13   0  11   7   4   9   1  10  14   3   5  12   2  15   8   6 ...
          1   4  11  13  12   3   7  14  10  15   6   8   0   5   9   2 ...
          6  11  13   8   1   4  10   7   9   5   0  15  14   2   3  12 ];
   S{8}= [ 13   2   8   4   6  15  11   1  10   9   3  14   5   0  12   7 ...
          1  15  13   8  10   3   7   4  12   5   6  11   0  14   9   2 ...
          7  11   4   1   9  12  14   2   0   6  10  13  15   3   5   8 ...
          2   1  14   7   4  10   8  13  15  12   9   0   3   5   6  11 ];

   row=@(plain_text,x) bin2dec(num2str([plain_text(6*x-5) plain_text(6*x)]));
   col=@(plain_text,x) bin2dec(num2str([plain_text(6*x-4) plain_text(6*x-3) ...
                                        plain_text(6*x-2) plain_text(6*x-1)]));
for n=1:16;
    L = cell(1,16);
    R = cell(1,16);
    L_0 = IPST(1:32);
    R_0 = IPST(33:64);
    L{1} = R_0;
```

```
end

    for n=1:16


        if n == 1
            ER=R_0(Expn);
        else
            ER=R{n-1}(Expn);
        end
        KER = xor(K{17-n},ER);
        SKER_0=ones(1,8);
        for j=1:8
            SKER_0(j)=S{j}(row(KER,j)*16+col(KER,j)+1);
        end
        SKER_1=dec2bin(SKER_0,4)';
        SKER=logical(reshape(SKER_1,1,32)=='1');
        PSKER=SKER(P32);
        if n==1
            R{1}=xor(L_0,PSKER);
        else
            R{n}=xor(L{n-1},PSKER);
            L{n}=R{n-1};

        end
    end
      assignin('base','Rn',R)
Pre=[R{16} L{16}];
DESbinary=Pre(InvIP);
ct_round_dec = DESbinary;

    ct_total_decrypt = [ct_total_decrypt ct_round_dec];


end
out = char(bin2dec(reshape(char(ct_total_decrypt+'0'), 8,[]).'));
out_final = out.'
```

3 TEA

Como vimos o modelo de criptografia TEA - *Tiny Encryption Algorithm* foi desenvolvido, a partir das redes de Feistel, por Roger M. Needham e David J. Wheeler em [WHEELER David J.; NEEDHAM 1994]. O modelo foi idealizado para atender a alguns problemas encontrados em muitos outros modelos criptográficos anteriores, como o exagerado consumo de memória e baixa velocidade de processamento. A solução encontrada pelos dois pesquisadores foi desenvolver um modelo que utilizasse operações bastante simplificadas, mas que fossem feitas inúmeras vezes, para trazer confiabilidade. Quando se trata de criptografia, o algoritmo do TEA não é um dos mais conhecidos ou mais utilizados, principalmente pelo fato de ser considerado um modelo de criptografia bastante simples, e muitas vezes fácil de ser quebrado a princípio, mas que pode ser uma afirmação equivocada, devido a uma das características mais fortes deste modelo, a velocidade de processamento da criptografia, o que inviabiliza muitas tentativas de quebra de código. Quando a velocidade de processamento é o fator de interesse, o TEA se torna mais eficiente que o AES por exemplo, (*Advanced Encryption Standard*), um dos mais conhecidos do meio [BUDIYANTO D; PUTRO 2018]. Neste capítulo apresentamos o funcionamento da criptografia e da decriptografia do TEA [WHEELER David J.; NEEDHAM 1994], além de uma simulação computacional delas [WHEELER David J.; NEEDHAM 1994].

3.1 CRIPTOGRAFIA

O modelo de criptografia TEA é do tipo cifra simétrica de bloco que utiliza uma chave privada e é baseado em redes de Feistel. O tamanho de bloco para o TEA é de 64 bits para a informação (texto original) que vai ser criptografada, e 128 bits para a chave. Basicamente, as operações de criptografia e decriptografia são a soma e subtração em módulo 2^{32}, XOR e deslocamento de bits, tanto à esquerda DE, quanto à direita DD. Como podemos ver, as operações são semelhantes às utilizadas no DES, porém veremos que com uma estrutura bem mais simples. O algoritmo TEA pode ser definido para um número n de rodadas dependendo da confiabilidade desejada, sendo que quanto maior for o número de repetições, mais segura estará a informação a ser processada porém ao mesmo tempo, o algoritmo requer maior capacidade de processamento e será mais demorado para completar o processo. Durante o desenvolvimento da criptografia [WHEELER David J.; NEEDHAM 1994], os autores sugerem que o TEA seja executado em 32 rodadas, apesar de a partir de 6 rodadas eles já considerarem que o algoritmo possui um desempenho razoável. Para aplicações em RFID ("*Radio-Frequency IDentification*"), por exemplo, em [ABDELHALIM M.; EL-MAHALLAWY 2013], foram utilizadas 50 rodadas para testar a viabilidade do algortimo.

A operação de soma módulo 2^{32}, denotada por $\boxplus$, ou subtração módulo 2^{32}, denotada por $\boxminus$, faz a operação normalmente entre os números em questão e retorna o resto da divisão do resultado por 2^{32}. Esta operação se torna importante já que o TEA muitas vezes requer a soma ou subtração entre dois blocos, porém este tipo de operação entre blocos de mesmo tamanho pode muitas vezes aumentar em 1 bit o tamanho da resposta, que é algo inviável neste caso. Então, o que a operação módulo 32 propõe no caso do TEA é garantir que todos os blocos tenham 32 bits. Pois neste caso, se em alguma

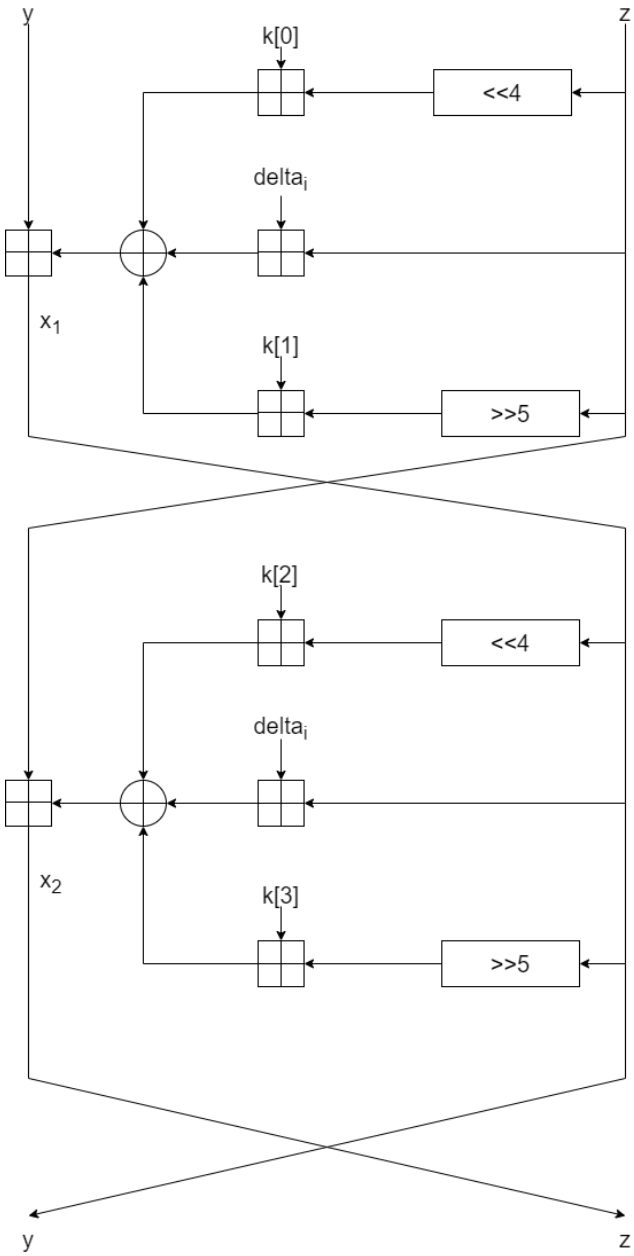
operação a resposta tiver tamanho de 33 bits ou mais, a operação módulo 32 retorna o resto da divisão por 32 [APOSTOL 1976].

O algoritmo TEA utiliza também uma constante, chamada de constante $delta$, que é derivada da proporção do número de ouro e é utilizada para garantir que as subchaves sejam distintas e que seu valor preciso não tenha significado criptográfico [ANDEM 2003]. Esta constante é dada por

$$delta = (\sqrt{5} - 1) \times 2^{31} \tag{3.1}$$

em módulo 2^{32}, ou 2654435769 na base decimal, ou ainda, $9E3779B9$, na base hexadecimal.

Figura 10 – Criptografia TEA



fonte: Próprio Autor

A Figura 10, mostra de forma detalhada o funcionamento de uma rodada da criptografia do TEA. A função rodada f neste caso, característica das redes de Feistel, realiza praticamente todo o processo da

rodada. Inicialmente o texto original x, com 64 bits, a ser criptografado é dividido em duas partes de 32 bits, y do primeiro ao trigésimo segundo bit da sequência e z do trigésimo terceiro ao sexagésimo quarto bit da sequência. A chave k que possui 128 bits, também deve ser segmentada em blocos de 32 bits. As subchaves de 32 bits são chamadas de $k(0)$, $k(1)$, $k(2)$ e $k(3)$.

A primeira operação a ser realizada no TEA é o deslocamento de bits no bloco z. Serão realizados dois tipos de deslocamentos: o deslocamento à esquerda DE de 4 bits, que para facilitar a notação será representado por $<< 4$ e, o deslocamento à direita DD de 5 bits, representado por $>> 5$. O bloco z deslocado $<< 4$ é somado módulo 2^{32} com $k(0)$, e z deslocado $>> 5$ é somado módulo 2^{32} com $k(1)$, e os resultados são armazenados. O mesmo bloco z é ainda somado módulo 2^{32} com uma constante advinda da constante $delta$, sendo que nesta primeira rodada

$$delta_1 = delta = (\sqrt{5} - 1) \times 2^{31}, \tag{3.2}$$

como em (3.1) e, nas demais rodadas,

$$delta_i = delta + delta_{i-1}, \tag{3.3}$$

onde $i = 2, \ldots, n$. Com estes três valores armazenados, é realizado a operação XOR, e o resultado é somado módulo 2^{32} com o bloco y. Deste procedimento, é obtido o bloco x_1, ou seja,

$$x_1 = y \boxplus f(z, delta_1, k_0, k_1) = y \boxplus ((z << 4) \boxplus k_0) \oplus (z \boxplus delta_1) \oplus ((z >> 5) \boxplus k_1). \tag{3.4}$$

Com x_1 também serão realizados o deslocamento $<< 4$ que será somado módulo 2^{32} com $k(2)$, o deslocamento $>> 5$ que será somado módulo 2^{32} com $k(3)$ e a soma módulo 2^{32} com a constante $delta$ dada em (3.7), armazenando mais uma vez as respostas. Então, a operação XOR é aplicada a estes três valores armazenados, e o resultado, é somado módulo 2^{32} com o bloco z inicial, obtendo então o bloco x_2, ou seja,

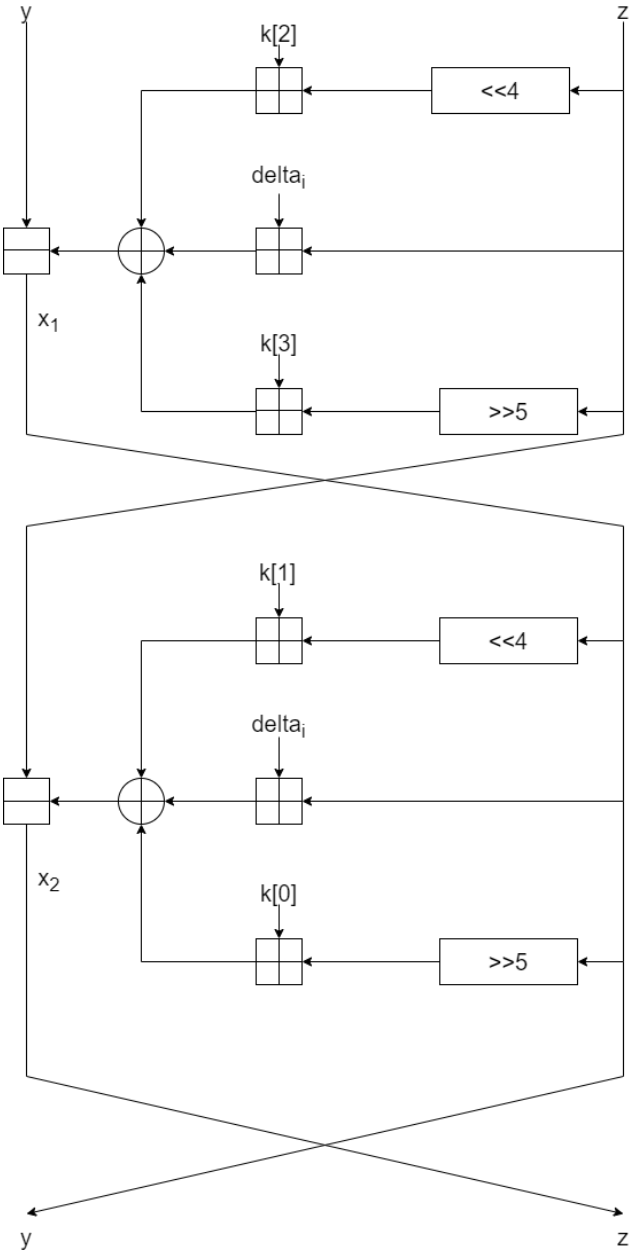
$$x_2 = z \boxplus f(x_1, delta_1, k_2, k_3) = z \boxplus ((x_1 << 4) \boxplus k_2) \oplus (x_1 \boxplus delta_1) \oplus ((x_1 >> 5) \boxplus k_3). \tag{3.5}$$

Ao final destes passos os valores de x_1 e x_2 passam a ser as novas entradas y e z, respectivamente, da próxima rodada [Hunn S.A.Y.; Naziri 2012]. Este processo é repetido nas n rodadas do funcionamento da criptografia do TEA, sendo que os valores de x_1 e x_2 encontrados na última rodada formam o texto cifrado quando concatenados.

3.2 DECRIPTOGRAFIA

A decriptografia, pelo fato do TEA ser um modelo baseado em redes de Feistel, segue a mesma lógica do modelo de criptografia, apenas invertendo a ordem das operações, ou seja, a última função a aparecer na criptografia, será a primeira a ocorrer na decriptografia, assim como a primeira função da criptografia será a última da decriptografia, e assim segue o modelo. A Figura 11 representa o modelo de decriptografia.

Figura 11 – Decriptografia TEA



fonte: Próprio Autor

Para a decriptografia, o bloco z sofre, inicialmente, 2 tipos de deslocamento, um DE de 4 bits e outro, DD de 5 bits. O bloco deslocado $<< 4$ é somado em módulo 2^{32} com $k(2)$, e o bloco deslocado $>> 5$ é somado em 2^{32} com $k(3)$ e os resultados armazenados. O mesmo bloco também é somado em módulo 2^{32} com a constante $delta$. Após 32 rodadas de criptografia, a constante $delta$ acumulou em 32 vezes o seu valor inicial. Mas ao invés de multiplicar em 32 vezes seu valor, pode-se simplesmente deslocar o valor de $delta_1$ 5 bits à esquerda. Então, teremos:

$$delta_1 = delta << 5, \tag{3.6}$$

E nas rodadas seguintes, teremos:

$$delta_i = delta_{i-1} - delta \tag{3.7}$$

onde $i = 2, \ldots, n$. Utilizando os três vetores armazenados, é realizada uma XOR entre eles, e seu resultado, subtraído de y, em uma operação 2^{32}. Ao fim deste processo, temos,

$$x_1 = y \boxminus f(z, delta_1, k_2, k_3) = y \boxminus ((z << 4) \boxplus k_2) \oplus (z \boxplus delta_1) \oplus ((z >> 5) \boxplus k_3). \tag{3.8}$$

Com x_1, também será realizado o deslocamento $<< 4$, somando em módulo 2^{32} com $k(1)$, e o deslocamento $>> 5$, somando em módulo 2^{32} com $k(0)$, além da soma em módulo 2^{32} com a constante $delta$. Com as três respostas obtidas, faremos um XOR, e o resultado, subtraído de z inicial em módulo 2^{32}, obtendo o bloco x_2, a seguir,

$$x_2 = z \boxminus f(x_1, delta_1, k_0, k_1) = z \boxminus ((x_1 << 4) \boxplus k_1) \oplus (x_1 \boxplus delta_1) \oplus ((x_1 >> 5) \boxplus k_0). \tag{3.9}$$

Em seguida, os blocos x_1 e x_2 se tornam y e z, respectivamente, para iniciar a próxima rodada. Este processo é repetido n vezes, até que os últimos valores de x_1 e x_2, sejam concatenados, obtendo novamente o texto original.

3.3 SIMULAÇÃO COMPUTACIONAL

A seguir apresentamos a implementação computacional desenvolvida para a simulação do TEA, utilizando um gerador aleatório de números binários, pois quando a mesma for utilizada para futuras simulações de desempenho, os resultados serão mais homogêneos.

