UNIVERSIDADE DO VALE DO RIO DOS SINOS
CIÊNCIAS EXATAS E TECNOLÓGICAS,
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
APLICADA - PIPCA

Criptografia com Caos

por

DANIEL FORMOLO

Dissertação submetida à avaliação
como requisito parcial para a obtenção do grau de
Mestre em Computação Aplicada

Prof Dr. Luiz Paulo Luna de Oliveira
Orientador

São Leopoldo, abril de 2009.

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Formolo, Daniel

Criptografia com Caos / por Daniel Formolo. — São Leopoldo: Ciências Exatas e Tecnológicas da UNISINOS, 2009.

116 f.: il.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos. Ciências Exatas e Tecnológicas, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada - PIPCA, São Leopoldo, BR–RS, 2009. Orientador: de Oliveira, Luiz Paulo Luna.

I. de Oliveira, Luiz Paulo Luna. II. Título.

O hóspede sabe que nada é seu: um quarto, uma janela, a rua que se perde onde o chão vira céu.

Ele entendeu que tudo está de passagem e não adianta arame, areia, cimento, casa de madeira, de palha, de vento.

Não lhe peçam assinatura de proprietário no papel. O hóspede tem outra escritura, a dos seus versos.

O universo é o seu hotel.

(*Mario Quintana*)

TITLE: “CRYPTOGRAPHY WITH CHAOS”

Abstract

The technological advance and value of information in the preset days lead to a constant search for more efficient, secure and adaptable ciphers. While de security is based on the complicated behavior of the algorithms, its efficiency is based on the simplicity of the mathematical operations involved. The chaotic systems are characterized by the property of showing complex behavior even being defined by simple algebraic mathematical expressions. Therefore, the proposal of ciphers based on chaotic dynamics has became an active research field. This work deals with the efficiency and security of one of the most complete chaotic cipher proposed so far. Based on this study, improvements are proposed for both efficiency and security. The final result is an optimized version, which is denominated *chaotic cipher* (CC) towards this work. This cipher does not suffer from the weaknesses of the chaotic ciphers proposed so far. Moreover, experimental comparisons shows that the CC performance is comparable with those of some of the most efficient ciphers of commercial usage (AES, RC4 and Sosemanuk). These results suggest that chaotic cryptography has a promising future as a method for commercial usage.

Keywords: Cryptography, Chaos, Algorithm.

Resumo

O avanço tecnológico e o valor que as informações representam no mundo moderno conduzem a uma constante busca por cifradores mais velozes, seguros e adaptáveis à diversos ambientes. Enquanto a segurança se baseia no comportamento complicado dos algoritmos, a eficiência se baseia na simplicidade das operações matemáticas envolvidas. Os sistemas caóticos têm a propriedade de exibir comportamento complicado mesmo sendo definido de modo determinístico por funções algebricamente simples. Portanto, a proposta de algoritmos de cifração baseados em sistemas caóticos é um tema de ativa pesquisa. Este trabalho estuda a eficiência e a segurança de um dos mais completos cifradores caóticos. Com base nesse estudo, melhorias são propostas tanto quanto à sua eficiência quanto à sua segurança. Com isto, chega-se a uma versão otimizada, a qual é referida ao longo deste trabalho como *cifrador caótico* (CC). O cifrador resultante não possui as fraquezas dos algoritmos caóticos propostos até então. Além disso, comparações experimentais mostram que o CC tem performance comparável com alguns dos mais eficientes algoritmos de cifragem de uso comercial (AES, RC4 e Sosemanuk). Tais resultados mostram que a criptografia caótica tem futuro promissor como método ce cifragem de uso comercial.

Palavras-chave: Criptografia, Caos, Algoritmo.

Sumário

Lista de Abreviaturas

AES	Advanced Encryption Standard
BIC	Bits Independence Criterion
CBC	Cipher Block Chaining
CCM	Counter CBC Mode
CCOS	Cifrador Caótico De Oliveira e Sobottka
CFB	Cipher FeedBack
DES	Data Encryption Standard
EAX	Encryptation Autentication mode
eEstream	Projeto que visa identificar um cifrado de fluxo simétrico para ser adotado como padrão de criptografia de fluxo simétrica
ECB	Eletronic Code Book
ECRYPT	European Network of Excellence for Cryptology
FIFO	First in fisrt out
IV	Initialization Vector
KB	Kilo Byte
GCM	Galois Counter Mode
HSM	Hardware Security Module.
IAPM	Integrity Aware Parallelizable Mode
LFSR	Linear Feedback Shift Register

LRU	Least Recently Used
MB	Mega Byte
MEF	Máquina de Estados Finitos
MHz	Mega Hertz
ms	Milisegundos
NIST	National Institut of Standards and Technology
OCB	Offset Code Book
OFB	Output Feedback
SAC	Strict Avalanche Criterion
SB	Sistema de Baptista
UML	Unified Modeling Language.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Criptologia é a ciência que estuda técnicas para restringir e obter acesso a informações sigilosas. Modernamente, ela é dividida em criptografia e criptoanálise. A primeira, estuda os métodos de cifração de dados. A segunda, estuda as maneiras de quebrar o sigilo criado pela criptografia, através de seus cifradores e protocolos de transporte de dados [10].

A criptografia é tão antiga, que estava presente até mesmo em hieroglífos egípcios [11]. Os romanos utilizavam códigos secretos para comunicar planos militares com a chamada Cifra de Cesar [11]. Com o passar do tempo e o advento dos circuitos micro-processados, os métodos de criptografia se modificaram. Hoje eles estão fortemente fundamentados em algoritmos matemáticos. Da mesma forma, a criptoanálise também acompanhou esta evolução [3].

Atualmente, com o aumento do poder de processamento de informações, é muito fácil quebrar textos cifrados com a maioria dos algoritmos clássicos de criptografia. Nos sistemas modernos, a construção de algoritmos criptográficos é dada por uma composição de funções matemáticas, denominadas *primitivas*. Para serem utilizadas na cifração/decifração de dados, tais primitivas devem possuir características convenientes como velocidade (eficiência), facilidade na troca de chaves, e altos índices de difusão e confusão [12, 13].

Existem critérios definidos para se avaliar e classificar as primitivas. Conforme estes critérios, pode-se dividir os algoritmos criptográficos em cifradores simétricos e assimétricos [10, 12]. Os algoritmos assimétricos usam chaves diferentes para cifrar

e decifrar dados. Por outro lado, os algoritmos simétricos usam a mesma chave para cifração e decifração. Os cifradores simétricos são divididos ainda em cifradores de fluxo e cifradores de bloco. Os cifradores de fluxo cifram as unidades de texto plano individualmente, a cada iteração, enquanto os cifradores de bloco cifram um bloco de unidades de texto plano a cada iteração [1]. Embora cifradores simétricos sejam mais rápidos que assimétricos, o uso da mesma chave dificulta a logística de troca de chaves [14]. Por conseqüência disso, a maioria dos sistemas criptográficos utilizam um modelo híbrido, isto é, a troca de chaves dos cifradores simétricos é realizada com um algoritmo assimétrico, e a troca de dados é feita com um algoritmo simétrico. O uso de cifradores simétricos para troca de dados se justifica pela sua velocidade e uso de poucos recursos de *hardware*. Essas características vão ao encontro das necessidades atuais da criptografia [15, 16]. Neste trabalho são estudados os cifradores simétricos e assimétricos, focando principalmente os cifradores simétricos de fluxo.

A criptografia esta presente nas mais diversas áreas, sua importância é muito maior do que em outros tempos. A criptografia é usada na cifração de mensagens, cifração de documentos, acesso a sistemas, envio de sinais de longa distância como nas transmissões de TV-Digital, de modo geral nas transações financeiras, como em bolsas de valores, operações entre bancos, cartões de crédito, caixas eletrônicos, entre outros [10]. A crescente utilização de equipamentos portáteis e novas tecnologias de informação estimulam a comunicação entre pessoas, entre empresas, e entre pessoas e empresas. Aliado a isso, é cada vez mais comum o uso de meios não seguros para troca de informação como *wireless*, *bluetooth* e comunicação via rádio. A conjunção desses fatores aumenta a demanda na transmissão de dados por mais segurança e eficiência, isto é, cifração mais veloz e com menor uso de memória.

Visando agrupar segurança com pouco uso de recursos computacionais, há vinte anos, pesquisadores vêm estudando as características dos sistemas caóticos e sua aplicabilidade na criptografia. Existem dois modos principais de utilização de sistemas caóticos: o Modo de Sincronização e o Modo de Busca. O primeiro foi inicialmente proposto por Pecora e Carroll. Este modo se baseia em sistemas caóticos com sincronização analógica [17]. Além deste trabalho, outros autores propuseram sistemas criptográficos baseados em sistemas caóticos síncronos, como em [18, 19]. Outro exemplo deste tipo de sistema foi o apresentado por Álvarez et al. em [20]. Nesse trabalho, Álvarez apresenta um cifrador baseado em um oscilador caótico não

linear, neste caso, um atrator de Lorentz-3D, o qual gera sinais pseudo-aleatórios que são combinados com o texto plano para gerar uma seqüencia de caracteres cifrados. O receptor regenera o sinal pseudo-aleatório de forma invertida. Combinando este sinal com o texto cifrado ele consegue recuperar o texto plano. Essa forma de cifrar dados tem alguns problemas, principalmente referentes a ruídos de comunicação e segurança, conforme apresentado em [21, 22, 9, 23].

O método de criptografia caótica classificado como Modo de Busca surgiu recentemente. Ele utiliza mapas caóticos sem a necessidade de sincronização de sistemas entre emissor e receptor. O Modo de Busca parte do princípio que, sendo $P = (P_j)_{1 \leq j \leq L}$ o texto plano com tamanho L escrito em um alfabeto finito $\mathcal{A}$ e, sendo $f_p : I \rightarrow I$, a família de mapas caóticos escolhida, com intervalo I e parâmetro p. O intervalo I é dividido em N subintervalos semi-abertos I_k, $k = 1, 2, \ldots, N$ denominados sítios. Cada sítio é associado de forma uniforme à um símbolo de $\mathcal{A}$. O processo de cifração consiste em iterar f_p consecutivamente, partindo da condição inicial (chave) $x_0 \in I$, computando o número de iterações necessárias para a correspondente órbita cair dentro de cada sítio I_k associado com a sucessão de símbolos que compõem P. A versão cifrada de P é a seqüencia $C = (C_j)_{1 \leq j \leq L}$ dos valores C_j, correspondentes a quantidade de iterações de f_p, necessárias para a órbita $(x_j) = (f_p^j(x_0))$ atingir os sítios associados a cada um dos símbolos consecutivos de P. Mais precisamente, C é a seqüencia de números naturais $C = (C_j)_{1 \leq j \leq L}$ tal que $f_p^{C_j}(x_{C_{j-1}})$ é a primeira ocorrência, em um intervalo I_k, associado à P_j. A decifração de C é feita usando a mesma chave, de maneira óbvia: lendo cada C_j como um símbolo P_j associado com o subintervalo I_k para que $f_p^{\sum_{i=1}^{j} C_i}(x_0)$.

A viabilidade desses sistemas se baseia na capacidade que os sistemas caóticos têm de exibir comportamentos complexos, mesmo com formulação matemática simples. Isto indica uma boa aplicação em sistemas criptográficos devido às características de confusão e difusão que o caos provê e pela simplicidade (baixo uso de processamento e memória) com que este método utiliza as propriedades do caos.

O mais estudado destes algoritmos foi proposto por Baptista [6]. Em seu trabalho, ele propôs um sistema de cifração baseado em um mapa caótico unidimensional, a saber, o mapa logístico, dado por $f_\mu(x) = \mu x(1 - x)$. Seu sistema tem um subdomínio apropriado de f_μ, com $3 \leq \mu \leq 4$, e utiliza as iterações do mapa logístico para cifrar os dados. Tal sistema será detalhado na Seção 3.2.

Sobre a proposta de Baptista, surgiram críticas quanto à segurança do método contra ataques com texto plano conhecido. Um dos ataque desses tipo foi proposto por Jakimoski e Kocarev em [24]. Entretanto, este método de ataque foi criticado por Li et al. que não o consideram viável [9]. Outros autores também apontaram críticas ao modelo de Baptista. Em seu trabalho, Álvarez et al. apontam deficiências de segurança, sugerindo outras três formas de ataque [8]. Uma delas é parecida com o método de ataque proposto por Jakimoski e Kocarev; a segunda utiliza análise de entropia e estatística; e a terceira, possibilita a dedução do valor do parâmetro μ da função logística, caso este faça parte da chave. Por outro lado, Wong [25] propõem um método para resolver o problema de entropia, utilizando o mesmo mapa de Baptista, porém cifrando os dados em blocos de texto plano, ao invés de símbolo a símbolo, conforme definido originalmente.

Além das críticas quanto à segurança, o método de Baptista apresenta algumas deficiências de outros tipos. Uma delas é a não uniformidade da freqüência com que os sítios associados aos símbolos do alfabeto utilizado são visitados pela órbita escolhida. Isso pode prejudicar muito a taxa de cifração/decifração, caso existam, na mensagem, símbolos associados à sítios de escassa visitação. Outro problema é quanto à escolha do valor do parâmetro μ. De fato, dentro da restrição $3 < \mu \leq 4$, há muitos valores para os quais f_μ não é caótico. Estes valores de μ não podem ser utilizados no mapa f_μ para cifração de dados. Além disso, o sistema de Baptista é dependente do *hardware*, isto é, temos que utilizar máquinas com a mesma precisão e mesma metodologia determinística de arredondamento numérico para cifrar os dados, caso contrário, a decifração dos dados fica comprometida. Isto restringe e muito seu uso generalizado.

Recentemente, foi proposto um método de cifração que utiliza as mesmas idéias básicas de Baptista, mas sem os problemas acima citados [26] [7]. No lugar da aplicação logística, foram usados mapas do tipo $\mathbb{H}_p(x) = r_p^{-1} \circ \mathbb{G} \circ r_p(x)$ definidos no intervalo $[0,\ 10^p)$ para um parâmetro racional positivo p, onde $\mathbb{G}(x) = 10x (\mathrm{mod}\ 10)$ e $r_p(x) = \sqrt[p]{x}$. É possível mostrar que este tipo de mapa unidimensional é caótico para todo valor do parâmetro p, ao contrário do que acontece no sistema de Baptista. A divisão do seu domínio é feita de tal forma que os sítios são visitados pelas órbitas de $\mathbb{H}_p$, com a mesma probabilidade. Além disso, é independente de máquina.

Este trabalho apresenta, em detalhes, ambos os métodos: Sistema de Baptista

e o sistema proposto de De Oliveira e Sobottka [26, 7]. Baseado no algoritmo de De Oliveira e Sobottka, o trabalho propõe um novo algoritmo criptográfico com melhor desempenho que o anterior e que, em momento algum, fere as prerrogativas do algoritmo antigo, definidas em [26, 7]. Este novo cifrador é aqui referido como *Cifrador Caótico* (CC). Os diferenciais do CC em relação ao antigo algoritmo são:

- Aumento da velocidade de cifração/decifração com a disposição das órbitas caóticas geradas em uma Tabela de Endereçamentos.
- Aumento na segurança, tornando o cifrador estatístico.
- Diminuição no tamanho do texto cifrado através de um novo método de armazenar os símbolos cifrados.

Além disso, o trabalho avaliar a eficiência do CC em termos de velocidade, segurança e características de uso. Para isso, foi criado um *software* com implementações do CC, além de outros três algoritmos da criptografia moderna, usados na comparação com o CC em termos de velocidade, segurança e características de uso. Os outros três algoritmos são: o cifrador de bloco AES (*Advanced Encryption Standard*) [27], que é o cifrador simétrico mais usado comercialmente; o Cifrador RC4, que há muito tempo é um dos cifradores de fluxo mais utilizados comercialmente; e o cifrador de fluxo Sosemanuk [28], cogitado como um dos possíveis vencedores do projeto *eEstream*, organizado por um convênio de empresas, instituições e universidades européias chamado de *ECRYPT*. O *eEstream* visa identificar cifradores de fluxo promissores, e que possam ser aplicados comercialmente [28]. Com isso, este trabalho se constitui num estudo inédito para a viabilidade do uso, em escala comercial, do CC e de cifradores similares a ele. Os resultados obtidos nessas comparações sugerem que o CC é eficiente e seguro, podendo se candidatar ao uso comercial.

Essa dissertação está dividida em seis capítulos e, ao final dela, um glossário esclarece os termos da área de criptografia que foram utilizados no trabalho. No Capítulo 2, é apresentada uma breve revisão dos conceitos básicos da criptografia moderna, incluindo os cifradores AES, RC4 e Sosemanuk. No Capítulo 3 é apresenta a dinâmica caótica dos mapas unidimensionais a qual, introduz a descrição do sistema de Baptista e do sistema proposto por De Oliveira e Sobottka. O Capítulo 4 refere-se ao

CC. No Capítulo 5 são apresentados os resultados da avaliação do CC. As conclusões sobre esta análise são apresentadas no Capítulo 6.

Capítulo 2

Criptologia Moderna

Conforme citado no capítulo anterior, a criptografia estuda os métodos de cifração de dados e a criptoanálise estuda as maneiras de quebrar o sigilo criado pela criptografia. A união dessas duas áreas forma a criptologia.

A criptografia utiliza cifradores para proteger as informações contidas nos dados. Estes cifradores são classificados segundo suas primitivas e as funcionalidades providas por estas. As primitivas são todo tipo de função de criptografia de baixo nível que compõem um cifrador. Elas podem ser utilizadas de forma isolada, como um cifrador simples, ou combinadas, formando um cifrador, em geral, mais robusto em termos de segurança. As primitivas têm características diferentes entre si. O uso delas e a maneira com que são combinadas, também geram cifradores com diferentes características. Segundo Menezes et. al. [12], os principais critérios para a avaliação das primitivas quanto ao uso pretendido são: segurança, funcionalidade e desempenho.

Conforme esses critérios, os cifradores são classificados como simétricos ou assimétricos. Os cifradores assimétricos utilizam chaves diferentes para cifrar e decifrar dados. Já os cifradores simétricos utilizam a mesma chave para cifrar e decifrar dados. As discussões neste trabalho serão focadas nos cifradores simétricos sem, no entanto, deixar de falar brevemente sobre os cifradores assimétricos, que são um ramo importantíssimo da criptografia e que complementam a criptografia simétrica nos sistemas de segurança usados hoje em dia.

2.1 Cifradores Assimétricos

Cifradores assimétricos ou de chave pública, têm como principal característica o uso de duas chaves distintas: uma para cifrar e outra para decifrar o texto. A chave de cifração costuma ser de domínio público, enquanto que a chave de decifração deve ser mantida sob sigilo absoluto.