```
function enc = TEA_ENC(txt,key)


delta = 2654435769; %constante delta em decimal
sum=0;
k0=uint32(key(1:32));       %primeira parte da chave
k1=uint32(key(33:64));      %segunda parte da chave
k2=uint32(key(65:96));      %terceira parte da chave
k3=uint32(key(97:128));     %quarta parte da chave
delta = uint32(dec2bin(delta))-uint32('0');     %constante delta em bin rio
ct_total_enc=[];
for i=1:size(txt,1)
    v0 = txt(i,1:32);     %divis o em do texto em bin rio para v0 e v1
    v1 = txt(i,33:64);

    for N=1:32
        sum=sum+delta;
```

```
        s1 = circshift(v1,-4);   %deslocamento    esquerda de v1
        s2 = v1;
        s3 = circshift(v1,5);    %deslocamento    direita de v1
        s11 = soma_quadrado(s1,k0);   %soma entre s1 e e k0
        s21 = soma_quadrado(s2,delta);  %soma entre delta e s2
        s31 = soma_quadrado(s3,k1);    %soma entre s3 e k1


        s22=bitxor(bitxor(s11,s21),s31);      %bitxor de s11, s21 e s31

        x1 = soma_quadrado(s22,v0);    %soma entre s22 e v0


        t1=circshift(x1,-4);          %deslocamento    esquerda de x1
        t2=x1;
        t3=circshift(x1,5);             %deslocamento    direita de x1

        t11=soma_quadrado(t1,k2);       % soma entre t1 e k2
        t21=soma_quadrado(t2,delta);    %soma entre t2 e delta
        t31=soma_quadrado(t3,k3);       %soma entre t3 e k3

        t22=bitxor(bitxor(t11,t21),t31);   %bitxor entre t11, t21 e t31
        x2=soma_quadrado(t22,v1);           %soma entre t22 e v1


        v0=x1;
        %no final v0 assume valor de x1 e v1 o valor de x2
        v1=x2;
    end
    v= [v0 v1];