Para isso, são utilizadas funções matemáticas do tipo

$$C = F_c(P, K_c) \quad \text{(cifração)} \qquad \text{e} \qquad D = F_d(C, K_d) \quad \text{(decifração)}, \tag{2.1}$$

onde

- P é o texto plano (original)
- C é o texto cifrado;
- D é o texto plano decifrado, tal que $D = P$;
- F_c é o algoritmo de cifração;
- F_d é o algoritmo de decifração;
- K_c é a chave de cifração;
- K_d é a chave de decifração.

Usualmente, tem-se $F_d = F_c = F$, isto é, a ação destas funções sobre os textos P e C só dependem das chaves K_c e K_d, respectivamente [1].

Em tais funções, o valor de saída é muito fácil de se encontrar a partir do valor de entrada. Porém, é exigido que o contrario seja extremamente difícil e computacionalmente inviável, sem se conhecer um determinado número (chave) que torne o cálculo do valor de entrada simples. Em outras palavras, enquanto a cifração é de fácil implementação, a decifração é efetivamente inviável sem o conhecimento da chave privada.

Nos cifradores assimétricos, o protocolo de cifração segue os seguintes passos:

- Cada usuário do sistema gera um par de chaves, uma de cifração e outra de decifração.

- Os usuários publicam suas chaves de cifração em um repositório público.
- Se o usuário B deseja enviar uma mensagem para o usuário A, ele cifra a mensagem com a chave pública de A.
- Por sua vez, ao receber a mensagem codificada, A decifra a mensagem com a sua chave privada, que somente ele conhece.

Esta assimetria de chaves traz profundas conseqüências nas áreas de confidencialidade, distribuição de chaves e autenticação. No protocolo acima, não é necessário um canal seguro para troca de chaves. Isso mostra claramente a facilidade na distribuição de chaves e a garantia de confidencialidade das informações. No caso da autenticação o protocolo deve ser modificado, incorporando um segundo par de chaves:

- Cada usuário do sistema gera dois pares de chaves, duas de cifração e outras duas de decifração.
- Os usuários publicam um chave de cifração, que será usada para codificar a mensagem e uma chave de decifração, que será usada como assinatura da mensagem, garantindo a procedência da mensagem. Ambas são disponibilizadas em um repositório público.
- Se B deseja enviar uma mensagem para o usuário A, ele assina a mensagem cifrando-a com sua chave de cifração, depois cifra a mensagem assinada com a chave pública de A.
- Por sua vez, ao receber a mensagem codificada, A decifra a mensagem com a sua chave privada e depois autentica a origem da mensagem com a chave de decifração publicada por B.

Alguns cifradores, como o RSA, permitem que qualquer uma das chaves seja escolhida como a chave de cifração, enquanto a outra automaticamente torna-se a chave de decifração [1].

2.2 Cifradores Simétricos

Os cifradores simétricos são divididos em dois conjuntos: cifradores de bloco e cifradores de fluxo. Os cifradores de bloco operam geralmente cifrando blocos de 64 bits de informação. Um mesmo bloco de texto plano gera o mesmo bloco de texto cifrado [1, 10]. Isto significa que, usando a mesma chave para cifrar blocos de texto plano iguais, o resultado serão blocos de texto cifrados idênticos, independente da quantidade e conteúdo dos blocos cifrados entre estes.

Os cifradores de fluxo operam cifrando bit a bit (ou byte a byte) as informações e usam a mesma chave para cifração e decifração . Assim, um bit de entrada, gera um bit cifrado diferente na saída do sistema [1]. Existem vários modos de cifradores de bloco e de fluxo operar. Estes modos tem por objetivo realizar um pré-tratamento dos dados de entrada do algoritmo e adicionar características como dependência entre blocos cifrados ou proteção contra ruídos na transmissão de dados do canal ou ainda transformar um cifrador de bloco em um cifrador de fluxo. Mas a função principal dos modos de operação é criar uma dependência entre a seqüência de blocos de texto cifrado, isto torna os cifradores mais seguros.

Existem quatro modos básicos de operação, dos quais foram derivados muitos outros ao longo do tempo, como os modos IAPM, CCM, EAX, GCM e OCB. A seguir, os quatro modos básicos são brevemente descritos, a fim de se mostrar a influência que cada Modo de Operação tem no resultado final da cifração, quando combinado com as primitivas de criptografia.

- Eletronic CodeBook (ECB): Esse é o modo mais simples. Cada parte do texto plano entra diretamente no algoritmo de cifração. Igualmente, a mesma chave, sem qualquer alteração, é utilizada na cifração de cada parte do texto plano. Isso significa que que para o texto plano $P = (P_1, P_2, P_i..., P_{i+n})$ se $P_i = P_{i+n}$, as partes de texto cifrado C_i e C_{i+n} sempre serão iguais. Este modo é ideal para cifrar quantidades pequenas de dados. A Figura 2.1 mostra seu uso na cifração. A decifração se dá de forma similar.

- Cipher Block Chaining (CBC): O CBC trabalha de forma similar ao ECB, usando a mesma chave para cifrar os blocos de dados. Para evitar o problema de gerar blocos cifrados idênticos para os mesmos blocos de textos planos, o CBC realiza

uma operação de XOR entre cada bloco de texto plano e o bloco anteriormente cifrado. Na primeira cifração, com não existe um bloco cifrado para ser combinado com P_1, se aplica a operação de XOR com um vetor de inicialização (IV). Este IV contém uma seqüência de dados com o mesmo tamanho do bloco de texto plano P_1 e é conhecido apenas pelo emissor e pelo receptor. A Figura 2.1 mostra o processo de cifração.

Se algum bit de um Bloco B for alterado o CBC consegue se recuperar após B+2 blocos. Porém ele não consegue se recuperar quando mais de um bit for alterado, perdido ou incluído. Este tipo de situação ocorre geralmente quando existe ruído no canal.

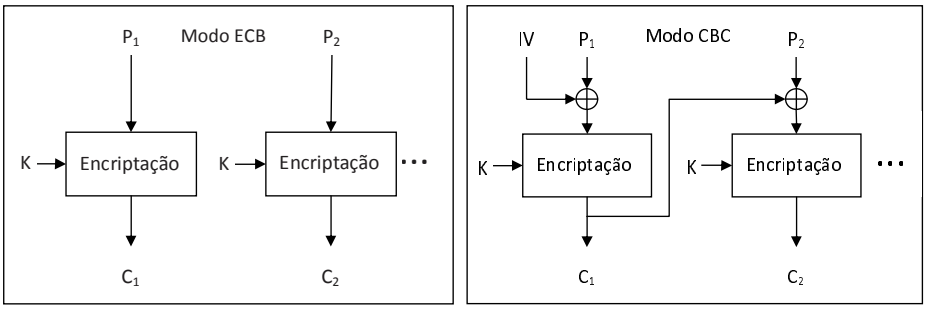

FIGURA 2.1 – cifração dos modos ECB e CBC. No modo CBC, o vetor IV deve conter valores conhecido entre emissor e receptor [1].

- Cipher FeedBack (CFB): Nesse modo a operação o vetor de inicialização IV, é conhecido apenas pelo emissor e pelo receptor, e é cifrado com a chave K. Depois uma função simples seleciona apenas a quantidade de bits que define um bloco ou unidade de texto plano. Chamamos todo este conjunto de operações de função de cifração.

  Sendo s a quantidade de bits de cada bloco do texto plano $P = (P_1, P_2, ..., P_n)$, uma operação de XOR é realizada entre o bloco P_1 e o resultado da função de cifração. A cifração dos próximos blocos segue a mesma operação de XOR entre os blocos do texto plano $P_2, P_3, ..., P_m$ e o vetor de bits, porém a cada operação o vetor de bits agrega s bits a sua direita e descarta os s bits mais a esquerda (bits mais significativos).

A decifração ocorre pela mesma forma, exceto que a operação de XOR se dá com cada bloco de texto cifrado para se obter os blocos de texto plano. Este Modo de Operação transforma cifradores de bloco em cifradores de fluxo. Além disso, ele provê auto-sincronismo do sistema de cifração. A Figura 2.2 demonstra o processo de cifração do modo CFB.

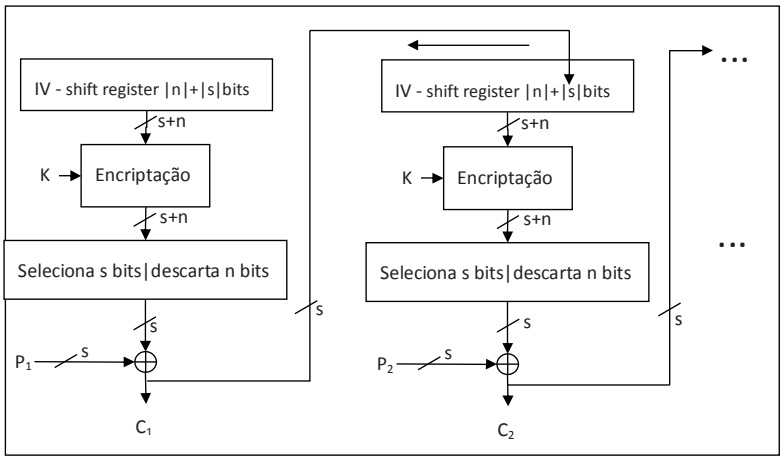

FIGURA 2.2 – cifração do Modo CFB [1].

- Output Feedback (OFB): Este modo é similar ao CFB, a única mudança esta na operação de XOR de cada bloco, que é alimentada pela função de cifração. Isto traz como benefícios a garantia de que se ocorrer um erro em um bit do bloco n os blocos seguintes não serão afetados, isto é, este modo é resistente a ruídos no canal de transmissão. A desvantagem deste modo são os ataques que modificam o texto cifrado. A Figura 2.3 demonstra o processo de cifração do modo OFB.

Em geral, não se adotam cuidados extremos com a segurança no Modo de Operação, pois ela esta fundada na função do algoritmo de cifração. Quanto mais forte a segurança gerada pelo algoritmo de cifração, menor o compromisso com segurança do Modo de Operação do algoritmo. De qualquer forma, alguns cuidados são necessários, como: padrões nos textos planos não devem aparecer nos textos cifrados; a entrada de dados no algoritmo deve ser aleatória; a manipulação do texto plano pela introdução de erros no texto cifrado deve ser evitada; e por fim, deve ser possível cifrar com segurança mais de um texto plano com a mesma chave [10].

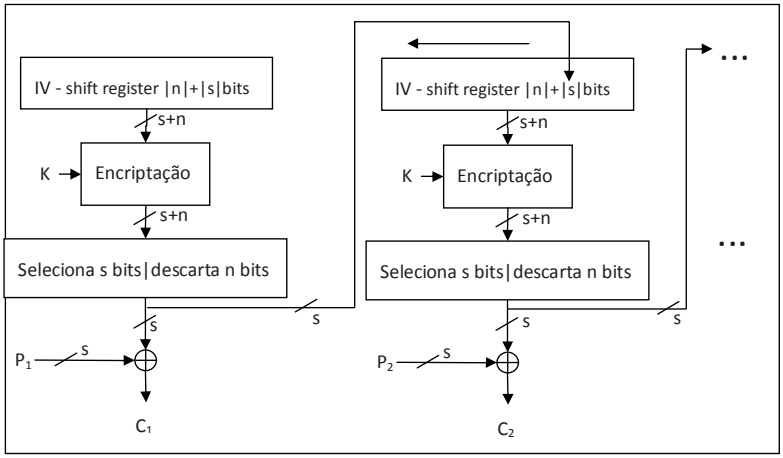

FIGURA 2.3 – Modo OFB [1].

Além da segurança, a eficiência do Modo de Operação não deve ser significantemente inferior à cifração dos dados e em certas circunstâncias o texto cifrado deve ter o mesmo tamanho do texto plano.

A terceira consideração é a tolerância a falhas. Algumas aplicações necessitam serializar a cifração e decifração dos dados enquanto outras necessitam cifrar a maior quantidade de dados possíveis ao mesmo tempo. Outras ainda necessitam recuperar bits do texto cifrado que, devido a ruídos no canal, tenham sido alterados. Cada Modo de Operação tem um diferente conjunto de características e aplicabilidades que atendem mais, ou menos cada um destes critérios[12].

2.3 Componentes Utilizados em Cifradores Simétricos

A estruturação de algoritmos de cifração modernos foi introduzida por Claude Shannon, sob a ótica de que um criptoanalista, conhecendo algumas características estatísticas do sistema, poderia quebrar a chave ou mesmo o texto cifrado. A partir desta idéia, surgiram os conceitos de difusão e confusão aplicados aos algoritmos criptográficos [13].

O princípio de confusão procura eliminar a relação estatística entre o texto cifrado e a chave [13]. Por outro lado, o princípio de difusão consiste em dissipar

as características estatísticas do texto plano, através de operações de permutação. Assim, padrões existentes nos textos planos não aparecem nos textos cifrados. Um dos primeiros componentes de difusão criado foi a *Rede Feistel*. Ela se propõe a gerar um cifrador seguro através da composição de dois ou mais cifradores simples de substituição. Inicialmente a rede divide o bloco de dados a ser cifrado em duas partes, L_0 e R_0. Em cada rodada n, o algoritmo recebe os valores resultantes da rodada anterior L_{n-1} e R_{n-1}. Conforme mostra a Figura 2.4, todas as rodadas têm a mesma estrutura e são parametrizadas com sub-chaves K_n advindas da chave original. Na rodada n, aplica-se uma função F sobre R_{n-1}, na forma $F(R_{n-1}, K_n)$. Ao final da rodada faz-se uma operação de XOR entre L_{n-1} e $F(R_{n-1}, K_n)$. O resultado desta operação gera R_n. O valor de L_n é o próprio R_{n-1} [1].

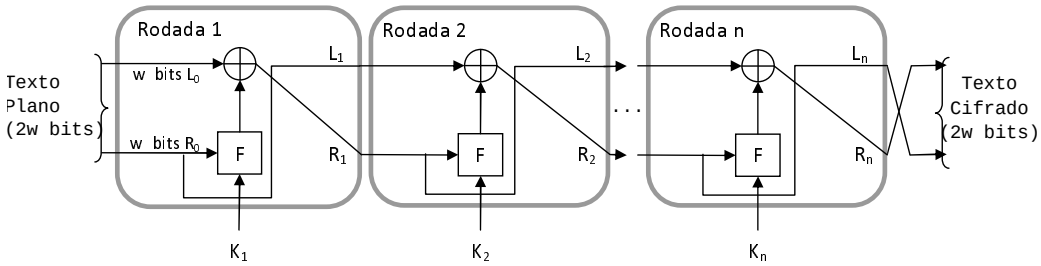

FIGURA 2.4 – Estrutura da Rede Feistel.

Um outro mecanismo muito utilizado para promover difusão em cifradores de bloco como o AES, são as *caixas de substituição*, aqui referidas como Caixa-S. Essas caixas são formadas por uma matriz composta de símbolos, que podem ser criados dinamicamente através de um algoritmo ou previamente definido. Com uma de suas principais funções é introduzir não linearidade no algoritmo, os valores contidos nelas são cuidadosamente escolhidos para gerar o máximo de não linearidade possível. Isso é importante para evitar ataques como criptoanálises linear e diferenciais, discutidos mais adiante. No caso da cifração, quando um dado é operado em uma Caixa-S, o resultado desta operação gera um símbolo correspondente ao dado de entrada. Para inverter a operação aplica-se uma segunda matriz que realiza a operação inversa, retornando o dado cifrado anteriormente.

A difusão ocorre quando varias Caixas-S são aplicadas repetidas vezes sobre o mesmo texto plano. Com isso, mesmo sendo uma operação determinística, a

mudança de um bit no texto plano pode gerar um texto cifrado totalmente diferente. A combinação de várias operações com Caixas-S eleva a incerteza, de forma que, a obtenção do texto plano se torna impraticável. Somente dispondo de vários textos planos e cifrados, e com ataques estatísticos muito refinados é, possível tentar desvendar a chave ou um texto plano que foi cifrado.

São recomendadas algumas diretrizes para criar uma Caixa-S. Ela deve ser bijetiva e não linear. É altamente recomendado que ela respeite o *Strict Avalanche Criterion* (SAC). Segundo este critério, a mudança de um bit do texto de entrada deve mudar, pelo menos, cinqüenta por cento dos bits de saída da Caixa-S [29]. Outro critério importante é o *output Bits Independence Criterion* (BIC). Segundo o qual, quando aplicado o SAC, os de bits do texto cifrado devem variar independentemente uns dos outros [30]. Por fim, outro critério desejável é a eqüiprobabilidade de entrada e saídas. Segundo este critério, qualquer entrada deve gerar como saída qualquer símbolo do alfabeto com a mesma probabilidade. O ataque diferencial explora justamente falhas neste critério. Existem outros critérios que podem ser avaliados, porém estes são os principais.

Geralmente uma Caixa-S gera na saída a mesma quantidade de bits que recebeu na entrada porém, isso não é necessariamente uma regra [31]. Um exemplo de aplicação de Caixas-S é mostrado na Figura 2.5. Neste exemplo, o dado hexadecimal 19 gera o código cifrado $D4$. Na mesma figura esta a Caixa-S inversa onde, a partir do código cifrado $D4$, se recupera o dado hexadecimal 19.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Cifragem

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Decifragem

FIGURA 2.5 – Exemplo genérico de uma Caixa-S (utilizada na cifração) e sua Caixa-S inversa (utilizada na decifração) [2].

As Caixas-S são geralmente utilizadas em cifradores de bloco, por outro lado, os cifradores de fluxo utilizam com freqüência um componente chamado *Linear Feedback Shift Register* (LFSR). Este componente é um vetor de dados onde cada unidade do vetor pode ser um bit, byte ou word. O LFSR combina rotação de cada unidade que o compõe com algumas operações como XOR ou ADD (adição). Por ser um componente linear, ele é utilizado junto com funções não lineares para aumentar a segurança do cifrador. Geralmente utiliza-se alguns LFSR e a chave como entrada de uma função não linear. A saída desta função é combinada com o texto plano através de uma operação de XOR. O resultado desta última operação é o texto cifrado. Na primeira operação dos LFSR, se utilizam vetores de inicialização (IVs). Estes vetores são préviamente fixados com valores conhecidos apenas pelos emissor e receptor. Existem duas formas de alimentar os LFSR nas operações subseqüentes . Uma delas é através da combinação de uma função aleatória (*clock* irregular). A outra alternativa combina os LFSR da entrada da função não linear de forma que, além de gerar bits para função não-linear, estes mesmos bits re-alimentam os próprios LFSR, como num ciclo (auto-sincronizados). Apesar de lineares, quando combinados com funções não lineares geram bons cifradores, além de serem simples e com baixo custo computacional [32]. O Sosemanuk utiliza este componente em sua arquitetura.