end

ct_total_enc = [ct_total_enc v];
enc = vec2mat(ct_total_enc, 64);
end
```

4 ANÁLISE DE DESEMPENHO

A fim de mostrar a efetividade do TEA, apresentamos uma análise de desempenho a partir das técnicas de difusão e confusão propostas por Shannon [SHANNON 1949], que são propriedades para criar uma cifra segura no sentido de ocultar as características estatísticas do texto original e da chave de modo que estes não possam ser deduzidos. Shannon estava preocupado com a criptoanálise, pois dependendo da análise estatística do texto simples, em que algumas mensagens têm uma distribuição de frequência, como letras, palavras ou frases, que podem ser transferidas para o texto cifrado na mesma distribuição de frequência. Se o invasor tiver algum conhecimento sobre essas estatísticas, talvez ele possa obter a chave de criptografia usada para criptografar o texto original. Essas operações sozinhas, não garantem segurança. No entanto, quando relacionadas, uma cifra forte pode ser construída. Praticamente todas as cifras de blocos utilizadas hoje possuem as características de difusão e confusão de Shannon, porém o interessante no TEA é que estas propriedades são alcançadas de forma completa a partir de seis rodadas e sem necessitar de técnicas elaboradas como o uso de P-boxes e S-boxes [WHEELER David J.; NEEDHAM 1994,ISRASENA 2005]. Ainda no sentido de que se a aleatoriedade for ruim então a criptoanálise pode fazer previsões sobre o texto original, podemos definir esta aleatoriedade utilizando o efeito avalanche, que foi introduzido por [FEISTEL 1973]. Assim como a difusão e confusão, o efeito avalanche é uma propriedade desejável dos algoritmos criptográficos, em especial das cifras de bloco. No caso de cifras de bloco de alta qualidade, uma alteração tão pequena na chave ou no texto original deve causar uma mudança drástica no texto cifrado. Uma desvantagem (que é a vantagem da criptografia) da difusão e confusão é a propagação de erros: um pequeno erro no texto cifrado se torna um erro grave na mensagem decriptografada e geralmente significa que a decriptografia é ilegível [TRAPPE W.; WASHINGTON 2006]. Sendo assim, iremos inicialmente, analisar os efeitos de difusão e confusão no TEA considerando 5, 32 e 50 rodadas. Em seguida, a partir dos resultados, mostraremos que de fato o efeito avalanche é uma característica do TEA.

4.1 DIFUSÃO

Dizemos que um modelo de criptografia com um texto original, que será criptografado em um texto cifrado, e uma chave que será utilizada no processo, possui a propriedade da difusão se, ao modificar um único bit do texto original, metade dos bits do texto criptografado for modificado, sem realizar a troca da chave [TRAPPE W.; WASHINGTON 2006]. Isso significa que as estatísticas de frequência no texto original são difundidas por vários bits no texto cifrado, o que dificulta um ataque estatístico significativo.

A partir do conceito de difusão realizamos uma modelagem computacional para verificar que de fato o TEA se enquadra nesta propriedade, e a partir de quantas rodadas e repetições obtemos resultados satisfatórios. Utilizando o algoritmo de encriptação implementado para o TEA no Capítulo 3, podemos obter a partir de uma informação (texto original) e uma chave pré-definida, o texto cifrado. A partir de então, os bits do texto original foram sendo modificados um a um (se fosse 0 passava a ser 1 e se fosse 1 passava a ser 0), e para cada um deles foi gerado novamente o texto cifrado. Após a troca

dos 64 bits, 64 textos cifrados foram gerados e cada um deles comparado com o texto cifrado gerado inicialmente a partir do texto original sem a troca de bits.

Iremos exemplificar as trocas de bits do texto original e os efeitos no texto cifrado, a partir de um texto original e uma chave pré-definidos. Consideramos neste exemplo o TEA com $n = 32$ rodadas e $m = 1$ repetição. Para uma melhor vizualização, iremos representar o texto cifrado com 64 bits em uma matriz X de dimensão 4×16.

$$X = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}. \tag{4.1}$$

Para exemplificar algumas mudanças de bits no texto original e seu efeito no respectivo texto cifrado, a seguir apresentamos as matrizes X_1, X_{23} e X_{60} que representam os textos cifrados após modificar o primeiro bit, o vigésimo terceiro e o sexagésimo bit no texto original, respectivamente. As mudanças dos bits em relação a matriz X dada em (4.1) estão em negrito.

$$X_1 = \begin{pmatrix} 1 & \mathbf{0} & \mathbf{0} & 1 & 1 & \mathbf{0} & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{0} & 1 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 0 & 1 & 0 & 0 & 0 & \mathbf{1} & 1 & \mathbf{1} & \mathbf{0} & 1 & \mathbf{1} & 1 \\ \mathbf{1} & 0 & \mathbf{1} & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & 1 & \mathbf{0} & 0 & 0 & \mathbf{1} \\ 0 & 1 & 1 & 1 & \mathbf{0} & 1 & \mathbf{0} & 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 & 1 & 0 \end{pmatrix}; \tag{4.2}$$

$$X_{23} = \begin{pmatrix} 1 & 1 & \mathbf{0} & 1 & 1 & \mathbf{0} & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 1 & 1 \\ 0 & 1 & \mathbf{0} & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & \mathbf{0} & \mathbf{1} & 1 \\ \mathbf{1} & 0 & \mathbf{1} & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & 1 & 1 & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 1 & 1 & \mathbf{0} & 0 & \mathbf{0} & \mathbf{1} & 1 & \mathbf{1} \end{pmatrix}; \tag{4.3}$$

$$X_{60} = \begin{pmatrix} \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 0 & \mathbf{1} & \mathbf{1} & 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \\ 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & \mathbf{1} & 1 & 1 & \mathbf{1} & \mathbf{0} & 1 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & \mathbf{0} & 0 & 1 & \mathbf{1} & 1 & \mathbf{1} \end{pmatrix}. \tag{4.4}$$

Podemos observar em (4.2) que ao trocar o primeiro bit no texto original, 32 bits foram trocados no texto cifrado. Em (4.3) que 27 bits foram trocados e em (4.4) que 41 bits foram trocados. Na Tabela 4 apresentamos a quantidade de bits modificados $\#i$ no texto cifrado após a troca do bit i no texto original, para todo $i = 1, \ldots, 64$. Como podemos observar Tabela 4, a menor quantidade de bits modificados é 25 obtida com a troca do bit 40 e a maior quantidade de bits é 41 obtida com a troca do bit 60. Se calcularmos a média dos resultados, ela estará por volta de 50% dos bits alterados porém como podemos observar não temos uma boa distribuição nos dados.

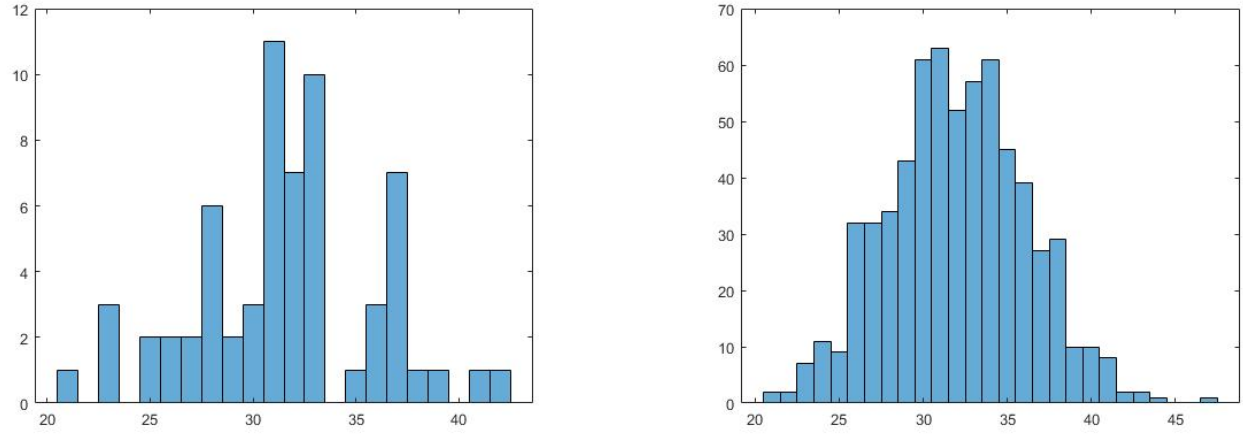
Tabela 4 – Quantidade de bits modificados no texto cifrado - Difusão

bit i	$\#i$	**bit** i	$\#i$	**bit** i	$\#i$	**bit** i	$\#i$
1	32	**17**	32	**33**	31	**49**	31
2	34	**18**	32	**34**	26	**50**	29
3	28	**19**	32	**35**	33	**51**	28
4	27	**20**	30	**36**	30	**52**	29
5	28	**21**	30	**37**	32	**53**	28
6	34	**22**	31	**38**	32	**54**	32
7	34	**23**	27	**39**	33	**55**	29
8	35	**24**	31	**40**	25	**56**	30
9	29	**25**	35	**41**	28	**57**	36
10	36	**26**	35	**42**	34	**58**	36
11	37	**27**	27	**43**	35	**59**	32
12	34	**28**	39	**44**	29	**60**	41
13	33	**29**	36	**45**	30	**61**	35
14	29	**30**	34	**46**	34	**62**	29
15	34	**31**	31	**47**	37	**63**	34
16	30	**32**	34	**48**	33	**64**	28

fonte: Próprio Autor

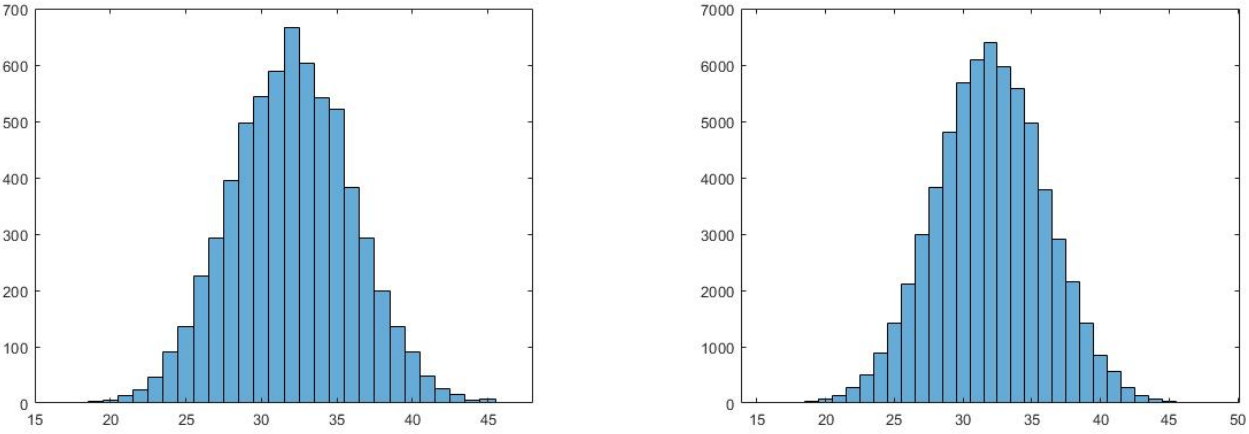
Para maior uma melhor análise dos resultados, as simulações das trocas de bits no texto original foram realizadas considerando $n = 5, 32$ e 50 rodadas e utilizando $m = 1, 10, 100$ e 1000 repetições no algoritmo. Para cada um dos casos, inicialmente foram geradas uma chave e um texto aleatório, para gerar resultados mais confiáveis.

Os histogramas da Figura 12 e da Figura 13 representam as quantidades de trocas de bits no texto original para $n = 5$ rodadas variando $m = 1, 10, 100$ e 1000 repetições. Assim como os histogramas da Figura 14 e da Figura 15 representam as quantidades de trocas de bits no texto original para $n = 32$ rodadas variando $m = 1, 10, 100$ e 1000 repetições. E por fim, os histogramas da Figura 16 e da Figura 17 representam as quantidades de trocas de bits no texto original para $n = 50$ rodadas variando $m = 1, 10, 100$ e 1000 repetições. Neles, podemos observar que o eixo x representa o número de bits modificados no texto original para um bloco de 64 bits, e o eixo y representa a frequência com que ocorre cada número. Em todos os casos analisados, podemos observar que a partir de $m = 100$ as maiores frequências ocorrem em torno de 32 bits modificados no texto original, ou seja, 50% dos bits, confirmando então a ocorrência da difusão no TEA mesmo que em poucas rodadas do algoritmo (para $n = 5$). Além disso, os histogramas para $m = 100$ e 1000 podem ser aproximados por uma distribuição gaussiana com média 32, como desejado.



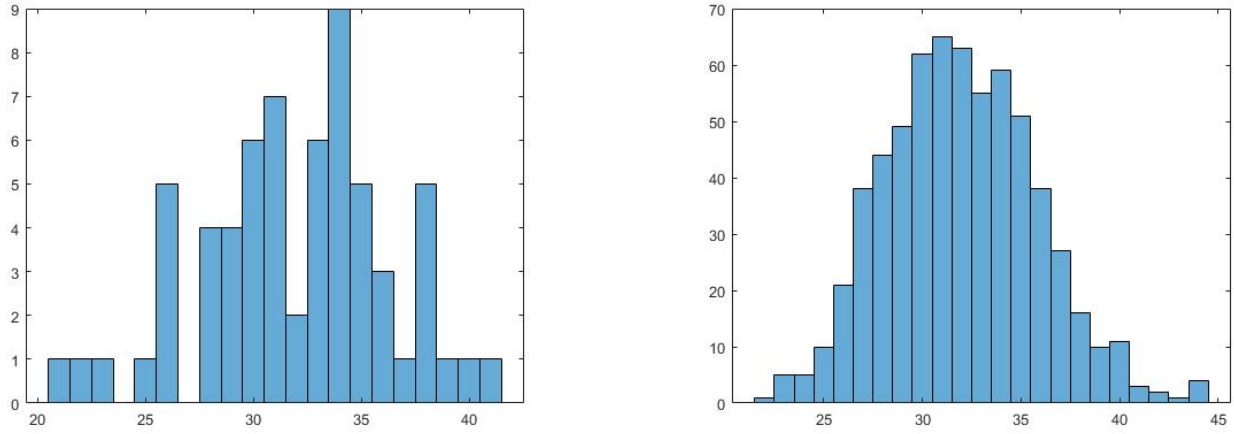
(a) m=1 (b) m=10

Figura 12 – Histograma da difusão para n=5 rodadas e m=1 e m=10 repetições.



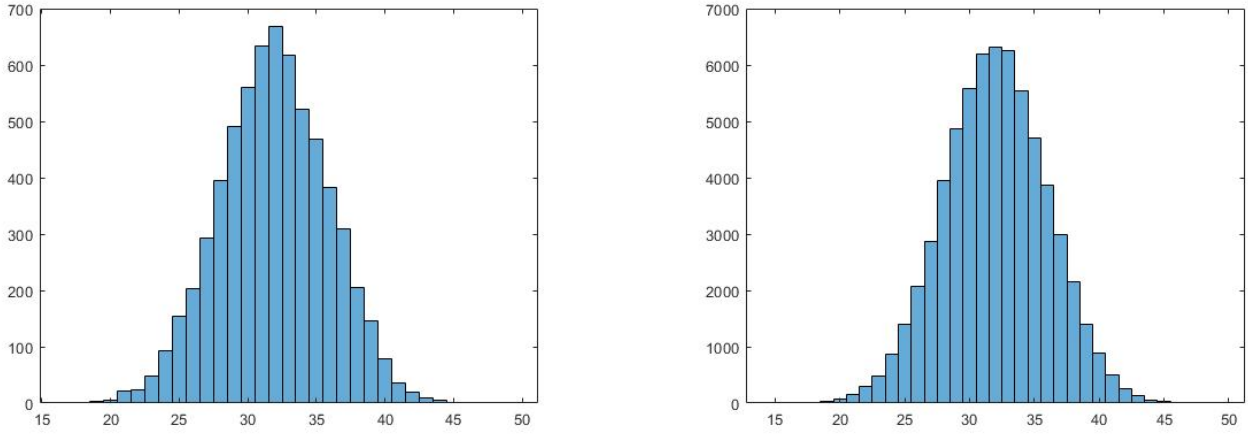
(a) m=100 (b) m=1000

Figura 13 – Histograma da difusão para n=5 rodadas e m=100 e m=1000 repetições.



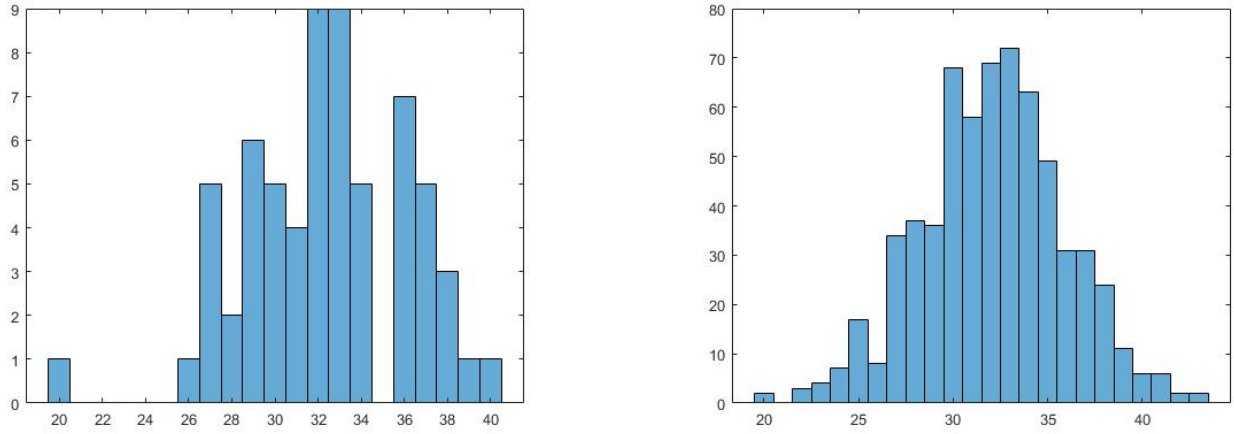
(a) m=1 (b) m=10

Figura 14 – Histograma da difusão para n=32 rodadas e m=1 e m=10 repetições.



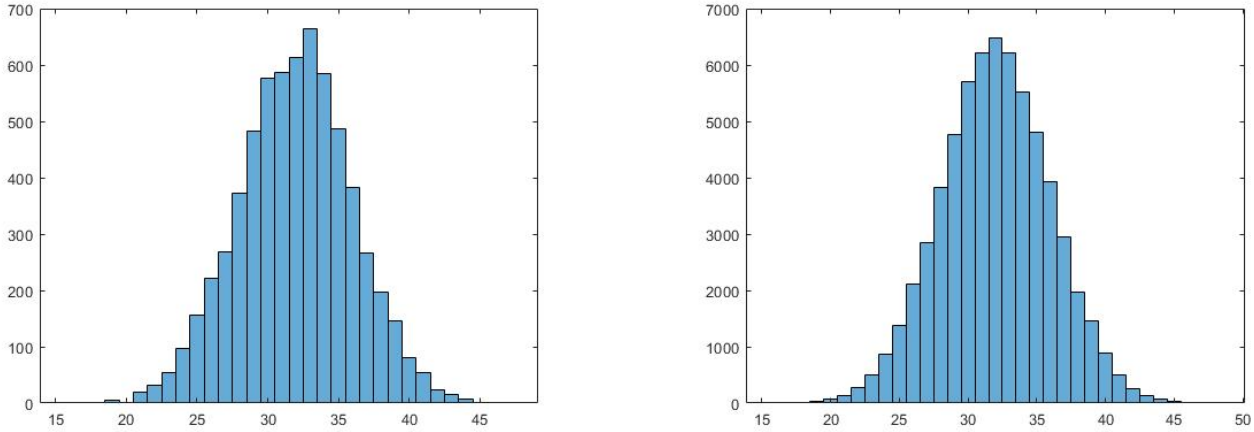
(a) m=100 (b) m=1000

Figura 15 – Histograma da difusão para n=32 rodadas e m=100 e m=1000 repetições.



(a) m=1 (b) m=10

Figura 16 – Histograma da difusão para n=50 rodadas e m=1 e m=10 repetições.



(a) m=100 (b) m=1000

Figura 17 – Histograma da difusão para n=50 rodadas e m=100 e m=1000 repetições.

4.2 CONFUSÃO

A propriedade da confusão diz que cada bit do texto cifrado deve depender de toda a chave, e de maneiras diferentes em diferentes bits da chave. O objetivo da confusão é tornar a relação de análise estatística entre o texto cifrado e a chave de criptografia a mais complexa possível. Essa propriedade dificulta a localização da chave no texto cifrado e, se um único bit da chave for modificado, boa parte do texto criptografado também será modificado [TRAPPE W.; WASHINGTON 2006]. A confusão torna muito difícil para o criptoanalista encontrar a chave, pois quando uma cifra possui essa propriedade, o criptoanalista provavelmente precisaria resolver a chave inteira simultaneamente.

A partir do conceito de confusão e utilizando a implementação computacional para encriptação do TEA apresentada no Capítulo 3, realizamos uma análise para mostrar que de fato o TEA possui esta propriedade, assim como fizemos para a difusão. Utilizando um gerador de números aleatórios para o texto original e para chave obtemos, através o algoritmo implementado, o texto cifrado. Após isso, armazenamos as informações do texto original e da chave para inciar a troca de bits da chave. Os bits da chave foram sendo modificados um a um, e para cada um deles foi gerado novamente o texto cifrado. Após a troca dos 128 bits, 128 textos cifrados foram gerados e cada um deles comparado com o texto cifrado gerado a partir do texto original inicialmente armazenado e da chave sem a troca de bits.