A inicialização dos componentes que formam um algoritmo geralmente é aleatória. Mesmo assim, existem muitas instruções que também poderiam ser aleatórias, mas por questões de otimização e segurança, têm valores pré-determinados. Um exemplo são as Caixas-S, as quais têm seus valores cuidadosamente escolhidos. Mesmo que existam cifradores que gerem Caixas-S dinamicamente, a maioria deles possuem Caixas-S fixas. Além das Caixas-S, existem outros exemplos de componentes com valores pré-determinados e dispostos de maneira a maximizar difusão e confusão na cifração. Estes componentes são específicos de cada cifrador. De forma geral, são componentes como multiplicação do bloco de dados por um valor fixo, rotação de bytes dentro de um bloco de dados com um número fixo de bytes para esquerda ou para direita, ou ainda multiplicação de matrizes agregando pesos diferentes para cada linha das matrizes multiplicadas, entre outros. Todos cuidadosamente pensados para gerar o máximo de confusão e difusão.

2.4 Criptoanálise

Criptoanálise é a ciência de recuperar textos planos sem acesso direto à chave. O sucesso do criptoanalista se dá quando ele recupera os dados originais ou a chave. Uma suposição fundamental em criptoanálise foi definida por Kerckhoffs [11], segundo ele, o segredo da cifração dos dados deve residir exclusivamente na chave. Existem sete tipos gerais de ataques. Todos assumem que o criptoanalista tem total conhecimento do algoritmo de cifração [10]. No que se segue, E representa o algoritmo de cifração, D o algoritmo de decifração, P^j são os textos planos e $C^j = E(P^j, K)$, suas versões cifradas com a chave K.

1. Ataque por Texto Cifrado: O criptoanalista tem vários textos cifrados com o mesmo algoritmo E. Ele tenta deduzir os textos planos ou a chave K, comparando varias mensagens cifradas com a mesma chave.

   Dados: $C^1 = E(P^1, K), C^2 = E(P^2, K), ..., C^j = E(P^j, K)$.

   Deduz-se: $P^1, P^2, \ldots, P^j$, K, ou um algoritmo para inferir P^{j+1} de $C^{j+1} = E(P^{j+1}, K)$.

2. Ataque por Texto Plano: O criptoanalista tem acesso a várias mensagens e seus respectivos textos cifrados. Ele tem como meta deduzir as chaves ou um algoritmo que quebre a codificação de outras mensagens cifradas com as chaves usadas nos textos cifrados de sua posse.

   Dados: P^1 e $C^1 = E(P^1, K), P^2$ e $C^2 = E(P^2, K), ..., P^j$ e $C^j = E(P^j, K)$.

   Deduz-se: K, um algoritmo para inferir P^{n+1} de $C^{j+1} = E(P^{j+1}, K)$, ou ambos.

3. Ataque por Texto Plano Escolhido: O criptoanalista tem acesso a um conjunto de textos cifrados e seus respectivos textos planos, mas também escolhe quais textos planos deseja cifrar.

   Dados: P^1 e $C^1 = E(P^1, K), P^2$ e $C^2 = E(P^2, K), ..., P^j$ e $C^j = E(P^j, K)$. Onde $P^1, P^2, ..., P^j$ são escolhidos.

   Deduz-se: K, um algoritmo para inferir P^{n+1} de $C^{j+1} = E(P^{j+1}, K)$, ou ambos.

4. Ataque por Texto Plano Escolhido e Adaptado: é um caso especial de Ataque por Texto Plano Escolhido, pois o criptoanalista pode subdividir a escolha dos pares

de textos cifrados e planos, desta forma a escolha de um subconjunto destes textos é baseada no resultado da cifração obtida de um subconjunto anterior.

5. Ataque por Texto Cifrado Escolhido: O criptoanalista pode escolher diferentes textos cifrados e ter acesso a seus respectivos textos planos. Seu trabalho é deduzir a chave. Esse tipo de ataque é comumente aplicado em cifradores de chave pública e algumas vezes efetivo contra cifradores simétricos.

   Dados: C^1 e $P^1 = D(C^1, K), C^2$ e $P^2 = D(C^2, K), ..., C^j$ e $P^j = D(C^j, K)$.

   Deduz-se: K.

6. Ataque por Chave Escolhida: Esse ataque não significa que o criptoanalista pode escolher a chave desejada para cifrar um texto plano. Nele o criptoanalista tem algum conhecimento da relação existente entre as chaves geradas para cifrar os textos. Através desse conhecimento ele gera um conjunto de textos cifrados e tenta de alguma maneira encontrar as chaves utilizadas na cifração. É um ataque obscuro que depende da quantidade e qualidade das informações que o criptoanalista tem sobre as chaves. Não é um ataque prático.

7. Ataque por Intimidação: Conhecido como engenharia social, usa de métodos como suborno, ameaças ou tortura para se conseguir a chave.

Ataque por Texto Plano Conhecido e por Texto Plano Escolhido são comumente utilizados pela facilidade de acesso a cifração de textos, possibilitando a introdução dos textos planos desejado. David Kahn relata exemplos históricos em seu livro a respeito destes ataques [11]. O que facilita a quebra do algoritmo através destes dois métodos de ataques é o fato de, na troca de mensagens, existirem padrões que se repetem como cabeçalhos e finalizadores de mensagens, estruturas internas de códigos fonte, programas executáveis, trechos de texto, imagens, etc.

Baseados nesses tipos gerais de ataques, foram criadas técnicas de criptoanálise que exploram fragilidades de determinados grupos de cifradores. A seguir são apresentadas algumas das técnicas modernas mais aplicadas na quebra dos cifradores avaliados neste trabalho. De uma forma ou de outra, todas elas se utilizam de estatística para realizar a criptoanálise dos cifradores.

- Ataque Diferencial: Esta técnica compreende um conjunto de diferentes ataques. Todos utilizam o princípio de comparar textos cifrados. Seus respectivos textos planos têm pequenas diferenças e são cifrados com a mesma chave.

  Esta técnica foi inicialmente aplicada a cifradores de bloco e posteriormente expandida a cifradores de fluxo e funções hash. Ela se caracteriza por ser um ataque de texto plano escolhido. Sendo $P = (P^1, P^2, ...P^n)$ os textos planos e $C = (C^1, C^2, ...C^n)$ os respectivos textos cifrados, se juntam em pares P e C através de uma operação $\oplus$ (XOR) na forma $P'_m = P^m \oplus P^{m+1}$ e $C'_m = C^m \oplus C^{m+1}$. O conjunto de textos planos têm poucas diferenças previamente escolhidas. Após a escolha dos textos planos e a obtenção dos respectivos textos cifrados, analisa-se estatísticamente possíveis desbalanços de cifração no conjunto C'_m, ou seja, como o conjunto P tem poucas diferenças, um bom cifrador deve gerar um conjunto C bem diversificado (balanceado). Com esta técnica, o criptoanalista explora estatísticamente algum padrão de P' e C' que apareçam com mais freqüência [33].

- Ataque por escorregamento (*slide attack*): Ele é utilizado sobre cifradores que podem ser dividido em partes iguais. Desta forma, o ataque trata essas partes como uma função F_i que produz saídas mediante determinadas entradas. Esta característica indica que o algoritmo utiliza ciclicamente a chave, dando indícios de vulnerabilidade à ataques por texto plano escolhido. Além disso, este ataque é independente do número de rodadas de cifração aplicadas ao texto plano e pode ser utilizado em cifradores de bloco ou de fluxo. Assumindo que o cifrador divide a chave em K_n partes e a mesma função F_i é aplicada sobre cada parte da chave, o criptoanalista escolhe $2^n/2$ pares de textos planos e cifrados, cada par denotado por (P, C). A idéia é encontrar a função exata que torne verdadeiras as equações $P^0 = F(P^1, K^1)$ e $C^0 = F(C^1, k^1)$, para os pares de texto $((P^0, C^0), (P^1, C^1))$. Este ataque mostrou-se eficiente contra cifradores DES-X, Brown-Seberry, Even-Mansour e cifradores que utilizam redes Feistel [34, 35].

- Ataque Linear: É classificado como um tipo de Ataque por Texto Plano. Ele tenta linearizar o comportamento do cifrador através da criação de funções lineares de entrada e saída utilizando operações XOR na forma $P_{i1} \oplus P_{i2}... \oplus P_{in} \oplus C_{i1} \oplus C_{i2} \oplus ...C_{in} = 0$. Onde P_{ij} é o j-ésimo bit do byte i do texto plano e C_{ij} é o

j-ésimo bit do byte i do respectivo texto cifrado. As relações necessárias para se obter uma aproximação linear efetiva são:

  - Escolhe-se um subconjunto dos bits da entrada e calcula-se a paridade (operação de XOR) dos mesmos;
  - Escolhe-se um subconjunto dos bits da saída e calcula-se a paridade dos mesmos;
  - A análise acima é repetida até que todos os subconjuntos de entrada e saída sejam verificados;
  - Os valores acima são tabelados de tal forma que as linhas da tabela contenham os subconjuntos possíveis de entrada, enquanto as colunas contenham os subconjuntos possíveis de saída;
  - As entradas desta tabela representaram o número de vezes que para um dado subconjunto de bits de entrada, a paridade do mesmo é igual a paridade do subconjunto de bits de saída correspondente;

  A partir dessa tabela, se relaciona a probabilidade de ocorrência de valores das saídas geradas, diante dos resultados das entradas geradas em rodadas internas do algoritmo. Se a tabela mostrar algumas relações (de linha de bits de entrada com colunas de bits de saída), que se destacam por ter maior ocorrência, é possível inferir a chave utilizada pelo cifrador [36]. Esse ataque foi inicialmente aplicado ao cifrador DES, mas pode ser extendido a outros cifradores.

- Ataque por Correlação: Este ataque é aplicado à cifradores de fluxo. Ele busca descobrir a chave através da relação existente entre os primeiros bits de entrada e o resultado da cifração destes bits. Muitas vezes, tal relação é procurada por comparação exaustiva de textos planos e cifrados. Uma forma de aplicar este ataque é em sistemas pequenos. Aplicações *Bluetooth*, são um bom exemplo pois, contém uma grande quantidade de mensagens pequenas [16]. Existem outras formas mais eficientes de criptoanálise por correlação, um delas é definida em [37]

2.5 Algoritmo AES

Com a evolução das máquinas, uma chave de 56 bits, como era normalmente usada, não poderia mais garantir a segurança desejada. Portanto, em 1997, o National Institut of Standards and Technology (NIST), órgão do governo americano, iniciou um processo de escolha do sucessor do algoritmo DES até então utilizado [3]. O algoritmo escolhido foi o Rijndael, criado por Joan Daemen e Vicent Rijmen [27]. Após a escolha, este algoritmo foi batizado de AES (Advanced Encryption Standard) e adotado como padrão de cifração do governo americano para cifradores simétricos.

O AES é um cifrador simétrico de blocos, originalmente projetado para cifrar blocos de texto plano de 128, 192 e 256 bits. Diferente da maioria dos cifradores modernos, ele não usa Redes Feistel. Para garantir a difusão dos dados cifrados, ele utiliza camadas de passos que misturam linearmente os dados e Caixas-S, como visto mais adiante [27]. Para aplicar o conceito de confusão, o AES utiliza um gerador de sub-chaves e a função AdicionaChaveRodada. O tamanho das chaves também varia entre 128, 192 e 256 bits [3]. Ele trabalha de forma iterativa, isto é, as mesmas operações são realizadas varias vezes sobre o mesmo grupo fixo de bytes, provocando assim a difusão dos dados [15, 2, 27].

Segundo Daemen e Rijmen, cada uma dessas operações pode ser divididas em funções. Denomina-se *rodada* cada uma das vezes que o conjunto total de operações é aplicado sobre o texto plano. Este conjunto de operações é mostrado na Figura 2.6.

A chave é subdividida em várias sub-chaves, cada uma delas é usada em uma rodada. O número de rodadas varia conforme o tamanho da chave e do bloco de dados. Entretanto, um número mínimo de rodadas é necessário para garantir confusão e difusão suficientes para resistir a ataques criptográficos [38]. A Tabela 2.1 mostra a relação de rodadas e tamanho da chave utilizadas pelo AES.

TABELA 2.1 – Número de Rodadas em Função do tamanho dos blocos e das chave no AES [4].

	Tamanho da chave	Tamanho do bloco	Número de rodadas
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

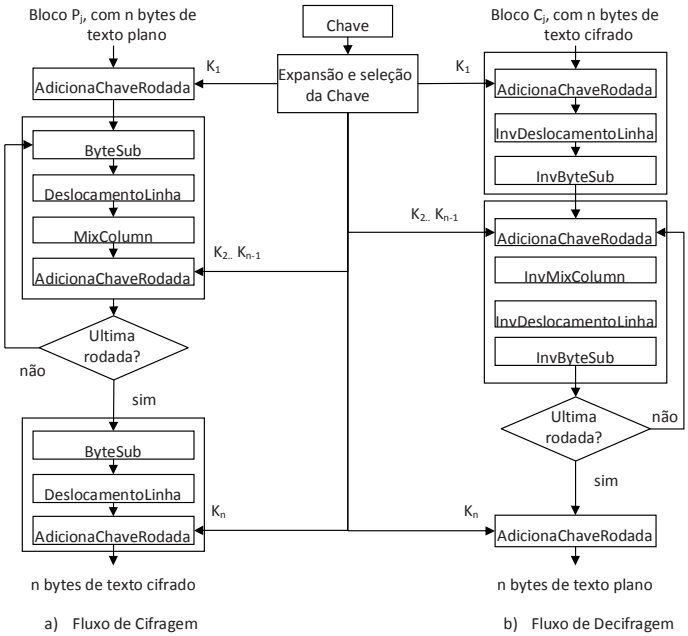

FIGURA 2.6 – Fluxo de cifração e decifração do Algoritmo AES [3].

O processo se inicia com a função geradora de sub-chaves a qual se divide em duas rotinas: expansão da chave e seleção da chave. Na expansão da chave, as primeiras N_k palavras de 32 bits contém a chave original. O restante das sub-chaves são geradas recursivamente através de dois algoritmos, um para chaves de origem menores ou iguais a 128 bits e outro para chaves de origem maiores que 128 bits. A descrição deste algoritmo, escrito em linguagem C, pode ser vista em [38]. Após a geração do conjunto de sub-chaves o AES entra no ciclo de cifração propriamente dito. Selecionando, a cada rodada, seis palavras consecutivas do vetor de sub-chaves.

Conforme Berent em [2] e Daemen em [27], o ciclo de cifração é composto das seguintes funções mostradas na Figura 2.6:

- AdicionaChaveRodada: Uma operação de XOR é realizada entre o bloco de texto plano e a sub-chave da rodada, a qual possui o mesmo tamanho do bloco de dados.

- ByteSub: Os bytes de cada bloco, resultante da operação AdicionaChaveRodada são substituídos por seus equivalentes em uma tabela de substituição (Caixa-S), seguindo a mesma metodologia descrita na Seção 2.3. Esta operação foi projetada

para atuar contra ataques lineares, diferenciais e de interpolação [39].

- DeslocamentoLinha: Os bytes de entrada da função são rotacionados em grupos de 4 bytes S_{ij} com $i, j = (0, 1, 2, 3, ...)$. Esta rotação depende do tamanho do bloco. A Figura 2.7 mostra um exemplo de rotação de bloco de 16 bytes (128 bits) divididos em blocos de 4 bytes. Nota-se que a primeira parte do bloco não é rotacionada, a segunda é rotacionada de 1 byte, a terceira de 2 bytes e a parte final de 3 bytes. Esta função protege o algoritmo contra ataques Multiset [39]. Além de aumentar a difusão dos dados [3].

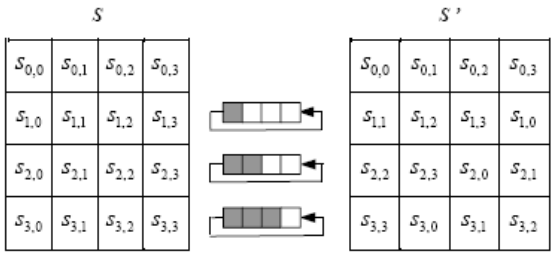

FIGURA 2.7 – Operação de DeslocamentoLinha usada no algoritmo AES [4].

- MixColumns: Tomando como base o exemplo da Figura 2.7, e sendo $0 \leq c <$ número de blocos, ela pode ser definida como uma operação matricial conforme:

$$\begin{pmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix}$$

A escolha da matriz é fixa e pré-definida, de modo a favorecer a segurança do cifrador. Tomando $S' = A \otimes S$, como o resultado desta multiplicação, os quatro bytes das colunas são trocados por:

$$S'_{0,c} = (02 \cdot S_{0,c}) \oplus (03 \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$$

$$S'_{1,c} = S_{0,c} \oplus (02 \cdot S_{1,c}) \oplus (03 \cdot S_{2,c}) \oplus S_{3,c}$$

$$S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (02 \cdot S_{2,c}) \oplus (03 \cdot S_{3,c})$$

$$S'_{3,c} = (03 \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (02 \cdot S_{3,c})$$

Isto proporciona a cada byte do grupo influenciar todos os outros bytes, conforme pode ser visto na Figura 2.8, o que provoca grande difusão entre os dados [27].

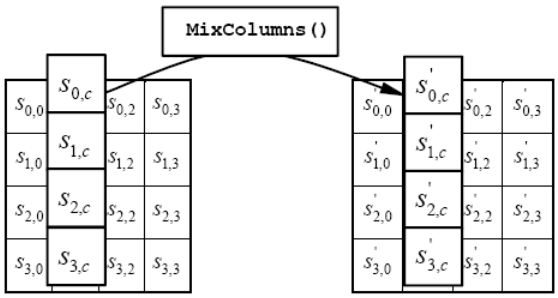

FIGURA 2.8 – Operação de MixColumns no algoritmo AES, sendo N_k o número de bytes da chave e Nb o número de bytes de cada bloco [4].

Conforme dito anteriormente, o AES utiliza-se de rodadas sobre operações simples para cifrar e decifrar os dados. Como ilustra a Figura 2.6 a cifração inicia-se com uma operação de AdicionaChaveRodada, entre o bloco de texto plano P_j e a parte K_n da chave total. O resultado é a entrada de uma Rodada. Na cifração, uma rodada é composta pelas funções básicas descritas anteriormente: ByteSub, DeslocamentoLinha, MixColumns e AdicionaChaveRodada, executadas na respectiva ordem. O resultado de cada operação é o valor de entrada da operação seguinte. Ao final, a função AdicionaChaveRodada faz uma operação de XOR entre o resultado de MixColumns e a parte K_{n+1} da chave. O resultado passa a ser o inicio de uma nova Rodada. A quantidade de rodadas é definida pelo tamanho da chave e dos blocos de dados, conforme a Tabela 2.1. Nota-se que ao final, a última rodada não realiza a operação de MixColumns.