Iremos exemplificar as trocas de bits da chave e os efeitos no texto cifrado, a partir de um texto original e uma chave pré-definidos. Consideramos neste exemplo o TEA, com $n = 32$ rodadas e $m = 1$ repetição. Novamente, para uma melhor vizualização, iremos representar o texto cifrado com 64 bits em uma matriz X de dimensão 4×16.

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}. \tag{4.5}$$

Para exemplificar algumas mudanças de bits na chave e seu efeito no respectivo texto cifrado, a seguir apresentamos as matrizes X_1, X_{57} e X_{76}, que representam os textos cifrados após modificar o primeiro bit, o quinquagésimo sétimo bit e o setuagésimo sexto bit na chave, respectivamente. As mudanças dos bits em relação a matriz X dada em (4.5) estão em negrito.

$$X_1 = \begin{pmatrix} \mathbf{1} & 1 & 0 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 & \mathbf{0} & \mathbf{1} & 1 \\ 1 & \mathbf{0} & 0 & 1 & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 1 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} \\ 1 & \mathbf{1} & 1 & 1 & \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 & 0 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 0 & \mathbf{0} & \mathbf{1} & 1 & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 0 \end{pmatrix}; \tag{4.6}$$

$$X_{62} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 1 & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 1 & 1 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & 1 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\ 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 \\ \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & \mathbf{1} & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & 1 & \mathbf{0} & 0 & \mathbf{1} \end{pmatrix}; \tag{4.7}$$

$$X_{81} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 1 & \mathbf{0} & 0 & 1 \\ 1 & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & 1 & 1 & \mathbf{0} & 0 & 1 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\ 1 & 0 & \mathbf{0} & 1 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 & 1 & \mathbf{0} & 1 & 0 & \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 0 & 1 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \mathbf{1} \end{pmatrix}; \tag{4.8}$$

Podemos observar em (4.5) que ao trocar o primeiro bit no texto original, 36 bits foram trocados no texto cifrado, em (4.7) que 41 bits foram trocados e em (4.8) que 21 bits foram trocados. Na Tabela 5 apresentamos a quantidade de bits modificados $\#i$ no texto cifrado após a troca do bit i na chave, para todo $i = 1, \ldots, 128$. Como podemos observar Tabela 5, a menor quantidade de bits modificados é 21 obtida com a troca do bit 81 e a maior quantidade de bits é 41 obtida com a troca do bit 62. Se calcularmos a média dos resultados, assim como na difusão, ela estará também por volta de 50% dos bits alterados, porém como podemos observar não temos uma boa distribuição nos dados, já que existe uma boa variação da quantidade de bits modificados.

Tabela 5 – Quantidade de bits modificados no texto cifrado - Confusão

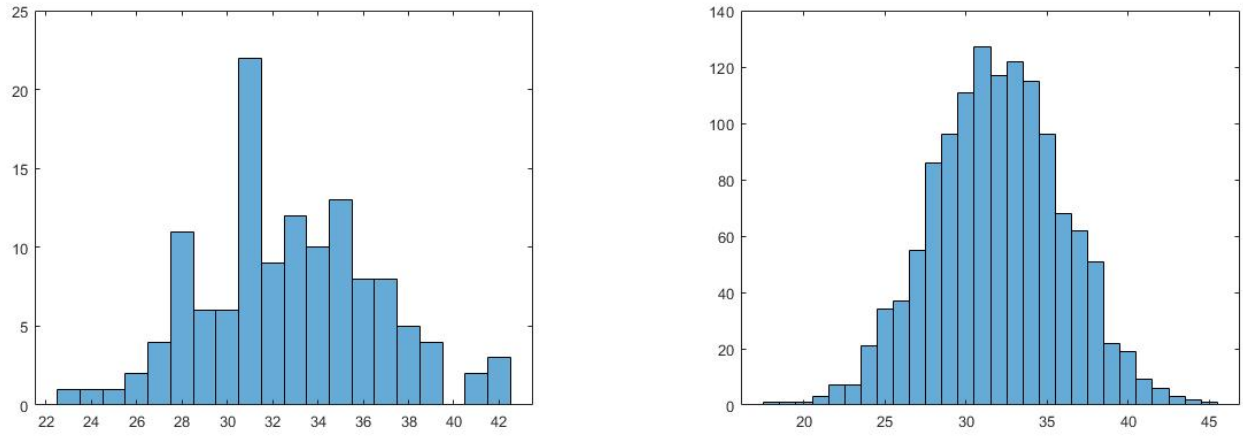
bit i	$\#i$	**bit** i	$\#i$	**bit** i	$\#i$	**bit** i	$\#i$	bit i	$\#i$	**bit** i	$\#i$	**bit** i	$\#i$	**bit** i	$\#i$
1	36	**17**	34	**33**	36	**49**	30	**65**	30	**81**	21	**97**	28	**113**	34
2	32	**18**	30	**34**	36	**50**	31	**66**	30	**82**	30	**98**	32	**114**	29
3	38	**19**	25	**35**	32	**51**	36	**67**	34	**83**	33	**99**	31	**115**	30
4	28	**20**	34	**36**	33	**52**	35	**68**	26	**84**	30	**100**	29	**116**	34
5	26	**21**	32	**37**	34	**53**	31	**69**	31	**85**	31	**101**	22	**117**	32
6	33	**22**	39	**38**	34	**54**	38	**70**	33	**86**	36	**102**	33	**118**	30
7	36	**23**	27	**39**	30	**55**	33	**71**	37	**87**	34	**103**	24	**119**	38
8	36	**24**	37	**40**	31	**56**	33	**72**	37	**88**	25	**104**	25	**120**	27
9	30	**25**	23	**41**	32	**57**	32	**73**	34	**89**	33	**105**	33	**121**	32
10	25	**26**	23	**42**	34	**58**	31	**74**	32	**90**	34	**106**	34	**122**	28
11	33	**27**	34	**43**	33	**59**	32	**75**	39	**91**	31	**107**	39	**123**	33
12	27	**28**	31	**44**	30	**60**	38	**76**	28	**92**	34	**108**	38	**124**	39
13	36	**29**	34	**45**	31	**61**	32	**77**	29	**93**	24	**109**	31	**125**	37
14	33	**30**	32	**46**	32	**62**	41	**78**	37	**94**	29	**110**	31	**126**	37
15	33	**31**	35	**47**	34	**63**	31	**79**	28	**95**	30	**111**	35	**127**	34
16	38	**32**	29	**48**	33	**64**	29	**80**	34	**96**	36	**112**	31	**128**	27

fonte: Próprio Autor

Para maior uma melhor análise dos resultados, as simulações das trocas de bits na chave foram realizadas considerando novamente $n = 5, 32$ e 50 rodadas e utilizando $m = 1, 10, 100$ e 1000 repetições no algoritmo. Para cada um dos casos, inicialmente foram geradas uma chave e um texto aleatório, para gerar resultados mais confiáveis.

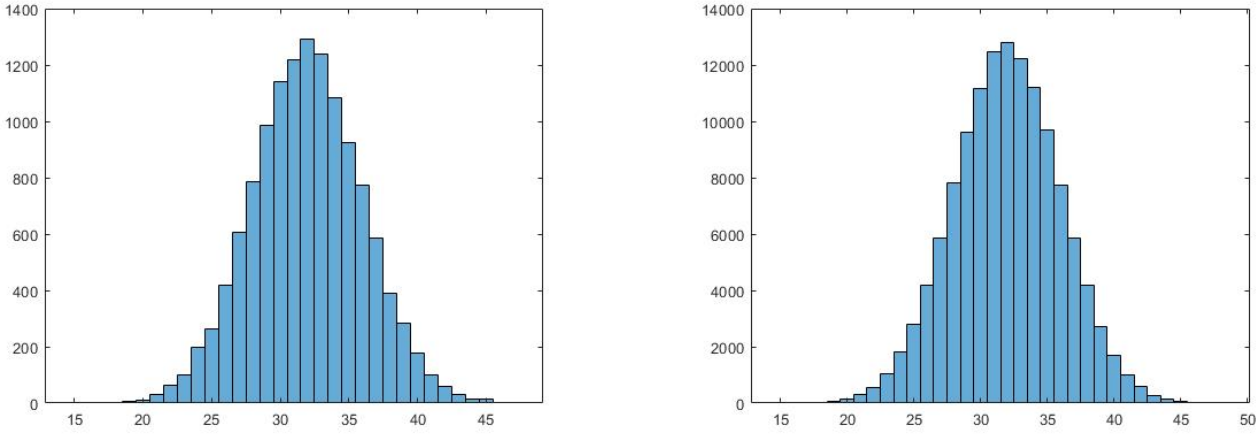
Os histogramas da Figura 18 e da Figura 19 representam as quantidades de trocas de bits na chave para $n = 5$ rodadas, variando $m = 1, 10, 100$ e 1000 repetições. Assim como os histogramas da Figura 20 e da Figura 21 representam as quantidades de trocas de bits na chave para $n = 32$ rodadas, variando $m = 1, 10, 100$ e 1000 repetições. E por fim, os histogramas da Figura 22 e da Figura 23 representam as quantidades de trocas de bits da chave para $n = 50$ rodadas, variando $m = 1, 10, 100$ e

1000 repetições. Neles, podemos observar que o eixo x representa o número de bits modificados na chave para um bloco de 128 bits, e o eixo y representa a frequência com que ocorre cada número. Em todos os casos analisados, podemos observar que a partir de $m = 100$ as maiores frequências ocorrem em torno de 32 bits modificados na chave, ou seja, 50% dos bits, confirmando então a ocorrência da confusão no TEA, mesmo que em poucas rodadas do algoritmo (para $n = 5$). Além disso, os histogramas para $m = 100$ e 1000 podem ser aproximados por uma distribuição gaussiana com média 32, como desejado.



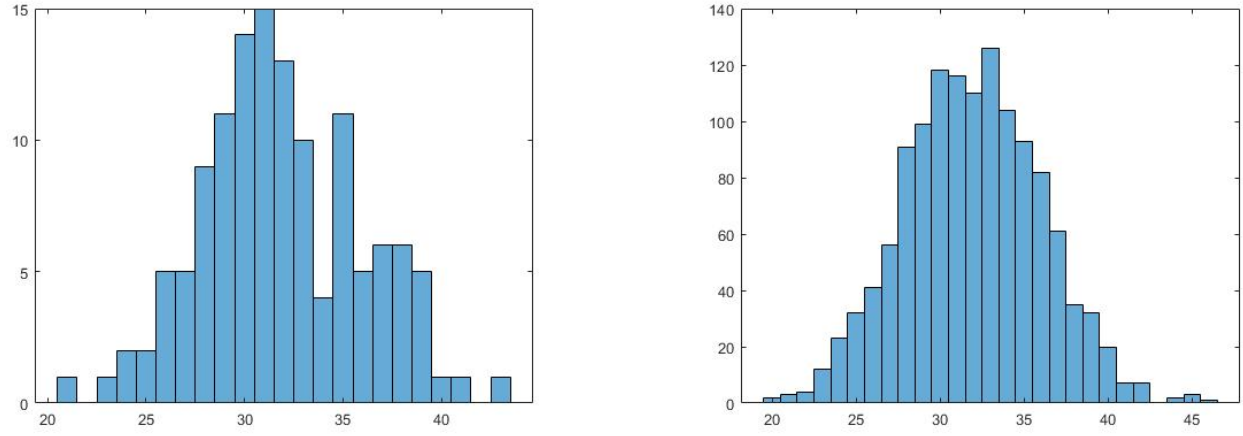
(a) m=1 (b) m=10

Figura 18 – Histograma da confusão para n=5 rodadas e m=1 e m=10 repetições.



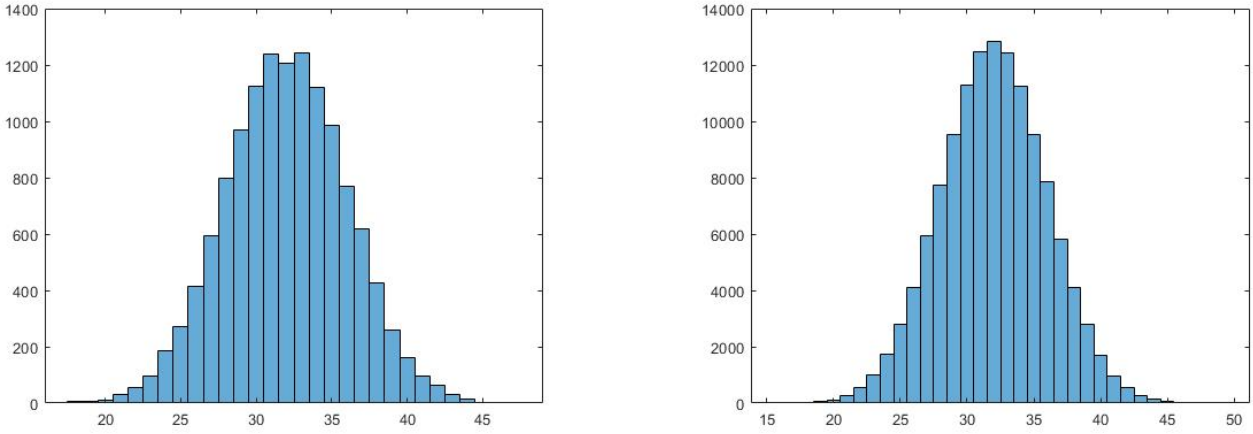
(a) m=100 (b) m=1000

Figura 19 – Histograma da confusão para n=5 rodadas e m=100 e m=1000 repetições.



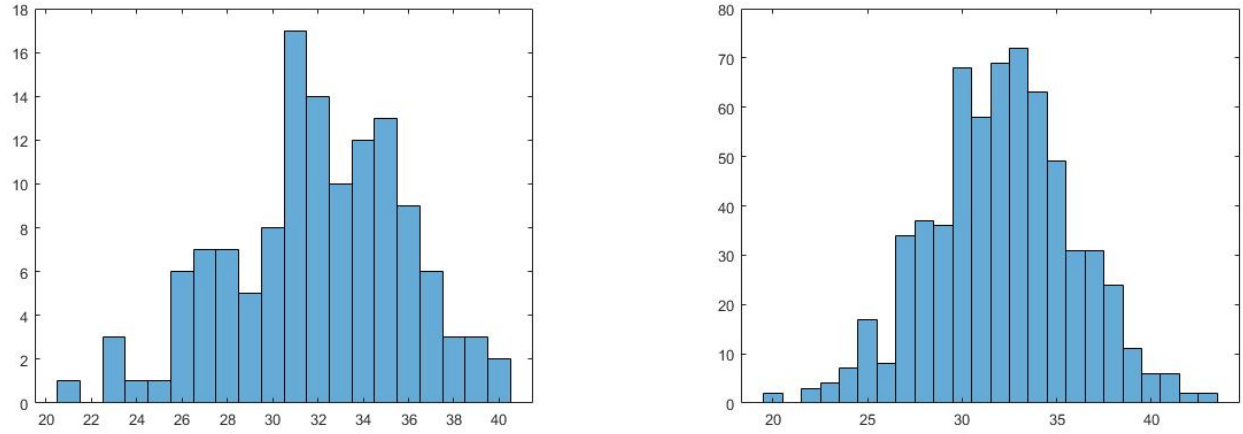
(a) m=1 (b) m=10

Figura 20 – Histograma da confusão para n=32 rodadas e m=1 e m=10 repetições.



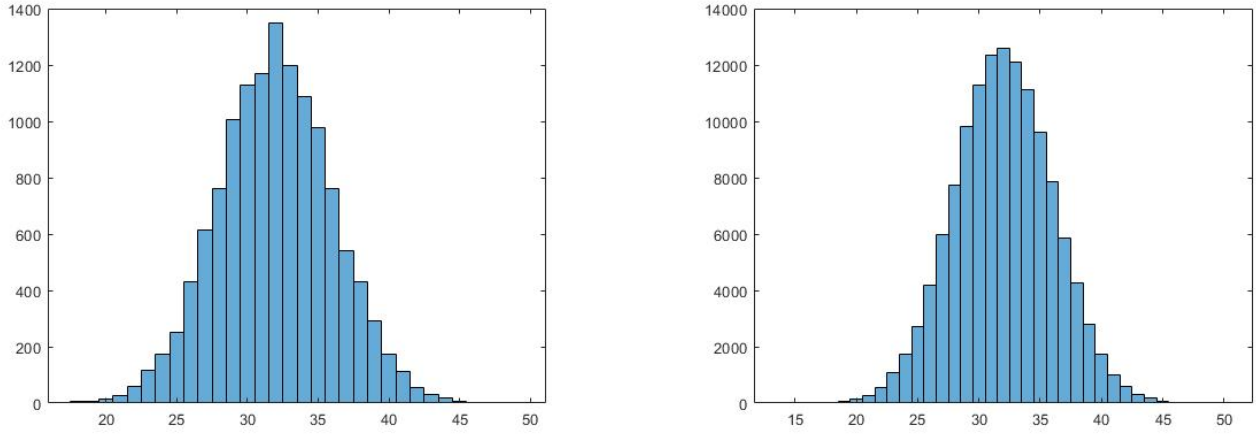
(a) m=100 (b) m=1000

Figura 21 – Histograma da confusão para n=32 rodadas e m=100 e m=1000 repetições.



(a) m=1 (b) m=10

Figura 22 – Histograma da confusão para n=50 rodadas e m=1 e m=10 repetições.



(a) m=100 (b) m=1000

Figura 23 – Histograma da confusão para n=50 rodadas e m=100 e m=1000 repetições.

4.3 EFEITO AVALANCHE

O efeito avalanche é uma característica importante para modelos criptográficos baseados em cifras de blocos, como o DES, AES e TEA. O efeito se baseia no fato de que, uma alteração em um bit do texto original ou em um bit da chave, deve produzir uma alteração em muitos bits do texto cifrado. A importância do efeito avalanche nesse tipo de cifra mostra que o algoritmo possui um elevado grau de aleatoriedade, e que portanto, se sujeito a uma criptoanálise, se torna mais complicada uma possível quebra da criptografia. O efeito avalanche, foi citado pela primeira vez por Horst Feistel [FEISTEL 1973], mas como podemos observar, as propriedade de confusão e difusão de Shannon [SHANNON 1948] analisadas neste trabalho, já continham boa parte dos conceitos utilizados.

A Equação 4.9 nos fornece uma forma de mensurar o efeito avalanche EA, temos que

$$EA = \frac{\text{número de bits alterados no texto cifrado}}{\text{número de bits no texto cifrado}}. \tag{4.9}$$

Uma boa cifra deve sempre satisfazer uma avalanche de 50%. Além disso, é desejado realizar essa análise para várias repetições de uma cifra e observar o efeito médio de avalanche.

Para o caso do TEA, como o texto cifrado possui 64 bits, se considerarmos $\#i$ a quantidade de bits modificados no texto cifrado quando o bit da posição i no texto original é trocado, então podemos calcular o efeito avalanche para cada $i = 1, \ldots, 64$ por

$$EA_i = \frac{\#i}{64}. \tag{4.10}$$

Na Tabela 6 exemplificamos os cálculos do efeito avalanche a partir da Equação 4.10, considerando os dados da Tabela 4 para o TEA com $n = 32$ rodadas.

Tabela 6 – Efeito Avalanche (EA) para cada mudança de um bit no texto original

bit i	$\#i$	EA_i	**bit** i	$\#i$	EA_i	**bit** i	$\#i$	EA_i	**bit** i	$\#i$	EA_i
1	32	50%	**17**	32	50%	**33**	31	48%	**49**	31	48%
2	34	53%	**18**	32	50%	**34**	26	41%	**50**	29	45%
3	28	44%	**19**	32	50%	**35**	33	52%	**51**	28	44%
4	27	42%	**20**	30	47%	**36**	30	47%	**52**	29	45%
5	28	44%	**21**	30	47%	**37**	32	50%	**53**	28	44%
6	34	53%	**22**	31	48%	**38**	32	50%	**54**	32	50%
7	34	53%	**23**	27	42%	**39**	33	52%	**55**	29	45%
8	35	55%	**24**	31	48%	**40**	25	39%	**56**	30	47%
9	29	45%	**25**	35	55%	**41**	28	44%	**57**	36	56%
10	36	56%	**26**	35	55%	**42**	34	53%	**58**	36	56%
11	37	58%	**27**	27	42%	**43**	35	55%	**59**	32	50%
12	34	53%	**28**	39	61%	**44**	29	45%	**60**	41	64%
13	33	52%	**29**	36	56%	**45**	30	47%	**61**	35	55%
14	29	45%	**30**	34	53%	**46**	34	53%	**62**	29	45%
15	34	53%	**31**	31	48%	**47**	37	58%	**63**	34	53%
16	30	47%	**32**	34	53%	**48**	33	52%	**64**	28	44%

fonte: Próprio Autor

Se considerarmos todos os valores de EA_i para cada $i = 1, \ldots, 64$, podemos calcular uma média para o efeito avalanche de um conjunto de dados do TEA, no qual cada bit i do texto original foi modificado, por

$$EA = \frac{\frac{\sum_{i=1}^{64} \#i}{64}}{64} = \frac{\sum_{i=1}^{64} \#i}{64^2}. \tag{4.11}$$

Considerando os dados da Tabela 6, podemos verificar que a média do efeito avalanche para aquele conjunto de dados é

$$EA = \frac{\sum_{i=1}^{64} \#i}{64^2} = \frac{2039}{64^2} \simeq 0,5, \tag{4.12}$$

que é o desejado. Porém neste exemplo foi realizada somente uma repetição do algoritmo. Para obtermos uma conclusão mais confiável, realizamos os cálculos a partir dos dados dos histogramas da Figura 15 (a) e (b) para $m = 100$ e 1000 repetições com $n = 32$ e o mesmo resultado foi encontrado, ou seja, $EA = 0,5$ ou 50%.

Agora vamos analisar o efeito avalanche em relação a troca dos bits da chave. Para exemplificar, vamos utilizar os dados da Tabela 5 que apresenta a quantidade de bits trocados $\#i$ no texto cifrado quando o bit i da chave é modificado, para $i = 1, \ldots, 128$. Para o cálculo do efeito avalanche EA_i iremos utilizar novamente a Equação 4.10 porém para $i = 1, \ldots, 128$.

Tabela 7 – Efeito Avalanche (EA) para cada mudança de um bit na chave (bit 1 à 64).

bit i	$\#i$	EA_i	**bit** i	$\#i$	EA_i	**bit** i	$\#i$	EA_i	**bit** i	$\#i$	EA_i
1	36	56%	**17**	34	53%	**33**	36	56%	**49**	30	47%
2	32	50%	**18**	30	47%	**34**	36	56%	**50**	31	48%
3	38	59%	**19**	25	39%	**35**	32	50%	**51**	36	56%
4	28	44%	**20**	34	53%	**36**	33	52%	**52**	35	55%
5	26	41%	**21**	32	50%	**37**	34	53%	**53**	31	48%
6	33	52%	**22**	39	61%	**38**	34	53%	**54**	38	59%