A decifração (ver Figura 2.6) é realizada aplicando-se as funções inversas a ByteSub, DeslocamentoLinha e MixColumns. Inicialmente, um bloco P_j

de bytes cifrados é aplicado a entrada da primeira rodada. A função AdicionaChaveRodada executa uma operação de XOR com a parte final K_n da chave. Ainda na primeira rodada, aplica-se as funções inversas InvDeslocamentoLinha e InvBytesSub. A partir da segunda rodada até a final são aplicadas as funções AdicionaChaveRodada, InvMixColumns, InvDeslocamentoLinha, InvByteSub. A última operação, AdicionaChaveRodada, aplica um XOR entre a saída da Rodada final e a parte inicial K_1 da chave. As operações executadas pelas funções inversas são computacionalmente mais custosas. Como conseqüência, a decifração é mais demorada que a cifração.

2.6 Algoritmo Sosemanuk

O Sosemanuk é um cifrador de fluxo que utiliza tamanhos de chave de 128 bits. Surgiu como um candidato do projeto *eEstream*, organizado por um convênio de empresas e universidades européias chamado de *ECRYPT*. Este projeto visa identificar cifradores de fluxo promissores, e que possam ser aplicados comercialmente [28]. Atualmente ele está na fase 3 de avaliação do projeto *eEstream*. Conforme as análises feitas até o momento, ele demonstra ser um cifrador simétrico muito promissor.

O Sosemanuk utiliza princípios básicos do cifrador Snow 2.0, e transformações derivadas do cifrador Serpent, ambos descritos abaixo. Tem como principais características a utilização de uma rotina muito rápida de preenchimento do vetor de inicialização IV e o uso de uma reduzida quantidade de dados estáticos. Estas características mantém seu desempenho constante para diferentes *hardwares*. Além disso, o compartilhamento de técnicas de diferentes algoritmos de cifração aumenta a segurança, principalmente para ataques de correlação. Mais ainda, o uso de pouca memória somado ao descarte freqüente de dados em memória sujeitos a exploração, reforça sua segurança.

Para entender o funcionamento do Sosemanuk devemos conhecer alguns detalhes dos dois cifradores nos quais ele é baseado:

Serpent: É um cifrador de blocos de 128 bits que divide-os em quatro blocos de 32 bits. O Sosemanuk utiliza algumas rodadas desde algoritmo. Uma rodada do Serpent é a aplicação dos 4 blocos de entrada sobre oito Caixas-S distintas. Seu

resultando são 4 novos blocos de 32 bits de saída. Em cada rodada do Serpent não existem transformações lineares. Mais detalhes sobre a transformação linear e o funcionamento completo do Serpent podem ser vistos em [40].

Snow: Assim como a maioria dos cifradores de fluxo, o Snow utiliza o LFSR. Geralmente, cada cifrador utiliza seu próprio conjunto de LFSRs. Entretanto, o Sosemanuk utiliza um LFSR similar ao algoritmo Snow [5].

Além das Caixas-S do Serpente e o LFSR modificado do Snow, o Sosemanuk utiliza também uma máquina de estados finitos (MEF). A interação entre estes três componentes pode ser vista na Figura 2.9. Como foi mencionado, o Sosemanuk processa unidades de 32 bits de textos planos. Os registradores $R1$ e $R2$ são a área de entrada de textos planos, eles recebem 32 bits de texto plano cada um.

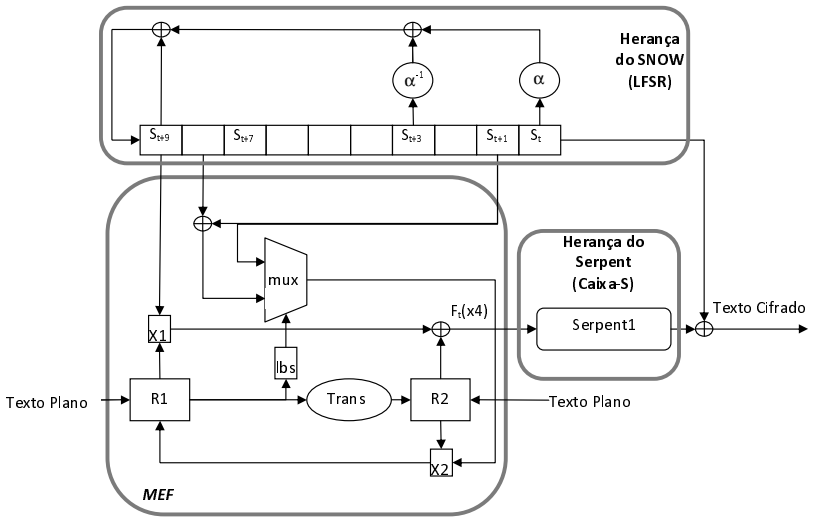

FIGURA 2.9 – Processo de cifração do Algoritmo Sosemanuk [5].

O Sosemanuk combina um LFSR com operações de rotação, XOR, $\alpha(X)$ e $\beta(X)$. As operações $\alpha(X)$ e $\beta(X)$ são definidas como sendo:

$\alpha(X) = X^4 + \beta^{23}(X)X^3 + \beta^{245}(X)X^2 + \beta^{48}(X)X + \beta^{239}(X)$
$\beta(X) = X^8 + X^7 + X^5 + X^3 + 1$
$X =$ Posição S_0 e S_3 do LFSR

A MEF pode ser vista na Figura 2.9, e é definida como:

$$MEF_t(R1_{t-1}, R2_{t-1}, s_{t+1}, s_{t+8}, s_{t+9}) \rightarrow (R1_t, R2_t, F_t) \quad , onde \tag{2.2}$$

$R1_t = (R2_{t-1} + mux(lsb(R1_{t-1}), s_{t+1}, s_{t+1} \oplus s_{t+8}))mod2^{32}$,
$R2_t = Trans(R1_{t-1})$,
$F_t = (s_{t+9} + R1_t mod2^{32}) \oplus R2_t$.

Sendo $lbs(x)$ o bit menos significativo de x e $mux(c, x, y)$ igual x se $c = 0$ ou y se $c = 1$. A função $Trans$ é definida como $Trans(z) = (M \times z \; mod2^{32}) <<< 7$. M é a constante hexadecimal 54655307, propositalmente escolhida para aumentar a segurança do algoritmo. A operação $<<< 7$ é o deslocamento de bits para esquerda em 7 unidades. As funções $X1$ e $X2$ são operações de XOR definidas como:

$X1 = s_{t+9} + R1_t mod2^{32}$,
$X2 = (R2_{t-1} + mux(lbs(R1_{t-1}), s_{t+1}, s_{t+1} \oplus s_{t+8}))mod2^{32}$.

O processo de inicialização da MEF e a entrada da chave ocorre em duas partes. A primeira chamada de *key schedule* utiliza o Serpent24. O Serpent24 é uma variação do Serpent. Ele utiliza apenas 24 rodadas para cifrar os dados, ao invés das 32 normalmente utilizadas. Nesta fase a chave do usuário é injetada no Serpent24 que gera 25 sub-chaves de 128 bits cada. O Serpent aceita chaves de 1 a 256 bits porém, o Sosemanuk foi projetado para trabalhar com chaves de 128 bits. Desta forma, a entrada de chaves superiores a 128 bits não garante o aumento da segurança na cifração [5].

A segunda parte do processo utiliza novamente o Serpent24, previamente inicializado pelo processo *key schedule*. Nesta fase, a entrada do algoritmo é o vetor de inicialização IV de 128 bits, o qual é compartilhado secretamente por ambos: emissor e receptor. Os valores da décima segunda $(Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12})$, décima oitava $(Y_3^{18}, Y_2^{18}, Y_1^{18}, Y_0^{18})$ e vigésima quarta rodadas $(Y_3^{24}, Y_2^{24}, Y_1^{24}, Y_0^{24})$ são utilizados para inicializar a MEF da seguinte maneira:

$(s_7, s_8, s_9, s_{10}) = (Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12})$

$(s_5, s_6) = (Y_1^{18}, Y_3^{18})$
$(s_1, s_2, s_3, s_4) = ((Y_3^{24}, Y_2^{24}, Y_1^{24}, Y_0^{24})$
$R1_0 = Y_0^{18}$
$R2_0 = Y_2^{18}$

A cada rodada, novas unidades de texto plano de 32 bits cada são inseridos nos registradores $R1$ e $R2$. Após a MEF gerar quatro f_t, se aplica o Serpent1 sobre $(f_t, f_{t+1}, f_{t+2}, f_{t+3})$. A exemplo do Serpent24, o Serpent1 é a aplicação de uma rodada do Serpent sobre os dados de entrada. A saída do Serpent1 é combinada com as quatro primeiras posições do $LSRF$ através de uma operação de XOR. Conforme segue:

$$(R_0, R_1, R_2, R_3) = Serpent1(f_t, f_{t+1}, f_{t+2}, f_{t+3} \oplus (S_t, S_{t+1}, S_{t+2}, S_{t+3})$$

Os valores resultantes desta operação são a saída do cifrador. Na decifração a inicialização da MEF é idêntica a cifração. O processo de decifração aplica as operações da MEF na ordem invertida para extrair o texto plano.

Como observado, o Sosemanuk procura unir a segurança do conhecido cifrador de blocos Serpent a outros padrões de projeto de cifração. Estes padrões são: não aplicação direta de IV e da chave na cifração; o uso de um tipo de LFSR; rápida inicialização; rápida cifração e uso de pouca memória. Mesmo assim, ele tem alguns problemas. Um deles é a não garantia de que chaves superiores à 128 bits promovam maior segurança. Além disso, o Sosemanuk ainda esta sobre avaliação. Mesmo com os promissores resultados dos estudos preliminares contra ataques de correlação, entre outros, ainda não se pode garantir que ele seja um cifrador totalmente seguro. Serão necessários mais estudos e testes para que o Sosemanuk tenha as garantias de segurança que o AES provê, após anos de estudos e aplicações práticas.

2.7 Algoritmo RC4

O RC4 é um cifrador de fluxo amplamente utilizado no mundo. Embora já exista uma nova versão do algoritmo (RC5) e, atualmente, outros cifradores mais seguros e velozes, ele ainda é muito utilizado [41]. Seu funcionamento é simples, ele tem uma

fase de inicialização da chave chamada de *key-scheduling algorithm* (KSA). O RC4 aceita chaves de 1 a 256 bits. Ele utiliza um vetor S que é primeiramente inicializado em sua totalidade, conforme mostra a Equação 2.3.

$$\begin{aligned} &for\ \ i\ \ from\ \ 0\ \rightarrow 255 \\ &\quad S[i] := i \\ &end\ \ for \end{aligned} \tag{2.3}$$

Após, o vetor S é processado com 256 iterações do algoritmo PRGA e, ao mesmo tempo, combina o resultado do algoritmo PRGA aos valores da chave. A Equação 2.4 mostra a segunda parte do processo de inicialização do RC4.

$$\begin{aligned} &j := 0 \\ &for\ \ i\ \ from\ \ 0\ \ to\ \ 255 \\ &\quad j := (j + S[i] + key[i\ \ mod\ \ \textbf{\textit{keylength}}])\ \ mod 256 \\ &\quad swap(S[i], S[j]) \\ &end\ \ for \end{aligned} \tag{2.4}$$

sendo $keylength$ o tamanho da chave.

O fluxo principal de cifração também utiliza o algoritmo FRGA. A cada iteração, o algoritmo PRGA modifica o seu estado interno e gera um byte de saída. O FRGA permanece ativo até que todos os valores do texto plano ($input$) sejam processados. A saída é o resultado de uma operação de $\oplus$ (XOR) entre o byte de entrada e o módulo de 256 do valor do vetor S, na posição $S[i] + S[j]$. Cada valor da posição $S[i]$ é trocado com a posição $S[j]$, a cada 256 iterações. Essa troca não é uma regra fixa, existem variações do RC4 que fazem a troca com mais freqüência, o que aumenta a segurança do algoritmo [42, 41]. A Equação 2.5 mostra o processo de cifração do RC4.

$$
\begin{array}{l}
i := 0,\ \ j := 0,\ \ idx := 0 \\
while\ \ idx < input.length \\
\quad i := (i + 1)\ \ mod\ \ 256 \\
\quad j := (j + S[i])\ \ mod\ \ 256 \\
\quad swap(S[i], S[j]) \\
\quad output\ \ S[(S[i] + S[j])\ \ mod\ \ 256] \oplus input[idx] \\
\quad idx := idx + 1 \\
end\ \ while
\end{array}
\tag{2.5}
$$

Embora seja muito utilizado e de construção simples, o RC4 é alvo de diversos ataques criptoanalíticos como os citados em [43, 44]. O RC4 não utiliza LFSR, que é um elemento comun em cifradores de fluxo. Para gerar símbolos cifrados, ele implementa uma máquina de estados sobre operações simples, iteradas em um número finito de rodadas. Desta forma, ele utiliza a base de cifradores de fluxo, a qual é gerar símbolos pseudo-aleatórios, baseados em um estado inicial e chave secreta, combinados com os símbolos planos.

Capítulo 3

Dinâmica Caótica Aplicada à Criptografia

Neste Capítulo, serão considerados dois algoritmos de cifração/decifração baseados em mapas caóticos. O primeiro é o de Baptista [6], que usa a dinâmica de um mapa logístico. O outro, proposto recentemente por De Oliveira e Sobottka [26], segue a mesma filosofia do algoritmo de Baptista, mas com aperfeiçoamentos importantes. Para tal, será feita uma breve revisão de conceitos em dinâmica caótica de mapas, e depois, as descrições desses dois algoritmos.

3.1 Conceitos Básicos de Mapas Caóticos

Um sistema dinâmico é um conjunto V de estados, denominado de *espaço de estados*, junto com uma regra que determina o estado presente x_n em função dos estados passados $x_{n-1}, x_{n-2}, ...$ Os sistemas dinâmicos podem ser classificados como determinísticos ou estocásticos. Um exemplo de sistema determinístico é dado por uma função ou mapa bem definida $f : V \rightarrow V$ como $x_n = f(x_{n-1})$. Nesta função pode-se determinar o estado atual unicamente observando-se o estado anterior. Nos sistemas estocásticos, o estado atual é definido através de uma regra aleatória. Por exemplo, estando no estado $x_n = 5$, e supondo-se que existem 50% de chances de $x_{n+1} = 6$ e outros 50% de chances de $x_{n+1} = 4$, conforme o resultado do lançar de uma moeda, não existe como determinar com certeza o estado x_{n+1} através de seu estado anterior x_n

[45].

Os sistemas dinâmicos também podem ser classificados como contínuos e discretos. Nos sistemas discretos a variável tempo é incrementada a cada iteração do sistema e assume valores no domínio dos inteiros não negativos, (incluindo o zero), conforme o exemplo citado acima, definido por um mapa $f : V \to V$. Nos sistemas contínuos, a variável tempo assume valores contínuos, isto é, em intervalos dos números dos reais. Talvez os exemplos mais famosos sejam aqueles definidos pela segunda lei de Newton, $\mathbf{F} = m\mathbf{a}$, que relaciona a força que age em um objeto de massa m com a sua aceleração $\mathbf{a}$. Com efeito, esta equação pode ser escrita sob a forma $\ddot{\mathbf{x}} - \mathbf{F}/m = \mathbf{0}$, onde cada ponto define uma derivada no tempo. Esta é uma equação diferencial ordinária, cujas soluções $\mathbf{x}(t)$ são trajetórias que definem qual o estado no tempo contínuo t, dado que em $t = 0$ tem-se a condição inicial $\mathbf{x}(0) = \mathbf{x}_0$.

No caso das aplicações criptográficas consideradas neste trabalho, os sistemas dinâmicos usados são determinísticos, discretos e unidimensionais. Mais precisamente, são dados por uma função ou mapa $f : V \to V$ em um subconjunto $V \subset \mathbb{R}$. Neste trabalho tais sistemas serão denotados genericamente por $\mathcal{S}$. A órbita γ de $\mathcal{S}$ correspondente à condição inicial x_0 é a seqüência $\gamma = (x_0, x_1, x_2, \ldots)$, onde $x_n = f^n(x_0)$. Neste caso, a variável temporal n de $\mathcal{S}$ é discreta, e seus estados são os pontos $x \in V$. As órbitas de $\mathcal{S}$ podem ser abertas, eventualmente periódicas ou periódicas. As *periódicas* são aquelas em que existe $m \in \mathbb{N}$ tal que $x_{n+m} = x_n$ para todo $n \in \mathbb{N}$. O menor valor de m é o *período* da órbita. As *eventualmente periódicas* são aquelas que, a partir de um certo $k \in \mathbb{N}$, tornam-se periódicas. As que não são periódicas, nem eventualmente periódicas, são denominadas de *órbitas abertas* [45].

Portanto, se o mapa $f : V \to V$ é simples, o correspondente sistema dinâmico $\mathcal{S}$ define uma maneira fácil de se obter uma progressão de valores numéricos. A simplicidade é uma das características requeridas para o uso na criptografia. Entretanto, para ser usado na cifração de mensagens, $\mathcal{S}$ precisa apresentar comportamento complexo, pois é isto que garantirá a segurança do sistema criptográfico. A maneira de um sistema determinístico apresentar comportamento complexo é a caoticidade. Segundo Devaney [46], um sistema dinâmico $\mathcal{S}$ definido pelo mapa $f : V \to V$ é *caótico* se (a) exibe sensibilidade às condições iniciais, (b) é *topologicamente transitivo*, e (c) o conjunto C^{per} dos pontos periódicos de f é denso em seu domínio V.

A sensibilidade de $\mathcal{S}$ às condições iniciais implica que as órbitas originadas

em duas condições iniciais próximas, eventualmente ficarão completamente descorrelacionadas, e portanto afastadas uma da outra. Esta é a raiz do comportamento complexo dos sistemas caóticos e da sua imprevisibilidade computacional. A transitividade topológica implica que órbitas típicas visitam todos as partes do domínio V de $\mathcal{S}$. Por causa disto, o afastamento entre órbitas distintas, mesmo que inicialmente próximas, assume valores próximos ao diâmetro do domínio V, o que implica em incerteza total. E, finalmente, a densidade de órbitas periódicas sinaliza a regularidade da dinâmica. Afinal, $\mathcal{S}$ é determinístico. Entretanto, os pontos de C^{per} pertencem a órbitas *instáveis*, ou *repulsoras*. Isto significa que órbitas próximas às órbitas em C^{per} tendem a se afastar delas. Por outro lado, as órbitas abertas de um sistema caótico $\mathcal{S}$ costumam ter um comportamento contrário, ou seja, são *atratoras*. Isto impede o *overflow* computacional.

Talvez o mais importante e popular exemplo de sistema dinâmico discreto unidimensional seja a família dos mapa quadráticos, dados por

$$f_\mu(x) = \mu x(1-x), \tag{3.1}$$

com parâmetro real $\mu > 1$, também conhecidos como *mapas logísticos*. Esta família de mapas exibe, a depender do valor do parâmetro μ, muitos dos regimes dinâmicos observados em sistemas dinâmicos em geral, inclusive regimes caóticos. A seguir, desenvolve-se um estudo mais detalhado da dinâmica dessa família de mapas.

Pode-se verificar facilmente que os mapas logísticos possuem $x_a = 0$ e $x_b = (\mu - 1)/\mu$ como pontos fixos, isto é, com período fixo $m = 1$. Nota-se que, qualquer que seja o valor de $\mu > 1$, o ponto x_b pertence ao interior do intervalo $I = [0,\ 1]$, e o ponto $x_a = 0$ é sempre repulsor. Se $1 < \mu < 3$ então x_b será atrator enquanto que, se $\mu > 3$, será repulsor. No primeiro caso, o mapa f_μ tem uma dinâmica inteiramente dominada por x_b, isto é, para toda condição inicial x_0 no interior do intervalo I, tem-se $\lim_{n\to+\infty} f_\mu^n(x_0) = x_b$, enquanto que fora do intervalo I, tem-se $\lim_{n\to+\infty} f_\mu^n(x_0) = -\infty$.

Quando a barreira $\mu = 3$ é ultrapassada, x_b perde sua estabilidade e surgem, próximo a ele, dois outros pontos periódicos de período 2 estáveis. Tem-se então uma órbita de período 1 (ponto fixo) instável e uma de período 2 estável. Na medida que μ continua crescendo, tais pontos estáveis vão se distanciando um do outro até que perdem

suas estabilidades e dão origem a dois novos pontos cada um, desta vez de período 4 e estáveis. Tem-se agora 3 pontos instáveis (períodos 1 e 2) e 4 estáveis (de período 4). Em termos de órbitas, temos duas instáveis (de períodos 1 e 2) e uma estável (de período 4). Tal seqüência de duplicação de período vai acontecendo mais rapidamente (em relação às variações de μ) até que o número de órbitas instáveis seja infinito. Como se verá mais tarde, é nesse ponto (valor de μ) que se tem caos (conforme definido por Devaney [46]). A Figura 3.1 ilustra essas transições, onde se pode notar a existência de muitas janelas de periodicidade [45]

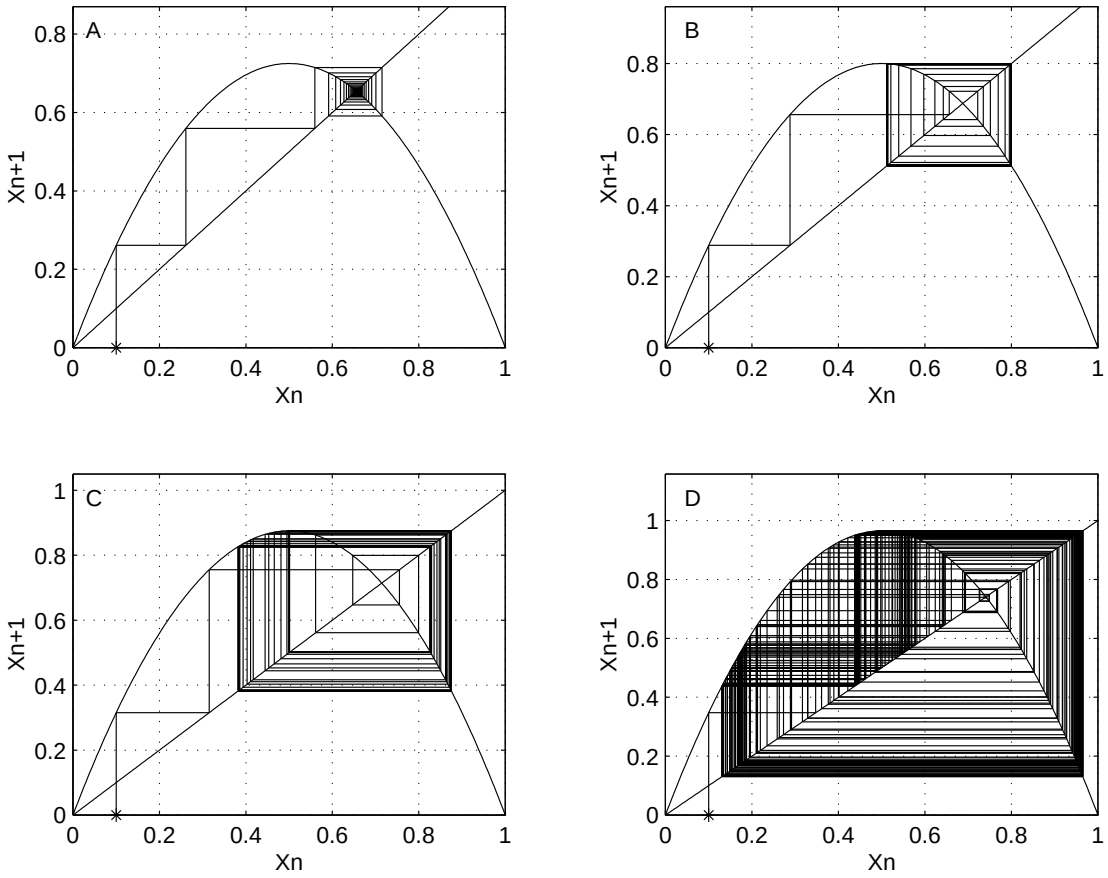

FIGURA 3.1 – Mapas de tabelamento para condição inicial $x_0 = 0.1$ e valores em (A), $\mu = 2,9$ (ponto fixo); em (B), $\mu = 3,2$ (período 2); em (C), $\mu = 3,5$ (período 4) e em (D), $\mu = 3,86$ (caos).

Depois de atingido este primeiro regime caótico, o sistema volta a ficar regular (não caótico). Passa a ter uma só órbita estável que perde sua estabilidade dando origem a duas novas órbitas estáveis que, posteriormente, perdem sua estabilidade dando origem a quatro novas órbitas estáveis e assim por diante, repetindo o comportamento anterior. Desta forma, no intervalo $3 < \mu \leq 4$ o mapa logístico alterna seu comportamento entre caótico e não caótico. As transições sofridas pelo mapa logístico quando se varia o parâmetro μ chamam-se de bifurcações.

De uma maneira geral, chama-se de *bifurcação* qualquer mudança estrutural ou qualitativa sofrida por um sistema dinâmico pela variação de seus parâmetros. As principais mudanças qualitativas sofridas por sistemas dinâmicos são: perda de estabilidade de órbitas periódicas estáveis, surgimento de novas órbitas periódicas, aniquilação de órbitas periódicas e mudanças no conjunto dominante (atrator ou repulsor). A Figura 3.2 mostra as bifurcações do mapa logístico para $2,5 < \mu < 4,0$. Tal gráfico de bifurcações pode ser obtido adotando-se o seguinte procedimento para cada valor de μ. Primeiramente, (a) escolhe-se um conjunto suficientemente 'grande' de condições iniciais no interior do intervalo I, depois (b) obtém-se as órbitas de cada uma dessas condições iniciais iterando-se f_μ um 'grande' número de vezes (então cada órbita tenderá ao mesmo atrator) e, finalmente, (c) descarta-se um 'grande' número de pontos iniciais de cada uma das órbitas obtidas (digamos metade dos pontos). O primeiro e o segundo passos servem para se detectar todos os pontos atratores, enquanto que o último serve para descartar os pontos (transientes) inicias das órbitas, mantendo-se assim, somente aqueles pontos que se confundem com os estados assintóticos, de tão numericamente próximos que estão. A palavra 'grande' é usada (três vezes) o que de fato não confere ao procedimento um caráter muito preciso. Realmente não há uma maneira geral e rigorosa de se estabelecer o quão grande têm que ser os 'grandes' mencionados acima. Decidir isso requer um procedimento do tipo tentativa-e-erro no sentido de que, a partir de um determinado valor, qualquer noção de 'grande' serve, isto é, conduz ao mesmo resultado gráfico. Finalmente, observamos que, no caso do mapa logístico, o passo (a) acima pode ser relaxado e ser executado a partir de uma única condição inicial. Entretanto, o processo como descrito acima é mais geral e aplicável a qualquer sistema.

Para $1 < \mu \leq 4$, intervalo $I = [0,\ 1]$ é um conjunto invariante do mapa logístico f_μ, isto é, $f_\mu(x) \in I$ para todo $x \in I$. Para $\mu > 4$, I deixa de ser invariante de

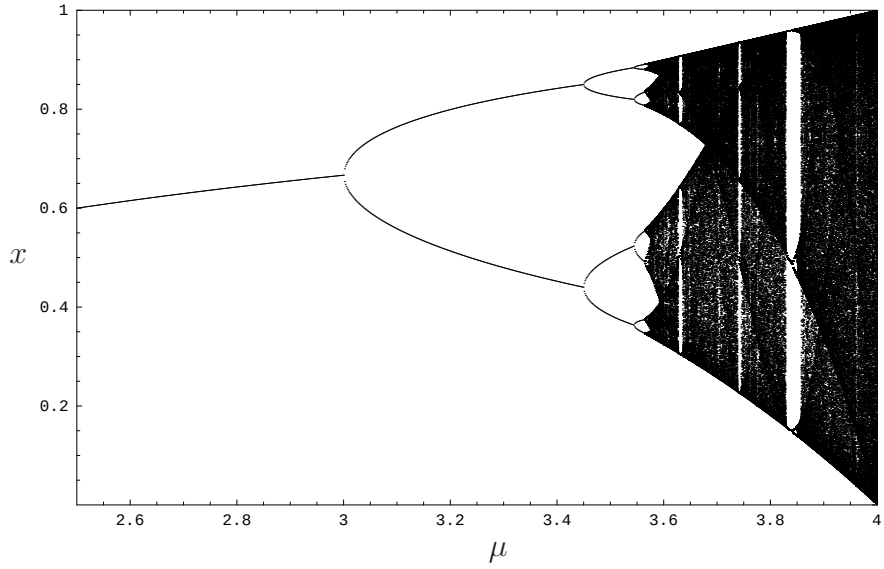

FIGURA 3.2 – Diagrama de bifurcações do mapa logístico para $2.5 < \mu < 4$ (eixo horizontal). No eixo vertical estão representados os pontos fixos/periódicos. Note a seqüência de duplicações de períodos até que os (vários) regimes caóticos se estabeleçam.

f_μ, mas continua sendo caótico num subconjunto $\Lambda \subset I$, este sim invariante de f_μ. É possível mostrar que Λ é de fato um conjunto de Cantor, que sabidamente tem geometria fractal, ou seja, é invariante por mudanças de escala. Este conjunto entretanto não possui órbitas estáveis. Portanto, Λ é um repulsor pois as órbitas que começam próximas de seus pontos divergem para $-\infty$. Por esta razão, não é possível qualquer explicitação de Λ por procedimentos computacionais que envolvam operações com ponto flutuante, pois os erros de arredondamento ocasionam divergências.

Como dito anteriormente, as características dinâmicas encontradas nos mapas logísticos são também encontradas em muitos outros sistemas dinâmicos, tanto discretos quanto contínuos. Por isto são muito importantes. Além disso, é neste tipo de mapa que se baseia um dos métodos de criptografia caótica abordados neste trabalho.

Conforme citado anteriormente, um sistema dinâmico $\mathcal{S}$ definido pelo mapa deve possuir dinâmica complexa para garantir a segurança do método de criptografia que o utiliza. Por isto, é conveniente que $\mathcal{S}$ seja caótico. Uma das maneiras de se quantificar a

caoticidade de um sistema dinâmico é através de seu *expoente de Lyapunov*. Por causa da transitividade topológica, é possível se definir o expoente de Lyapunov $\mathcal{L}$ de $\mathcal{S}$ a partir de qualquer uma de suas órbitas típicas. Com efeito, se $\gamma = (x_0, x_1, x_2, \ldots)$ é uma órbita típica de $\mathcal{S}$, então

$$\mathcal{L}(\gamma) := \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} \log_b |f'(x_n)|, \tag{3.2}$$

onde $x_n = f^n(x_0)$. Os expoentes de Lyapunov locais de f em x_n são calculados pela equação:

$$l(x_n) := \log_b |f'(x_n)| \tag{3.3}$$

Tais expoentes medem a separação (se $l(x_n) > 0$) ou aproximação (se $l(x_n) < 0$) local das órbitas próximas a γ em torno de x_n, para cada instante n. A Equação (3.2) diz que $\mathcal{L}(x_0)$ é a média aritmética dos expoentes de Lyapunov locais avaliados nos pontos de γ. Como uma órbita típica visita todas as partes do domínio de $\mathcal{S}$, tem-se que $\mathcal{L}(x_0)$ é uma medida global da sensibilidade de $\mathcal{S}$ às condições iniciais. Por isto, muitas vezes $\mathcal{L} = \mathcal{L}(x_0)$ é denominado expoente de Lyapunov *global* de $\mathcal{S}$ [45].

Em (3.2), a base b do logaritmo pode ser qualquer número positivo maior que 1, dizendo respeito à unidade em que a separação esta sendo medida. Os valores mais utilizados são $b = 2$, $b = 10$ e $b = e$ (número de Napier). No primeiro caso, o expoente é medido em bits por unidade de tempo. Note que γ é repulsora se $\mathcal{L}(\gamma) > 0$ e atratora se $\mathcal{L}(\gamma) < 0$. A Figura 3.3 mostra o expoente de Lyapunov do mapa logístico (3.1), para $3.5 \leq \mu \leq 4$, junto com seu diagrama de bifurcação. Nela pode-se notar claramente que, nas janelas de periodicidade, tem-se $\mathcal{L} < 0$.

Os expoente de Lyapunov são importantes em termos de criptografia, na medida que quantificam a taxa de produção de incerteza do mapa caótico no tempo. Como regra geral, quanto mais caótico for o sistema, mais segura será a criptografia nele baseado. O algoritmo de De Oliveira e Sobottka [7] utiliza os expoentes de Lyapunov para aumentar a segurança da cifração de dados.

O expoente de Lyapunov, juntamente com a propriedade de ergodicidade são a base da segurança do algoritmo de De Oliveira e Sobottka. Por ser ergódico, o algoritmo garante que as chances de ocorrer visitação não uniforme dos sítios de seu

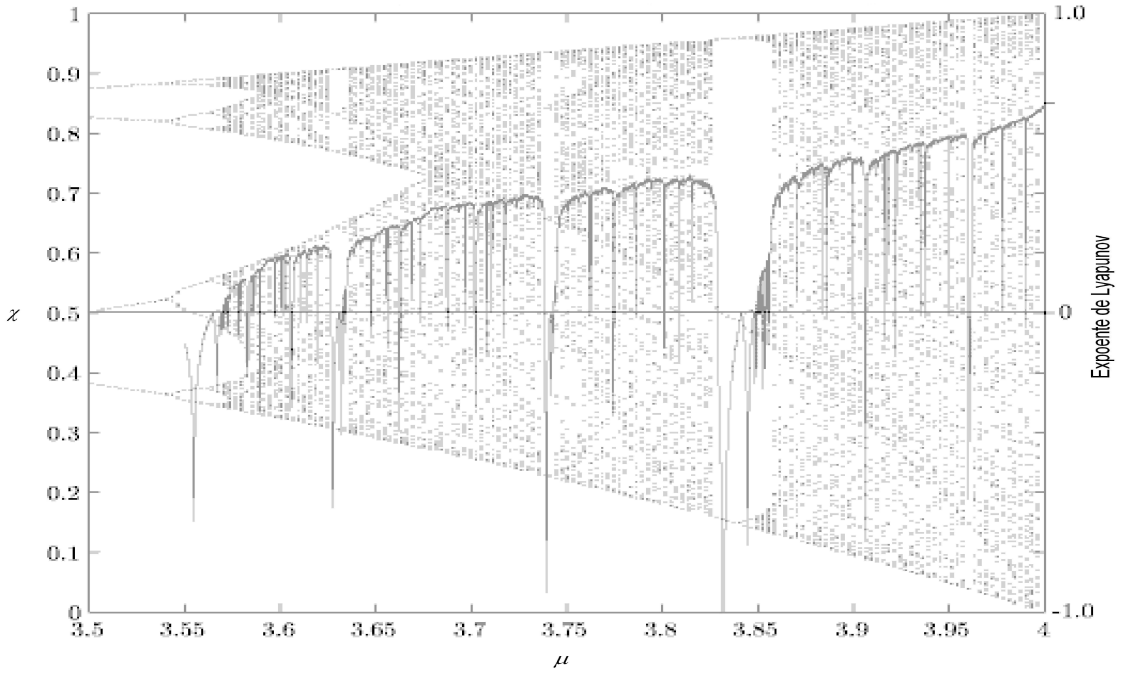

FIGURA 3.3 – Expoentes de Lyapunov e diagrama de bifurcação do mapa logístico plotados juntos para $3.5 \leq \mu \leq 4$.

mapa caótico tende a zero. Para se ter uma idéia de quão pequena é a chance de gerar um mapa com visitação não uniforme, podemos comparar isso a chance de um gerador pseudo-aleatório sortear um milhão de vezes seguidas o mesmo número. A chance existe, porém tende a zero. A Teoria Ergódica usa o termo técnico "quase"para referenciar-se a essa propriedade. Conforme o sentido dado, esse termo técnico pode definir tanto eventos que tendem a nunca ocorrer, quanto eventos que tendem a ocorrer sempre.

3.2 O Cifrador de Baptista

No que se segue, $P = (P_1, P_2, \ldots, P_L)$ é o texto plano e $C = (C_1, C_2, \ldots, C_L)$ a sua versão cifrada. $\mathcal{A}$ denota o alfabeto utilizado com $\#\mathcal{A}$ símbolos.

Em seu modelo, Baptista [6] escolhe um subintervalo $X = [x_{min},\ x_{max}] \subset [0,\ 1]$ do domínio de um mapa Logístico (3.1) caótico com $3 < \mu < 4$. O intervalo X é então

dividido em $\#\mathcal{A}$ sítios de tamanho $e = (\ x_{max} - x_{min})/\#\mathcal{A}$. A cada um dos sítios I_k é associado um símbolo do alfabeto $\mathcal{A}$, definindo um mapeamento dos símbolos de $\mathcal{A}$ aos subintervalos I_k. A Figura 3.4 mostra o esboço do mapeamento adotado por Baptista.

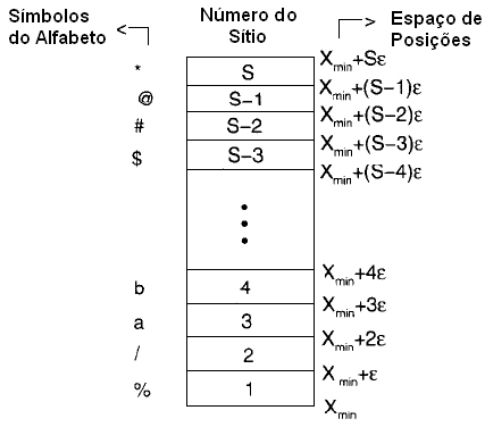

FIGURA 3.4 – Mapeamento dos símbolos do alfabeto dentro do intervalo do Mapa Caótico definido por Baptista [6].