7	36	56%	**23**	27	42%	**39**	30	47%	**55**	33	52%
8	36	56%	**24**	37	58%	**40**	31	48%	**56**	33	52%
9	30	47%	**25**	23	36%	**41**	32	50%	**57**	32	50%
10	25	39%	**26**	23	36%	**42**	34	53%	**58**	31	48%
11	33	52%	**27**	34	53%	**43**	33	52%	**59**	32	50%
12	27	42%	**28**	31	48%	**44**	30	47%	**60**	38	59%
13	36	56%	**29**	34	53%	**45**	31	48%	**61**	32	50%
14	33	52%	**30**	32	50%	**46**	32	50%	**62**	41	64%
15	33	52%	**31**	35	55%	**47**	34	53%	**63**	31	48%
16	38	59%	**32**	29	45%	**48**	33	52%	**64**	29	45%

fonte: Próprio Autor

Da mesma forma, podemos calcular uma média para o efeito avalanche de um conjunto de dados do TEA, no qual cada bit i da chave foi modificado, por

$$EA = \frac{\frac{\sum_{i=1}^{128} \#i}{128}}{64} = \frac{\sum_{i=1}^{128} \#i}{128 \times 64}. \tag{4.13}$$

Tabela 8 – Efeito Avalanche (EA) para cada mudança de um bit na chave (bit 65 à 128).

bit i	$\#i$	EA_i	**bit** i	$\#i$	EA_i	**bit** i	$\#i$	EA_i	**bit** i	$\#i$	EA_i
65	30	47%	**81**	21	33%	**97**	28	44%	**113**	34	53%
66	30	47%	**82**	30	47%	**98**	32	50%	**114**	29	45%
67	34	53%	**83**	33	52%	**99**	31	48%	**115**	30	47%
68	26	41%	**84**	30	47%	**100**	29	45%	**116**	34	53%
69	31	48%	**85**	31	48%	**101**	22	34%	**117**	32	50%
70	33	52%	**86**	36	56%	**102**	33	52%	**118**	30	47%
71	37	58%	**87**	34	53%	**103**	24	37%	**119**	38	59%
72	37	58%	**88**	25	39%	**104**	25	39%	**120**	27	42%
73	34	53%	**89**	33	52%	**105**	33	52%	**121**	32	50%
74	32	50%	**90**	34	53%	**106**	34	53%	**122**	28	44%
75	39	61%	**91**	31	48%	**107**	39	61%	**123**	33	52%
76	28	44%	**92**	34	53%	**108**	38	59%	**124**	39	61%
77	29	45%	**93**	24	37%	**109**	31	48%	**125**	37	61%
78	37	58%	**94**	29	45%	**110**	31	48%	**126**	37	58%
79	28	44%	**95**	30	47%	**111**	35	55%	**127**	34	53%
80	34	53%	**96**	36	56%	**112**	31	48%	**128**	27	42%

fonte: Próprio Autor

Considerando os dados das Tabelas 7 e 8, a média do efeito avalanche para aquele conjunto de dados é

$$EA = \frac{\sum_{i=1}^{128} \#i}{128 \times 64} = \frac{4093}{128 \times 64} \simeq 0.5, \tag{4.14}$$

como esperado. Novamente para neste exemplo foi realizada somente uma repetição do algoritmo, e para obtermos uma conclusão mais confiável, realizamos os cálculos a partir dos dados dos histogramas da Figura 21 (a) e (b) para $m = 100$ e 1000 repetições com $n = 32$ e o mesmo resultado foi encontrado, ou seja, $EA = 0.5$ ou 50%.

5 CONCLUSÕES

Os modelos criptográficos representam uma parte importante dos dispositivos eletrônicos atuais, principalmente aqueles que trocam algum tipo de informação. Nesse sentido, o estudo apresentado neste trabalho propiciou uma análise mais profunda de dois modelos criptográficos baseados em redes de Feistel, o DES e principalmente do TEA. Foram realizados os estudos sobre a criptografia e a decriptografia destes modelos. Além disso foram realizadas implementações computacionais o que nos permitiu realizar algumas análises destes tipos de criptografia, procurando entender para qual tipo de contexto cada uma delas seria mais efetiva. O DES, mais robusto, é mais recomendado para equipamentos que permitem um processamento mais pesado. Já o TEA, tem um processo muito mais simplificado, e por conta da sua rapidez pode ser utilizado em dispositivos menos robustos, que possuem capacidade de processamento mais reduzida. Além disso, a sua simplicidade e rapidez dificulta a possibilidade de quebra de segurança, pois o tempo hábil para que isso aconteça é mínimo. Assim como a maioria dos modelos criptográficos utilizados nos dias de hoje, o TEA possui as propriedades de difusão e confusão propostas por Shannon, porém diferentemente da maioria dos algoritmos, sem precisar de tabelas como P-boxes e S-boxes e ainda alcança um difusão e confusão completa em apenas 6 rodadas. Dessa forma, foi apresentado também neste trabalho uma análise sobre a difusão e confusão de Shannon, mostrando que de fato este algoritmo apresenta tais propriedades a partir de um pequeno número de rodadas. As análises foram realizadas variando o algoritmo em 5, 32 e 50 rodadas e repetindo o algoritmo 1, 10, 100 e 1000 vezes. Considerando 100 e 1000 repetições foi possível observar através de histogramas que a distribuição de frequência dos bits trocados tanto no texto original (difusão) quanto na chave (confusão) seguiam uma distribuição gaussiana com média 32. Além disso, foi proposta uma análise utilizando a teoria do efeito avalanche, mostrando que na média o TEA apresenta um efeito avalanche de 50%. A confirmação desta propriedade, conjunta com confusão e difusão, contribuem para mostrar que o TEA é de fato confiável, uma vez que ela mostra que o modelo permite um grau de aleatoriedade alto, aliado à velocidade com que o processo é completado. Estes fatores comprovam que ele pode sim ser muito bem empregado em sistemas que não possuam um processamento tão avançado, seja por serem de uma tecnologia arcaica ou por escassez de recursos.

Como continuidade deste trabalho podem ser realizadas implementações computacionais em *hardware* de modo a analisar melhor o desempenho deste algoritmo, principalmente no cenário real de aplicações como em RFID, IoT e em sistemas móveis. Além disso, variações do TEA podem ser estudadas, como o XTEA - *eXtended Tiny Encrypt Algorithm*. Dando continuidade as análises de desempenho pode-se também avaliar se o TEA apresenta o critério estrito avalanche - SAC (*strict avalanche criterion*), que é uma formalização do efeito avalanche criado para torna-lo mais prático. Esses conceitos foram introduzidos por Webster e Tavares em 1985 [WEBSTER A.; TAVARES 1970]. O SAC é satisfeito se, sempre que um único bit de entrada é modificado, cada um dos bits da saída muda com uma probabilidade de 50%. O critério não diz que exatamente metade dos bits mudam, mas sim que cada bit tem probabilidade de 50% de alteração. Análises para melhorar a seguraça do TEA também podem ser realizadas, como por exemplo, considerar os modos de operação [Preneel 2011].