A chave de cifração é dada por um valor $x_0 \in (0,\ 1)$, a partir do qual $f_\mu = \mu x(1-x)$ é iterada, dando origem à órbita $\gamma = (x_0, x_1, x_2, \ldots)$ com $x_n = f_\mu^n(x_0)$, $n = 0, 1, 2, \ldots$. A cifração se baseia na órbita γ associada a uma condição inicial x_0 (a chave). De modo breve, a versão cifrada de cada símbolo P_j constituinte de um texto plano $P = (P_1, P_2, \ldots, P_L)$ é dada pelo número de iterações de f_μ necessárias para que a órbita γ visite o sítio associado a P_j. Por cifrar um símbolo de cada vez, o sistema de Baptista é classificado com um sistema de cifração de fluxo (*stream cipher*).

A viabilidade do processo de cifração se baseia na transitividade dos mapas caóticos, que garante que cada um dos sítios será visitado depois de um certo número de iterações de f_μ. Portanto, o parâmetro μ deve ser muito bem escolhido, de modo a garantir a caoticidade de f_μ. Por exemplo, valores de $\mu < 3$ não podem ser aceitos pois ai f_μ não é caótico. Além disso, valores de $\mu > 3$ onde o expoente de Lyapunov não é positivo, também não devem ser aceitos. Como pode se ver na Figura 3.3, mesmo escolhendo valores $3 \leq \mu \leq 4$, não se pode garantir que f_μ é caótico. Isto se constitui num importante inconveniente do método de Baptista.

Na versão cifrada $C = (C_1, C_2, \ldots, C_L)$ de $P = (P_1, P_2, \ldots, P_L)$, cada componente C_j é o número de iterações de f_μ, a partir do ponto anterior $f^{C_1+C_2+\ldots+C_{j-1}}(x_0)$, necessárias para a órbita γ visitar um sítio correspondente ao

símbolo P_j que se deseja cifrar. Portanto, para cada símbolo do texto plano, um novo valor de iterações é gerado no texto cifrado [6].

Notasse que a contagem do número de iterações para cifrar P_j se inicia sempre do ponto usado para cifrar o símbolo anterior P_{j-1}. Portanto, a cifração de P_j depende da cifração de todos os símbolos planos anteriores, de P_1 a P_{j-1}. Esta é uma característica desejável em códigos de cifração pois causa a difusão das características estatísticas de P em relação à sua versão cifrada C [1].

Como exemplo hipotético de funcionamento, supondo-se que, é aplicado o mapa f_μ, a partir de um certo x_0 (chave), sobre o texto plano "did", isto é, $P = (d, i, d)$. Desta cifração obtém-se o texto cifrado $C = (3, 4, 2)$, conforme representado na Figura 3.5. Isto significa que partindo do sítio x_0, o mapa foi iterado três vezes até alcançar um sítio correspondente ao símbolo plano "d". O valor três se torna então o primeiro caractere do texto cifrado. A partir deste sítio, foram iteradas mais quatro vezes para alcançar o sítio do símbolo plano "i". O número quatro, é o segundo caractere do texto cifrado. Por fim, a partir do sítio "i", são necessárias mais duas iterações para alcançar novamente um outro sítio "d". O número dois compõe então o último caractere do texto cifrado. A decifração se faz de maneira óbvia: calcula-se as iteradas $x_1 = f_\mu^3(x_0)$, $x_2 = f_\mu^4(x_1)$ e $x_3 = f_\mu^2(x_2)$, e verifica-se qual dos símbolos de $\mathcal{A}$ está associado aos sítios I_k aos quais pertencem cada um dos valores x_1, x_2 e x_3.

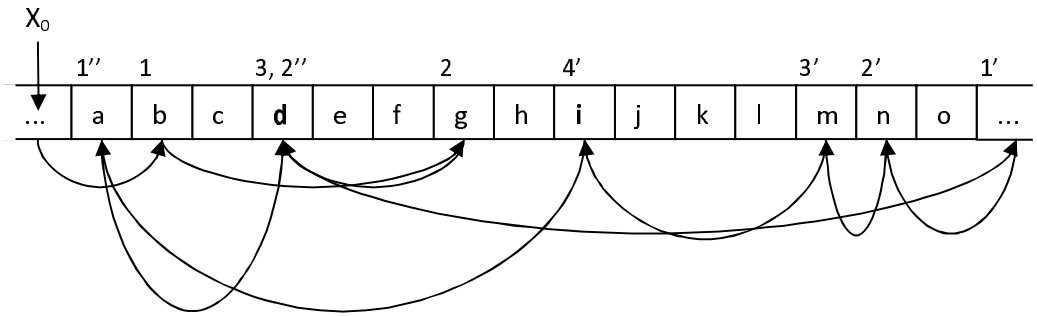

FIGURA 3.5 – Representação hipotética da cifração e decifração do texto plano $P = (d, i, d)$, utilizando um mapa caótico f_μ, com condição inicial x_0. O mapa f_μ está dividido em sítios relacionados aos símbolos do alfabeto utilizado. Os valores sobre o mapa correspondem as iterações para cifrar $C_1 \rightarrow (1, 2, 3), C_2 \rightarrow (1', 2', 3', 4')$ e $C_3 \rightarrow (1'', 2'')$.

É importante notar que, em geral, f_μ necessita de um número mínimo de iterações até que, a partir de x_0, exibam comportamento caótico. Antes disso, condições iniciais parecidas podem gerar os mesmos resultados em termos de cifração. Portanto, o algoritmo de Baptista define como N_0 o número mínimo de iterações sobre a função logística antes de iniciar a cifração do primeiro símbolo.

Como exemplo adicional, citaremos o exemplo dado por Baptista [6], onde se tem a cifração de "hi", $P = (h, i)$, com os seguintes parâmetros: $N_0 = 250$; $\mu = 3.8$ e $x_0 = 0,232323000000000$ e a letra "h", associada ao sítio 104 que represento pelo intervalo [0,44140625000000, 0,44375000000000). A letra "i"associada ao sítio 105 que representa o intervalo [0,44375000000000, 0,44609375000000). Uma possível versão cifrada de P é $C = (1713, 364)$. O primeiro valor C_1 corresponde a um número de iterações, necessárias para x_n cair no sítio 104. A partir de x_0, para o segundo valor C_2 inicia-se novamente a contagem de iterações a partir da posição de x_{1713} no sítio 104 até que x_n atinja o sítio 105.

Críticas ao Modelo de Baptista

Segundo Li e Lian [47], as características de sensibilidade as condições iniciais e parâmetros de controle e variabilidade podem ser ligadas aos princípios de difusão e confusão propostos por Shannon [13] e que, atualmente, são características necessárias a um bom sistema criptográfico.

O algoritmo proposto por Baptista sofreu críticas quanto a falhas de segurança. Em seu artigo, Álvarez et. al. [8] tratam de algumas dessas potenciais falhas, algumas utilizando métodos clássicos como Ataque por Texto Plano Escolhido, com os quais é possível descobrir a chave x_0 com facilidade. Outra forma de ataque proposta por Álvarez et. al., é através da análise da entropia dos dados gerados. A Tabela 3.1 dá um idéia do nível de entropia máxima e o atingido pelo cifrador de Baptista para textos planos contendo 1000 bytes.

Nesse estudo nota-se que para textos planos com alfabeto grande, a relação entre a entropia ideal e a real se aproximam. Porém, ainda assim existe uma diferença na distribuição de símbolos que pode ser explorada.

Li et al. [9], criticam a visitação não uniforme dos sítios na dinâmica de f_μ, o que é considerado um facilitador para ataques que analisam a entropia do sistema (ataque

TABELA 3.1 – Entropia ideal e obtida com o cifrador de Baptista para textos planos cifrados com alfabetos de entrada de diferentes tamanhos [8].

Tamanho do alfabeto	2	4	8	16	32	64	128	256
Entropia do cifrador de Baptista	0.71	1.58	2.75	3.84	4.87	5.89	6.88	7.86
Entropia ideal	1	2	3	4	5	6	7	8

diferencial e ataque linear). Estes mesmos autores fazem questionamentos quanto a algumas características indesejadas em cifradores como o tamanho do texto cifrado ser maior que o do texto plano, o que segundo Schneier, é um fator limitante para algumas aplicações [10].

Devido a estes e outros problemas como os citados em [48, 49, 9], algumas propostas de modificação do algoritmo de Baptista surgiram [48]. Além disso, novos algoritmos foram propostos como em [50, 18]. A maioria destes sistemas sofrem de algum problema quanto ao desempenho, segurança e, principalmente, degradação dinâmica de métodos de cifração caóticos, aplicados a sistemas digitais [49, 48]. Este problema ocorre quando o cifrador é dependente de máquina, como é o caso do cifrador de Baptista. Isto é, conforme o número de iterações aumenta, pode existir a perda de precisão no cálculo do mapa. Como conseqüência o sistema pode divergir ou, convergir para um ou mais pontos fixos.

3.3 O Cifrador de De Oliveira e Sobottka

Recentemente, De Oliveira e Sobottka [7] propuseram um cifrador baseado no modelo definido por Baptista. Este sistema será aqui referido como Cifrador Caótico, e denotado por CC. O sistema baseia-se na família de mapas $\mathbb{H}_p$ com parâmetro $p \in \mathbb{Q}_*^x$, racional positivo, definidos como

$$\mathbb{H}_p : [0,\ 10^p) \to [0,\ 10^p) \quad \text{com} \quad \mathbb{H}_p(x) = r_p^{-1} \circ \mathbb{G} \circ r_p(x), \tag{3.4}$$

onde $\mathbb{G}(x) = 10x (\mathrm{mod}\ 10)$ e $r_p(x) = \sqrt[p]{x}$. As funções r_p e r_p^{-1} definem uma equivalência entre $\mathbb{H}_p$ e $\mathbb{G}$.

Como $\mathbb{G}$ é sabidamente caótica, o mapa $\mathbb{H}_p$ é caótico para todo $p \in \mathbb{Q}_+^*$. Portanto, ao contrário do algoritmo de Baptista, o parâmetro $p \in \mathbb{Q}_+^*$ poderia ser adotado como parte da chave [26].

Com este modelo, De Oliveira e Sobottka se propõem a resolver três problemas encontrados no algoritmo de Baptista. O primeiro refere-se à limitação do espaço de escolha do parâmetro μ da família de funções logísticas, que no sistema de Baptista é bastante limitado. Aqui, este problema não existe pois $\mathbb{H}_p$ é caótico para todo $p \in \mathbb{Q}_+^*$. Outro problema é quanto à não homogeneidade na probabilidade em que os sítios são visitados pelas órbitas. O terceiro se deve ao fato de que o uso do sistema de Baptista é dependente da máquina usada, pois é sensível aos métodos de arredondamento, enquanto que no CC não há degradação do caos.

Basicamente o algoritmo CC deve, inicialmente, escolher os parâmetro p e d, depois, divide-se o domínio $[0, 10^p)$ em 10^d subintervalos semi-abertos à direita I_k, e define-se uma associação uniforme dos símbolos de $\mathcal{A}$ para os sítios I_k. Então, segue-se a idéia original de Baptista [6], que consiste em iterar $\mathbb{H}_p$ sucessivamente a partir da chave $x_0 \in [0,\ 10^p)$, o número necessário de vezes para a órbita $\gamma = (x_0, x_1, \ldots)$ atinja consecutivamente os sítios correspondente a cada símbolo P_j do texto plano $P = (P_1, P_2, \ldots, P_L)$. A mensagem cifrada C é então a seqüência composta de número C_j de iterações de $\mathbb{H}_p$, necessários para a órbita $\gamma = (x_0, x_1, \ldots, x_n)$, dada por $(x_n) = (\mathbb{H}_p^n(x_0))$, visitar os sítios associados a cada um dos consecutivos símbolos de P. Mais precisamente $C = (C_1, C_2, \ldots, C_L)$ é a seqüência natural de números C_j tais que $\mathbb{H}_p^{C_j}(x_{C_{j-1}})$, cai, pela primeira vez, no intervalo I_k associado a P_j. A decifração de C é feita usando a mesma chave, de maneira obvia: lendo cada C_j como o símbolo P_j associado ao sub-intervalo I_k ao qual pertence $\mathbb{H}_p^{\sum_{i=1}^{j} C_i}(x_0)$.

Para cada $p \in \mathbb{Q}_+$, a medida λ_p associada a $\mathbb{H}_p$ é dada por $\lambda_p\Big((a,b)\Big) = \frac{\sqrt[p]{b} - \sqrt[p]{a}}{10}$, para todo $(a,b) \subseteq [0,\ 10^p)$. Segundo De Oliveira e Sobottka, se definido que $I_k = \left[\frac{k^p}{10^{p(d-1)}}, \frac{(k+1)^p}{10^{p(d-1)}}\right)$, $k \in K_d$, isto significa que a freqüência de visitação de cada sítio I_p é a mesma e igual a $1/10^p$. Portanto o cifrador é isento de tendências quanto à cifragens de sílabas do alfabeto $\mathcal{A}$ utilizado [26].

Ajuste do Grau de Segurança

O grau de confusão aplicado ao texto cifrado pode ser regulado através do expoente de Lyapunov $\mathcal{L}_p$, dado por (3.2) e dos expoentes de Lyapunov locais $l_p(x_n)$, dados por (3.3), pois eles quantificam diretamente a sensibilidade de $\mathbb{H}_p$ às condições iniciais, e portanto, a segurança do sistema. A Figura 3.6 mostra o comportamento dos expoentes de Lyapunov locais de $\mathbb{H}_p$ em cada ponto de seu domínio $[0,\ 10^p)$, para $p = 2$ e $p = 4$. Pode-se notar que sempre existem 10 regiões (intervalos) R_r, $r = 0, 1, \ldots, 9$, bem definidas e similares onde o expoente de Lyapunov local varia de modo estritamente crescente.

Nas discussões, abaixo adota-se a seguinte Definição de segurança, conforme De Oliveira e Sobottka [7].

Definição: *Diz-se que o sistema criptográfico é ϵ-seguro contra ataque por força bruta quando, para quase todo par de condições iniciais x_0 e x_0' tais que $|x_0 - x_0'| \geq \epsilon$, tem-se que as órbitas correspondentes assumem uma distância superior a 0.9×10^p, antes que o primeiro símbolo P_1 de P seja cifrado. Em outras palavras, se $|x_0 - x_0'| \geq \epsilon$, tem-se uma incerteza maior que noventa por cento do tamanho total do domínio $\mathbb{H}_p$, antes de decifrar P_1.*

Para prevenir a quebra de sigilo, o CC usa a sensibilidade às condições iniciais de $\mathbb{H}_p$ para garantir um grande grau de incerteza a partir de erros na chave. Para isto, escolhe-se um valor $l_0 < 1$ como limite para o mínimo de sensibilidade às condições nos pontos correspondentes aos símbolos C_j da versão cifrada de P_j. Assim, para a ϵ-segurança, é necessário que somente sejam aceitos como versão cifrada de P_j valores de C_j tais que, nos pontos de $[0,\ 10^p)$ correspondentes, se tenha expoente de Lyapunov local maior do que ou igual a l_0.

Com a exigência acima, cada uma das regiões R_r, mencionadas, fica dividida em duas regiões disjuntas $R_{r,p,l_0} = B_{r,p,l_0} \dot{\cup} C_{r,p,l_0}$ onde $l_p(x) < l_0$ em B_{r,p,l_0} e $l_p(x) \geq l_0$ em C_{r,p,l_0}. Denotando $B_{r,p,l_0} = [r^p,\ a_{r,p,l_0})$ e $C_{r,p,l_0} = [a_{r,p,l_0},\ (r+1)^p)$, pode-se mostrar

que o ponto divisor a_{r,p,l_0} é dado pela seguinte fórmula [7]:

$$a_{r,p,l_0} = \left[\frac{r}{1 - 10^{(p-l_0)/(1-p)}}\right]^p \quad r = 0, 1, 2, \ldots, 9. \tag{3.5}$$

Com a fórmula acima, pode-se comparar diretamente o ponto encontrado x em $[0,\ 10^p)$ com a_{r,p,l_0}, verificando se ele atende à condição do expoente de Lyapunov local mínimo, e se o sítio correspondente ao valor C_j pode ser aceito como versão cifrada de P_j.

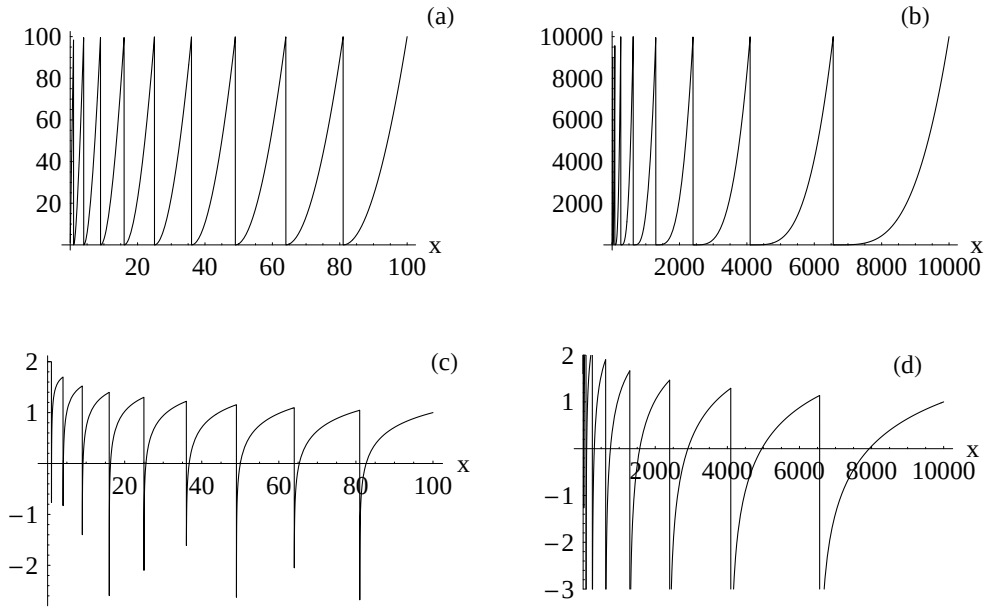

FIGURA 3.6 – Gráfico de $\mathbb{H}_p$ e o respectivo expoente de Lyapunov local para p=2 ((a) e (c)) e p=4 ((b) e (d)). Os valores mostrados de l_p são limitados ao intervalo [-3,2] para facilitar a comparação [7].

Capítulo 4

Algoritmo Proposto

Neste capítulo, será apresentado o algoritmo proposto (CC), com o objetivo de torna-lo competitivo quando comparado aos cifradores tradicionalmente utilizados atualmente, tanto em termos de velocidade, quanto em termos de segurança. As principais diferenças em relação ao algoritmo de De Oliveira e Sobottka são: melhor alocação da órbita (Seção 4.1), melhor formatação do arquivo cifrado (4.3) e maior segurança com o modo estatístico (Seção 5.2).

4.1 Modelo criptográfico do CC

A implementação se baseia na relação entre a seqüencia de visitação dos sítios I_k (ver Seção 3.3) percorridos por uma órbita $\gamma = (x_0, x_1, \ldots)$ e a seqüencia dos dígitos da representação decimal de $\sqrt[p]{x_0}$, dada pela seguinte Proposição [7]:

Proposição: *O d-bloco de dígitos $\overline{s_{n+1} \ldots s_{n+d}}$ da representação decimal de $\sqrt[p]{x_0}$ determina o subintervalo (sítio) I_k ao qual $x_n = \mathbb{H}_p^n(x_0)$ pertence, isto é, $x_n \in I_k$ com $k = \overline{s_{n+1} \ldots s_{n+d}}$.*

Na Proposição acima, o mapa caótico dado por $\mathbb{H}_p(x_0)$ é iterado n vezes a partir da condição inicial x_0 pertencente ao intervalo $[0, 10^p)$. Enquanto que, x_n é o valor da n-ésima iterada do mapa caótico, seu valor esta dentro do intervalo definido para o sítio I_k. Por fim, d é o tamanho, em dígitos, de cada sítio.

Como mencionado na Seção 3.3, esta Proposição implica que, iterar $\mathbb{H}_p$ a partir de x_0 sucessivamente, equivale a percorrer progressivamente os dígitos de $\sqrt[p]{x_0}$ em janelas de tamanho d, para encontrar os d-blocos de dígitos associados aos símbolos do alfabeto $\mathcal{A}$ que compõem o texto plano. Como existem métodos de cálculo de $\sqrt[p]{x_0}$ para qualquer precisão desejada, tem-se uma implementação independente da máquina utilizada.

O funcionamento do cifrador consiste em calcular os dígitos de $\sqrt[p]{x_0}$ e agrupá-los em d-blocos. Cada d-bloco esta associado a um símbolo do alfabeto. Iniciando-se sempre a partir do último d-bloco lido, procura-se o próximo d-bloco associado ao símbolo que se deseja cifrar. O símbolo cifrado corresponde ao número de procuras (iterações) feitas até encontrar o d-bloco relacionado com o símbolo que se deseja cifrar. A Figura 4.1 mostra com são agrupados os dígitos para formar d-blocos e como é feita a iteração sobre eles. A Figura mostra quatro iterações sobre uma órbita não periódica, dividida em d-blocos de tamanho d=3. Nota-se que, na busca, a parte inteira de $\sqrt[p]{x_0}$ é sempre ignorada.

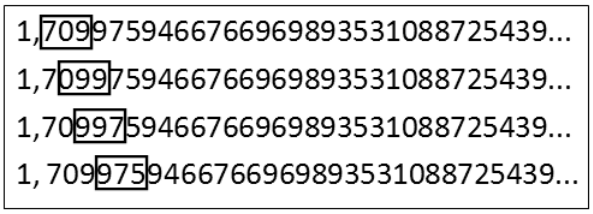

FIGURA 4.1 – Exemplo de varredura dos dígitos de $\sqrt[p]{x_0}$, com $p = 3$ e $x_0 = 5$, para $d = 3$.

A descrição apresentada acima define o método de cifração apresentado em [26] e [7]. Essa forma de cifração traz vantagens sobre a maioria dos outros cifradores caóticos, como a visitação uniforme dos sítios I_k. Em especial, essa forma de cifrar dados resolve o problema da dependência de máquina, a qual é um dos principais problemas do cifrador de Baptista e outros similares. De fato, é sabido que as máquinas têm precisão finita e isso afeta as órbitas dos mapas caóticos devido a sensibilidade desses sistemas às condições iniciais. Portanto, a maioria dos cifradores caóticos, exige que a máquina de cifração tenha a mesma precisão da máquina de decifração (8, 32, 64 bits, ...), caso contrário, há medida que os mapas são iterados, a órbita de decifração vai divergir da órbita gerada na cifração, impedindo a decifração da mensagem de forma

correta, mesmo conhecendo-se a chave. Alguns cifradores propõem contra-medidas para evitar esse inconveniente, como em [48, 50]. No CC esse problema não existe pois o uso da operação de raiz para extrair o mapa caótico tem cálculo exato.

Além das vantagens já existentes no cifrador, novas características foram introduzidas no CC originalmente proposto, tornando-o ainda mais seguro e eficiente.

Foi adicionada ao CC uma tabela de endereços Γ. Essa tabela é montada com as posições dos d-blocos associados à cada símbolo do alfabeto $\mathcal{A}$, na representação decimal de $\sqrt[p]{x_0}$. Mais precisamente, cada linha está associada a um elemento de $\mathcal{A}$, enquanto as colunas contém os d-blocos de cada elemento. Note que o símbolo P_{j+1} está associado a posição $\Gamma_{l_{P_{j+1}},c}$, sendo $l_{P_{j+1}}$ a linha referente ao símbolo P_{j+1} e a coluna c, é a c-ésima vez que o símbolo P_{j+1} aparece na representação decimal de $\sqrt[p]{x_0}$.

A Tabela 4.1 mostra um exemplo de Γ, com $\mathcal{A} = \{a, b, c\}$. Neste exemplo, as posições $10, 106, 215, ..., 2567$ correspondem a sítios do símbolo a. As posições $29, 198, 327, ..., 2608$ correspondem ao símbolo b e por fim, $5, 77, 284, ..., 2670$ correspondem ao símbolo c.

TABELA 4.1 – Exemplo de uso da tabela Γ.

Alfabeto	Coluna 1	Coluna 2	Coluna 3	...	Coluna n
a	**10**	**106**	215	...	2567
b	29	198	**327**	...	2608
c	5	77	**284**	...	2670

Dado que, em geral, Γ é montada apenas uma vez, o processamento principal de cifração/decifração constitui uma seqüencia de procuras e salvamentos. O uso de Γ é basicamente como segue. Primeiro, encontra-se a linha correspondente ao símbolo corrente, isto é, o símbolo P_{j+1} do texto plano P. Então, na linha $l_{P_{j+1}}$ toma-se o primeiro endereço de d-bloco maior do que o anteriormente usado para cifrar (iteração). A versão cifrada é a diferença entre o endereço encontrado e o endereço imediatamente anterior. Por exemplo, de acordo com a Tabela 4.1, a versão cifrada do texto "aabc"é

$$a \mapsto \mathbf{10}, a \mapsto 106 - 10 = \mathbf{96}, b \mapsto 198 - 106 = \mathbf{92}, c \mapsto 284 - 198 = \mathbf{86}.$$

Nota-se que a opção 29 de b é ignorada, pois ela é menor que a posição do bloco (iteração) atual, que é 106. O mesmo ocorre com c, quando os índices 5 e 77 são ignorados. Existem várias formas de realizar a procura dos d-bloco dentro da tabela. A maneira mais eficiente, em termos de velocidade, é ajustando somente a linha da tabela conforme o símbolo a ser cifrado, isto é, a próxima busca inicia sempre na mesma coluna que foi encontrado o d-bloco anterior. Sendo $P = \{..., P_{j-1}, P_j, P_{j+1}, ...\}$ o conjunto de símbolos do texto plano P, a definição dessa regra é:

$$C_{j+1} := \min \left\{ \Gamma(l_{P_{j+1}}, c) - \Gamma(l_{P_j}, q) : \ \Gamma(l_{P_{j+1}}, c) - \Gamma(l_{P_j}, q) > 0, \ c \geq q \right\}. \quad (4.1)$$

Esta maneira de busca dos d-blocos não altera as propriedades do algoritmo originalmente definido em [26] e [7]. Algumas vezes, $\Gamma(l_{P_{j+1}}, q-1) - \Gamma(l_{P_j}, q) > 0$ e até $\Gamma(l_{P_{j+1}}, q-2) - \Gamma(l_{P_j}, q) > 0$ porém, conforme a Equação 4.1, estes d-blocos, potencialmente válidos, são ignorados pois $c \geq q$. Isso é feito para evitar testes constantes em colunas a esquerda da última coluna usada na cifração. Testes como esses geralmente resultam d-blocos inválidos. Evitando esses testes o CC ganha em velocidade, em troca de, eventualmente, descartar alguns poucos d-blocos válidos.

Embora percam-se alguns d-blocos, a inclusão de Γ e sua forma de utilização, tornam o cifrador muito mais eficiente em termos de velocidade e o algoritmo fica independente do tamanho do alfabeto, isto é, no modelo original o número de iterações para encontrar um sítio válido corresponde ao tamanho do alfabeto, ou seja, para um alfabeto de 256 símbolos, o número médio de testes até encontrar um sítio válido fica em torno de 256 buscas. Atualmente, o número de teste fica geralmente entre uma e três buscas, independente do alfabeto utilizado. O uso de Γ também torna o cifrador mais adaptável as exigências de processamento e uso de memória dos diversos ambientes de cifração. Além disso, Γ aumenta as possibilidades de paralelização de processamento do CC. Todos esses benefícios são aprofundados no Capítulo 5.

A Figura 4.2 representa a arquitetura geral do cifrador. Conforme ela mostra, a construção da tabela Γ é criada na inicialização do software. Para que não ocorram problemas de utilizar todos os índices da tabela, ela é renovada dinamicamente em duas situações.

A primeira, e mais crítica, quando no meio de uma cifração o algoritmo utiliza as

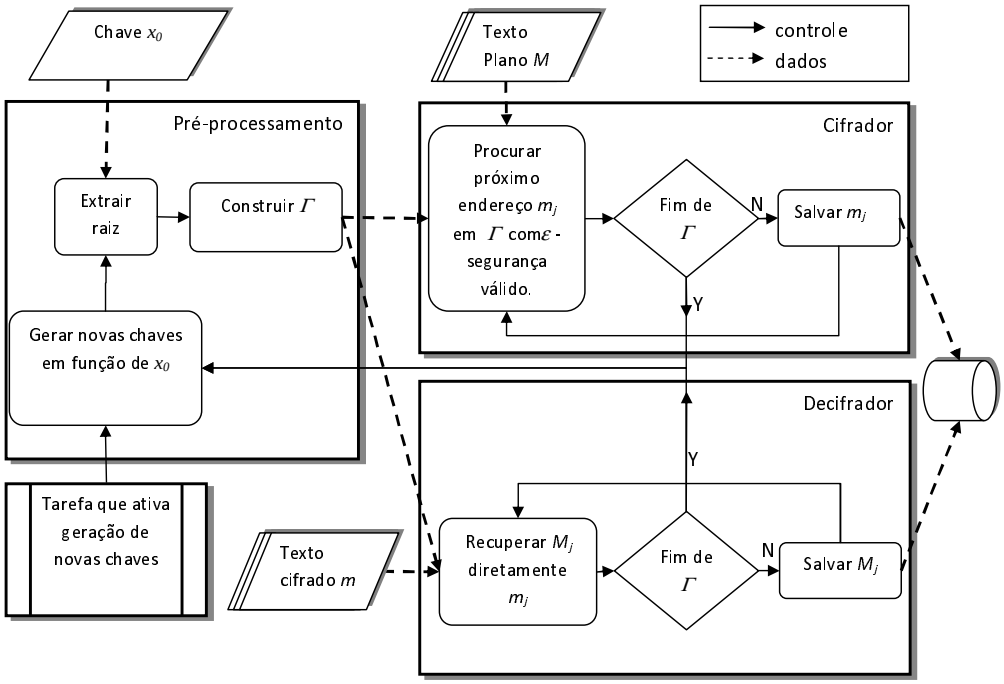

FIGURA 4.2 – Fluxograma geral do cifrador caótico.

últimas colunas de Γ. Neste caso, Γ é alimentada com novos valores para que a cifração prossiga. A situação mais comum ocorre quando o cifrador esta ocioso, neste caso ele aproveita este tempo para gerar um novo mapa caótico, e popular a tabela com uma nova órbita caótica. Este procedimento substitui apenas os valores das colunas já utilizadas. Desta forma, caso o cifrador utilize as últimas colunas de Γ, ele reinicia a procura a partir da sua primeira coluna, sem a necessidade de parar a cifração pois, os valores do início da tabela são novos e não vão gerar lacunas de segurança devido a possível repetição de valores antigos do mapa caótico. Conforme será visto no capítulo 5, o uso da mesma órbita de um mapa caótico para cifrar diversos textos não é indício de falha de segurança. A atualização dinâmica de Γ pode ser regulada para agir de tempos em tempos, ou após n textos cifrados com a mesma órbita.

Na decifração os d-blocos são armazenados em uma lista. Neste caso, Γ vira uma tabela de uma linha com cada d-bloco disposto em ordem de aparecimento. Para cada símbolo, o texto cifrado contém a quantidade de posições que devem ser iteradas a partir da última posição utilizada do domínio do mapa caótico. Desta forma, a decifração

ocorre de forma direta, isto é, o valor extraído do texto cifrado C_j é somado ao último índice utilizado de Γ, digamos C_{j-1}. O valor contido em $\Gamma(0, C_j + C_{j-1})$ é convertido diretamente no símbolo plano correspondente a C_j. Ou seja, a decifração é muito simples, utiliza apenas uma operação de soma, uma recuperação de memória e uma conversão direta $1 \rightarrow 1$.

Além da inclusão de Γ ao CC, outras melhorias foram propostas e implementadas. Uma delas se destaca por tornar o cifrador estatístico. Isso evita problemas de segurança, principalmente para ataques como o *One-time pad attack* e o ataque de Jakimoski-Kocarev. Esta modificação é descrita na Seção 5.2, no tópico referente ao *One-time pad attack*. As outras modificações e seus benefícios são analisados ao longo do Capítulo 5. A seguir é descrito em detalhes o processo completo de cifração e decifração do CC.

4.2 Processos de cifração e decifração

Pre-arranjos do sistema

Antes de iniciar a cifração/decifração, procede-se as seguintes operações, as quais devem ser feitas de comum acordo entre o transmissor da mensagem (quem vai realizar a cifração) e o receptor da mensagem (quem vai realizar a decifração):

1. Um número inteiro positivo d é escolhido. Considera-se todos blocos compostos por d dígitos, cada um dos dígitos podendo assumir um valor inteiro entre 0 (inclusive) e 9 (inclusive). Com essa construção, tem-se 10^d blocos de d dígitos chamados de d-*blocos*.

2. Cada símbolo de $\mathcal{A}$ é associado ao menos a um d-bloco. Por exemplo, se $d = 3$ e $\mathcal{A} = \{a, b, c, \ldots z\}$ é o alfabeto latino, temos $a = 000, b = 001, c = 002, \ldots, z = 025, a = 026, b = 027, c = 028, \ldots, z = 051, \ldots$. Neste caso, cada símbolo de $\mathcal{A}$ é associado a, aproximadamente, $10^d/\#\mathcal{A}$ d-blocos distintos com distribuição uniforme.

3. Calcula-se $\sqrt[p]{x_0}$ com $kL + d$ decimais, onde $k \geq \#\mathcal{A}$ é um inteiro positivo e

$L \geq 1$. Assim, tem-se $\sqrt[p]{x_0} = e_0, e_1e_2e_3 \ldots e_{kL+d}$, onde $e_i \in \{0, 1, 2, \ldots 9\}$ para $i \in \{1, 2, \ldots, kL + d$. A parte inteira e_0 é descartada. Isto define uma sucessão de d-blocos $(B_\ell)_{\ell=1,\ldots,kL}$. Mais especificamente para cada $\ell \in \{1, \ldots, kL\}$ tem-se que $B_\ell = e_\ell e_{\ell+1} \ldots e_{\ell+d-1}$ é um d-bloco. Por exemplo, se $d = 3$, $B_1 = e_1e_2e_3$, $B_2 = e_2e_3e_4$, $B_3 = e_3e_4e_5$, e assim por diante.

4. Dada a lista de *d-blocos*, o índice que identifica a localização de cada *d-blocos* é distribuídos em Γ na linhas correspondentes aos símbolos de $\mathcal{A}$ e com os *d-blocos* em ordem crescente.

Cifração

De posse da mensagem a ser cifrada $P = (P_1, P_2, \ldots P_L)$ e da tabela Γ, procedesse da seguinte maneira:

1. Para cifrar o primeiro símbolo P_1, procura-se em Γ na linha correspondente a P_1 e coluna inicial 0, o primeiro índice correspondente a P_1, o qual será a versão cifrada C_1 do primeiro símbolo plano P_1.

2. Para o segundo símbolo P_2, se inicia a busca em Γ a partir da linha correspondente a P_2 e a coluna 0. Caso o valor contido em $\Gamma(P_2, 0) < \Gamma(P_1, 0)$, se avança uma posição na coluna e se volta a testar a condição $\Gamma(P_2, 1) < \Gamma(P_1, 0)$, até que para uma certa coluna, $\Gamma(P_2, C) > \Gamma(P_1, 0)$. Assim, como símbolo cifrado para P_2 temos $C_2 = \Gamma(P_2, C) - \Gamma(P_1, 0)$.

3. O terceiro símbolo P_3, segue a mesma regra de cifração, iniciando a busca a partir de $\Gamma(P_3, C)$ até satisfazer a condição $\Gamma(P_3, C') > \Gamma(P_2, C)$. Sendo $C_3 = \Gamma(P_3, C') - \Gamma(P_2, C)$. Os outros símbolos, $(P_4, P_5, \ldots P_L)$, seguem a mesma regra.

Decifração

De posse da versão cifrada $C = (C_1, C_2, \ldots C_L)$ e da chave $K = (x_0,\ p)$, procede-se da seguinte maneira:

1. Calcula-se $\sqrt[p]{x_0}$ com $kL + d$ decimais, e descartamos sua parte inteira $e_0 \dots e_n$. Assim obtém-se a seqüência $e_1e_2, e_3 \dots e_{kL+d}$.

2. Lê-se o C_1-ésimo d-bloco da seqüência $e_1e_2, e_3 \dots e_{kL+d}$. O valor contido nesse bloco, indica diretamente o primeiro símbolo plano P_1.

3. Para se obter o segundo símbolo da mensagem P, lê-se o (C_1+C_2)-ésimo d-bloco da seqüência $e_1e_2, e_3 \dots e_{kL+d}$. O valor do d-bloco = $(C_1 + C_2)$ está associado ao símbolo P_2. De maneira análoga obtém-se os símbolos $P_3, \dots, P_L$. Em outras palavras, o símbolo P_j da mensagem $P = (P_1, P_2, \dots, P_L)$ é obtido lendo-se o símbolo do alfabeto $\mathcal{A}$ associado ao d-bloco $B_{C_1+C_2+\dots+C_j}$ da seqüência $e_1e_2, e_3 \dots e_{kL+d}$.

4.3 Formatação do arquivo cifrado

A exemplo dos outros cifradores caóticos da mesma família, os arquivos cifrados do CC são formados pelas quantidades de iterações necessárias para cifrar cada símbolo plano. Em alfabetos de 256 símbolos, o número médio de iterações na cifração será de 256. Acontece que, são necessários mais de 8 bits (1 byte), para representar valores maiores que 255. Isto significa que para muitos casos, serão necessários 2 bytes para representar cada byte de texto plano. A mesma regra vale para outros alfabetos compostos por mais, ou menos de 256 símbolos. Portanto, tanto no caso do CC quanto de outros cifradores caóticos, os textos cifrados terão o dobro do tamanho dos seus textos planos correspondentes.