REFERÊNCIAS

ABDELHALIM M.; EL-MAHALLAWY, M. A. M. Design and implementation of an encryption algorithm for use in rfid system. **International Journal of RFID Security and Cryptography**, v. 2, p. 51–57, 06 2013.

ANDEM, V. A cryptanalysis of the tiny encryption algorithm /. 01 2003.

APOSTOL, T. M. ***Introduction to analytic number theory***. 1. ed. [S.l.]: Springer, 1976.

BIRYUKOV, A. Feistel cipher. In: ____. **Encyclopedia of Cryptography and Security**. Boston, MA: Springer US, 2011. p. 455–455. ISBN 978-1-4419-5906-5. Disponível em: <https://doi.org/10.1007/978-1-4419-5906-5_577>.

BUDIYANTO D; PUTRO, P. A. W. Comparison of implementation tiny encryption algorithm (tea) and advanced encryption standard (aes) algorithm on android based open source cryptomator library. In: **2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)**. [S.l.: s.n.], 2018. p. 136–139.

CARPENTIERI, B. v. 2018, p. 1–9, 05 2018.

DAMAJ, I.; HAMADE, S.; DIAB, H. Efficient tiny hardware cipher under verilog. In: **2008 High Performance Computing Simulation Conference**. [S.l.: s.n.], 2008.

DAOR J.; DAEMON, J. R. V. Aes proposal: rijndael. 10 1999.

DELFS H.; KNEBL, H. ***Introduction to Cryptography: Principles and Applications***. 2. ed. [S.l.]: Springer, 2007.

DIKEVAR A. P.; ROUT, S. R. Tiny encryption algorithm on various platforms. **IJEDR**, v. 3, 2015.

FEISTEL, H. *Cryptography and Computer Privacy*. **Scientific American**, v. 228, 1973.

FELDHOFER, M.; WOLKERSTORFER, J. Hardware implementation of symmetric algorithms for rfid security. In: ____. **RFID Security: Techniques, Protocols and System-on-Chip Design**. Boston, MA: Springer US, 2008. p. 373–415. Disponível em: <https://doi.org/10.1007/978-0-387-76481-8_15>.

FELIX, N. **DES(str,KEY,Mode)**. [S.l.], 2020. <https://www.mathworks.com/matlabcentral/fileexchange/53768-des-str-key-mode?s_tid=FX_rc1_behav>. Acesso em: 8 de agosoto de 2020.

GILBERT, S. *RFID Security : Tiny Encryption Algorithm And Authentication Protocols*. **Theses and dissertations**, v. 1093, 2009.

HUNN S.A.Y.; NAZIRI, S. I. N. The development of tiny encryption algorithm (tea) crypto-core for mobile systems. In: . [S.l.: s.n.], 2012.

HUSSAIN, M. A.; BADAR, R. Fpga based implementation scenarios of tea block cipher. In: **2015 13th International Conference on Frontiers of Information Technology (FIT)**. [S.l.: s.n.], 2015. p. 283–286.

ISRASENA, P. Design and implementation of low power hardware encryption for low cost secure rfid using tea. In: **2005 5th International Conference on Information Communications Signal Processing**. [S.l.: s.n.], 2005. p. 1402–1406.

ISRASENA, P. Securing ubiquitous and low-cost rfid using tiny encryption algorithm. In: . [S.l.: s.n.], 2006. p. 4 pp.

JONES H. S.; LIDDELL, H. G. S. R. ***A Greek-English Lexicon: With a Revised Supplement***. [S.l.]: Clarendon Press, 1996.

JUNOD P.; CANTEAUT, A. ***Advanced Linear Cryptanalysis of Block and Stream Ciphers***. 1. ed. [S.l.]: IOS Press, 2011.

MA, E. Y.-T.; OBIMBO, C. An evolutionary computation attack on one-round tea. **Procedia Computer Science**, v. 6, p. 171 – 176, 2011. ISSN 1877-0509. Complex adaptive sysytems. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050911004984>.

MATHIVANAN, K. "overview of modern cryptography". **International Journal of Computer Science and Information Technologies**, Vol. 6 (1), p. 2015, 350–353, 09 2015.

MENEZES A. J.;VAN OORSCHOT, P. C. S. A. ***Handbook of Applied Cryptography***. 1. ed. [S.l.]: CRC Press, 1996.

PAAR C.; PELZL, J. ***Understanding Cryptography : a Textbook for Students and Practitioners***. 1. ed. [S.l.]: Springer, 2009.

PRENEEL, B. Modes of operation of a block cipher. In: ____. **Encyclopedia of Cryptography and Security**. Boston, MA: Springer US, 2011. p. 789–794. ISBN 978-1-4419-5906-5. Disponível em: <https://doi.org/10.1007/978-1-4419-5906-5_599>.

SAURABH S.; SHARMA, P. M. S. Y. P. J. Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions. **Journal of Ambient Intelligence and Humanized Computing**, p. 1–18, 04 2017.

SCHNEIER, B. ***Applied Cryptography***. 20. ed. [S.l.]: Wiley, 2017.

SHANNON, C. E. *A Mathematical Theory of Communication*. ***The Bell System Technical Journal***, v. 27, p. 379–423, 623–656, 1948.

SHANNON, C. E. *Communication Theory of Secrecy Systems*. ***The Bell System Technical Journal***, v. 28(4), p. 656–715, 1949.

STALLINGS, W. ***Cryptography and Network Security: Principles and Practice***. 1. ed. [S.l.]: Prentice Hall, 1999.

TRAPPE W.; WASHINGTON, L. C. ***Introduction to Cryptography with Coding Theory***. 2. ed. [S.l.]: Pearson Prentice Hall, 2006.

TUCHMAN, W. A brief history of the data encryption standard. In: ____. **Internet Besieged: Countering Cyberspace Scofflaws**. USA: ACM Press/Addison-Wesley Publishing Co., 1997. p. 275–280. ISBN 0201308207.

WEBSTER A.; TAVARES, S. On the design of s-boxes. In: . [S.l.: s.n.], 1970.

WHEELER DAVID J.; NEEDHAM, R. M. *TEA, a tiny encryption algorithm*. ***Lecture Notes in Computer Science***, v. 1008, p. 363–366, 1994.