No caso do CC foram implementadas duas maneiras de gerar os textos cifrados. A primeira, e mais obvia, é semelhante a outros cifradores caóticos da mesma família. Ela consiste em utilizar 2 bytes para representar cada símbolo cifrado. Nessa maneira, o símbolo cifrado segue a seguinte regra:

Sendo C_j = número de iterações = valor a ser cifrado e, resto(x,y) a operação que retorna o resto de divisão inteira de x por y,

$C_j < 256 \rightarrow C_{j1} = 0, C_{j2} = C_j$

$C_j \geq 256 \rightarrow C_{j1} = \lfloor C_j/256 \rfloor, C_{j2} = resto(C_j, 256)$

Portanto, a representação $C_j = (C_{j1}, C_{j2})$ ocupa dois bytes. Assim, temos $C = ((C_{11}, C_{12}), (C_{21}, C_{22}), ..., (C_{L1}, C_{L2}))$. Na decifração, o CC lê o texto cifrado em pares (C_{j1}, C_{j2}), recuperando-se P_j através da operação inversa $P_j = (C_{j1} \times 256) + C_{j2}$.

Na segunda forma, o CC mescla o uso de 1 e 3 bytes para representar cada símbolo cifrado. Ela é definida como:

$C_j < 256 \rightarrow C_{j1} = C_j$
$C_j \geq 256 \rightarrow C_{j1} = 0, C_{j2} = \lfloor C_j/256 \rfloor, C_{j3} = resto(C_j, 256)$

O símbolo 0 é usado como marcador de início de bloco com mais de 1 byte. O símbolo 0 pode ser utilizado como marcador, pois não existe qualquer situação em que a cifração de um símbolo corresponda a 0 iterações. Quando o símbolo 0 é encontrado, o decifrador lê os outros dois bytes do texto plano e decifra-os como um símbolo plano a exemplo do primeiro método, ou seja:

$P_j = C_{j1}$, se $C_j \neq 0$
$P_j = C_{j2} \times 256 + C_{j3}$, se $C_j = 0$

Aplicando o segundo método em alfabetos de 256 símbolos, na grande maioria das vezes, é usada a regra $P_j = C_{j1}$ para cifração, ou seja, um byte plano gera apenas um byte cifrado. Essa característica reduz o texto cifrado para até 1.6 vezes o tamanho do texto plano. Em todos os testes foi utilizada esta formatação de arquivos cifrados.

4.4 Gerenciamento da ϵ-segurança

Conforme a Definição 3.3, um cifrador definido pelo sistema dinâmico $(K_p, \mathbb{H}_p)$, onde $\mathbb{H}_p := r_p^{-1} \circ \mathbb{G} \circ r_p$ e $K_p = [0,\ 10^p)$, tem ϵ-segurança quando, para quase todo par de chaves $K =$ e $K' = (x_0', p)$ tal que $|K - K'| \geq \epsilon$, as órbitas correspondentes às condições iniciais K e K' se distanciam uma da outra mais do que $0,9 \times r_p^{-1}(10)$ antes de P_1 ser decifrado.

Devido à sua caoticidade, uma maneira de se controlar a ϵ-segurança do sistema é através dos expoentes de Lyapunov locais do sistema dinâmico $(K_p, \mathbb{H}_p)$. O expoente de Lyapunov local para $(K_p, \mathbb{H}_p)$ no ponto $y \in [0, 10^p)$ é definido como o valor $l_p(y)$ dado pela expressão:

$$l_p(y) := \log_b \left| \mathbb{H}'_p(y) \right|,$$

onde $\mathbb{H}'_p(y)$ representa a derivada de primeira ordem da função $\mathbb{H}_p$ no ponto y e $|\cdot|$ denota o valor absoluto, ver [7]. Existem duas formas de implementar a ϵ-segurança. A seguir são descritas essas formas.

Implementação de ϵ-segurança (forma I):

1. Escolhe-se um valor positivo l_0 que seja menor que o supremo sobre expoentes locais de Lyapunov de $(K_p, \mathbb{H}_p)$ e maior que $-\log \epsilon$;

2. Nessa forma, a ϵ-segurança consiste em somente aceitar como válidos para a cifra, valores $n \in \mathbb{N}$, onde $\mathbb{N}$ denota o conjunto de todos os números inteiros positivos, para os quais a órbita do sistema dinâmico $(K_p, \mathbb{H}_p)$, a partir da condição inicial x_0, tenha expoente de Lyapunov maior ou igual a l_0. Em outras palavras, para cifrar o primeiro símbolo P_1, procura-se o menor inteiro positivo C_1 tal que o d-bloco B_{C_1} na seqüência $e_1 e_2 e_3 \ldots e_{kL+d}$, está associado ao símbolo P_1 e $l_p(\mathbb{H}_p^{C_1}(x_0)) \geq l_0$. De maneira recursiva, para cifrar P_j, procura-se o menor inteiro positivo C_j tal que o d-bloco $B_{C_1+\ldots+C_j}$ na seqüência $e_1 e_2 e_3 \ldots e_{kL+d}$ está associado ao símbolo P_j e $l_p(\mathbb{H}_p^{C_1+\ldots+C_j}(x_0)) \geq l_0$.

Observação: O valor l_0 escolhido não necessita ser conhecido para que se realize a decifração de $C = (C_1, \ldots, C_L)$ e se recupere a mensagem $P = (P_1, \ldots, P_L)$.

Implementação de ϵ-segurança (forma II):

1. Na segunda forma de implementar a ϵ-segurança, escolhe-se um valor $r_0 > 0$ e descartar $e_0, e_1 \ldots e_{r_0-1}$ da representação decimal de $r_p(x_0)$, realizando a cifração buscando os d-blocos sobre a seqüência $e_{r_0} e_{r_0+1} \ldots e_{r_0+kL+d}$;

2. o valor r_0 é escolhido como número necessário de iterações do sistema dinâmico $(K_p, \mathbb{H}_p)$ sobre a condição inicial x_0 para que, uma distância ϵ de x_0, seja

expandida a uma distância maior ou igual que $0,9 \times r_p^{-1}(10)$, depois de r_0 iteradas de $\mathbb{H}_p$;

3. Usa-se o os expoentes locais de Lyapunov para o sistema dinâmico $(x_p, \mathbb{H}_p)$ para obter uma expressão implícita para r_0. De fato, desejamos que $0,9 \times r_p^{-1}(10) \leq \epsilon 10^{\sum_{n=1}^{r_0} l_p(\mathbb{H}_p^n(x_0))}$, o que implica em r_0 é determinado implicitamente pela expressão:

$$\sum_{n=1}^{r_0} l_p(\mathbb{H}_p^n(x_0)) \geq \log r_p^{-1}(10) + \log \frac{0,9}{\epsilon}.$$

Observação: o valor r_0 escolhido necessita ser conhecido para que se realize a decifração de $C = (C_1, \ldots, C_L)$ e se recupere a mensagem $P = (P_1, \ldots, P_L)$. Nesse caso, poderíamos considerá-lo parte da chave de cifração/decifração, a qual passaria a ser o termo $K_p = (x_0, p, r_0)$. O *software* concebido neste trabalho se ateve a atender a ϵ-segurança da forma I.

4.5 Considerações sobre o CC

Conforme apresentado neste Capítulo, o algoritmo CC conserva as características originais do algoritmo proposto por De Oliveira e Sobottka. Tais características são a distribuição uniforme de dígitos, independência de máquina e controle do grau de segurança. Além disso, o CC acrescenta novas características como aumento de desempenho de cifração e decifração conforme descrito na Seção 5.1. Outras características como a independência do tamanho do alfabeto, o paralelismo e o aumento da adaptabilidade a diferentes ambientes são abordados na Seção 5.3. Por fim, as novas características do CC trazem também ganhos de segurança. Os ganhos com segurança são abordados na Seção 5.2.

As novas características do CC em relação ao cifrador antigo são devido a três modificações substanciais. A inclusão da Tabela de Endereçamentos, tratada na Seção 4.1 traz vantagens em termos de velocidade, independência de alfabeto, melhor paralelismo e de certa maneira, adaptabilidade do cifrador a diversos ambientes. A inclusão de uma nova formatação do texto cifrado traz ganhos na adaptabilidade do cifrador a diversos ambientes. A terceira modificação refere-se á inclusão do

comportamento estatístico no CC. Esse comportamento afeta a segurança impedindo ataques que antes eram possíveis. A Seção 5.2 relata como funciona essa modificação e os ataques que ela impede. No próximo capítulo as novas características são abordadas em conjunto com as antigas, bem como os resultados de análises e testes de velocidade, segurança e aplicabilidade do CC.

Capítulo 5

Resultados

Neste Capítulo são apresentados os resultados dos experimentos de desempenho, segurança e a análise das características de software, realizados sobre o CC. Como base de comparação, o experimento de desempenho foi aplicado também aos cifradores AES, RC4 e Sosemanuk. Adicionalmente, são abordados aspectos de segurança e possíveis aplicações do CC. Para facilitar a avaliação desses aspectos o CC é constantemente comparado as características de segurança e aplicabilidade dos cifradores AES, RC4, Sosemanuk e outros algoritmos da família de cifradores caóticos.

Todos os testes foram realizados no mesmo ambiente: computador composto de processador AMD Semprom Mobile 3000, com 1.8GHz, 1280 Mbytes de memória RAM, 64 Kbytes de memória cache em L1 e 128 Kbytes de memória cache em L2. A velocidade do barramento de dados é 400 MHz. Foi utilizado o sistema operacional Linux Ubuntu em modo de segurança, sem qualquer interface gráfica e como usuário administrador. Essas medidas garantem o processamento dos testes sob total controle, sem que outros processos interfiram na qualidade das medidas.

5.1 Teste de desempenho

No teste de desempenho foi utilizado o Plano de Uma Única Variável, isto é, apenas uma variável será modificada enquanto as outras permanecem fixas. A variável alterada foi o tamanho dos textos planos. A variável observada foi o tempo de cifração e decifração. Foi medido o tempo de cifração/decifração para dezesseis tamanhos

diferentes de arquivos. O tamanho desses arquivos variou entre $512KB$ e $16MB$.

Devido à complexidade linear, essa gama de tamanhos permite prever o comportamento dos cifradores em função de diferentes tamanhos de textos planos.

Para cada tamanho de arquivo foi computada a média de cifração/decifração de dez amostras de textos planos não viesados. Textos não viesados são os que contém todos os símbolos do alfabeto distribuídos em quantidades iguais e de forma pseudo-aleatória. Ao contrário, os textos viesados carregam tendências. Isso significa que símbolos ou seqüencias de símbolos surgem com mais freqüência no texto. O alfabeto $\mathcal{A}$ utilizado neste experimento é constituído dos 10 símbolos numéricos ASCII, isto é $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Como o tamanho da chave influencia diretamente no desempenho dos algoritmos, todos os cifradores vão utilizar chaves de cifração de 128 bits, a exceção do CC, ao qual não se aplica essa característica, pois a chave é sempre um par (x_0, p), sobre o qual aplica-se a operação $\sqrt[p]{x_0}$ para extrair um número irracional. Optou-se também por manter o AES operando em modo CFB, o que o torna também um cifrador de fluxo.

A Figura 5.1 mostra o resultado desse experimento com o tempo de codificação e decodificação dos cifradores CC, AES, RC4 e Sosemanuk. Como pode ser observado, todos os algoritmos têm comportamento linear. Para cifração o RC4 é duas vezes mais rápido, e que o CC, o AES é quatro vezes mais rápido e o Sosemanuk é oito vezes mais rápido. Entretanto, para decifração o CC aumentou muito sua eficiência, sendo quase duas vezes mais rápido que o RC4, 1.3 vezes mais rápido que o AES, e 2.5 vezes mais lento que o Sosemanuk. O aumento de desempenho do CC para decifração se deve as características de implementação do algoritmo. Na decifração, o CC utiliza cada símbolo cifrado como índice da memória que contém o mapa caótico. Com isso, cada símbolo plano é recuperado diretamente da memória do computador. Portanto, seu único processamento é o acesso a memória que contém os valores correspondentes aos símbolos decifrados.

Quando comparado a outros cifradores caóticos, o CC é visivelmente superior. A iniciar pelo cifrador de De Oliveira e Sobottka, ao qual oCC é baseado. Neste caso o CC é 4 vezes mais rápido que o cifrador original na cifração e 10 vezes mais rápido na decifração. Outro exemplo disso é o cifrador caótico proposto em [51], que também apresenta testes de desempenho. Os autores citam, inclusive, o trabalho [26], o qual é uma das referências teórico-matemáticas do CC. Os testes de desempenho em [51]

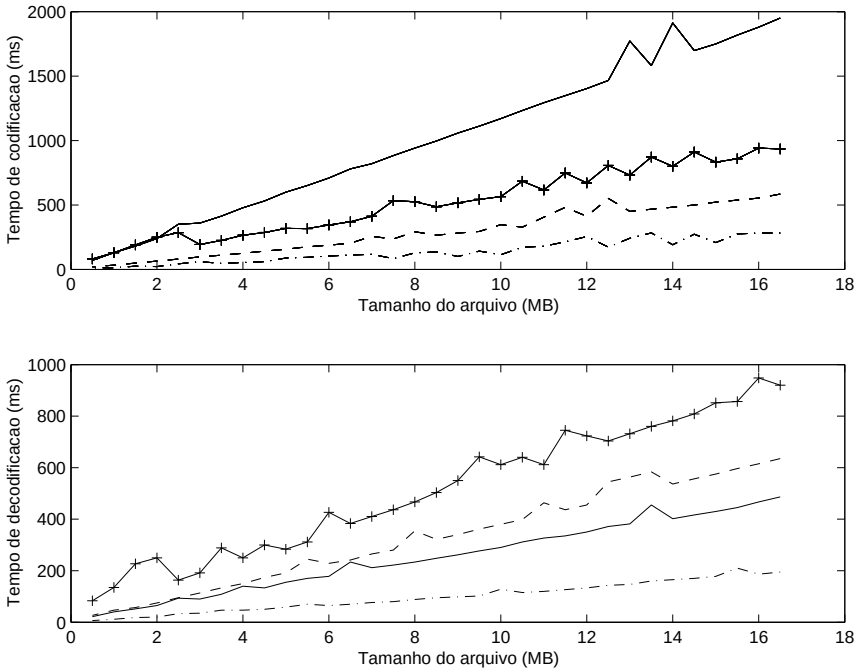

FIGURA 5.1 – Comparação de velocidade de cifração e decifração com variação no tamanho dos textos, entre CC (linha sólida), Sosemanuk (linha com tracejado e ponto), RC4 (linha tracejado e cruz), e o AES (linha tracejada).

foram realizados com uma máquina Pentium III com processador de 1GHz e 256MB de memória RAM.

Isto significa um poder de processamento inferior em 80% e 5 vezes menos memória, quando comparada com a máquina utilizada nos testes do CC. Foram utilizados textos planos de 258KB e 41KB. O desempenho obtido na cifração foi de 1,66 e 0,25 segundos respectivamente. Fazendo uma aproximação grosseira, a taxa de cifração do algoritmo caótico proposto no artigo ficou em 155,4 kbytes/segundo para o texto de 258KB e 164 kbytes/segundo para o texto de 41KB. Como média da cifração dos dois textos obtém-se 160 kbytes/segundo.

O CC tem comportamento linear, independente do tamanho do texto plano. Portanto, é possível estimar a taxa de cifração através do desempenho do CC para qualquer tamanho de texto pois, a taxa se mantém constante. Para cifrar um arquivo de 16MB, o CC gasta, em média, 2 segundos, portanto sua taxa de cifração é 8192

kbytes/segundo. Isto significa que o CC é 51,2 vezes mais rápido que o cifrador caótico proposto em [51].

Como os ambientes de teste são diferentes, a comparação dos resultados precisa ser aproximada. Uma aproximação linear é suficiente para termos boa idéia da diferença de desempenho dos dois cifradores. Mesmo que o cifrador apresentado em [51] executasse em uma máquina com 5 vezes mais poder de processamento, ainda assim, se obteria 800kbytes/segundo, muito inferior aos 8192kbytes/segundo do CC.

Devido a suas propriedades, o desempenho do CC não é afetado pelo desbalanço dos textos planos utilizados no teste. Os mesmos testes foram repetidos para diferentes textos planos viesados. Para todos os cifradores estudados neste trabalho, não foi encontrada nenhuma influência no desempenho de cifração e decifração.

É importante mencionar que a implementação do AES e do Sosemanuk utiliza uma biblioteca de alto desempenho [52]. Essa biblioteca verifica o tipo de processador usado e escolhe diferentes rotinas internas para executar a cifração e decifração de maneira otimizada. As rotinas são escritas em linguagem Assembly e são especializadas para extrair o máximo desempenho do *hardware* usado. Para o RC4, foi utilizada a implementação disponível no código do sistema operacional FreeBSD, ou seja, também uma implementação bem escrita visando o melhor desempenho possível. É provável que uma implementação do CC no mesmo nível de eficiência dos outros algoritmos o conduziria a resultados melhores.

Outro fator a ser levado em conta é o diferente desempenho dos algoritmos em *hardwares* diferentes [53, 54]. É necessário notar também que para arquivos pequenos (abaixo de 500 bytes), o CC atinge desempenho muito próximo ao AES, Sosemanuk e RC4. Além disso, a taxa de cifração atual, que é 8192kbytes/segundo, já está dentro do limite prático para utilização comercial. Mesmo assim, o desempenho do CC pode aumentar mais com otimizações de código.

5.2 Analise de segurança

Dizer que um algoritmo criptográfico é realmente seguro, é uma afirmação muito delicada. Ela deve ser baseada em vários aspectos como análise dos símbolos cifrados, arquitetura de software, análise teórico-matemática do algoritmo e ataques propostos

pela comunidade. Baseados nos tipos de ataques gerais descritos na Seção 2.4, surgiram diversas propostas de criptoanálise de cifradores caóticos. Abaixo, é analisado o comportamento do CC diante dos ataques mais relevantes a cifradores caóticos.

Análise de Entropia

Conforme [49], a maioria dos mapas caóticos têm regiões de seus domínios que são visitadas com maior freqüência que outras. Mesmo que os parâmetros do mapa caótico mudem, continuarão existindo regiões no domínio do mapa caótico com maior taxa de visitação. A criptoanálise proposta em [8] explora essa característica. Como existem regiões mais visitadas que outras, a taxa de visitação de sítios correspondentes a cada símbolo também é diferente. Portanto, de posse de um conjunto de textos planos, é possível criar uma tabela relacionando os símbolos do alfabeto e as respectivas quantidades médias de iterações utilizadas para cifrar cada um dos símbolos. Após uma certa quantidade de textos planos, a tabela fica suficientemente completa para se estimar textos planos, apenas observando seus respectivos textos cifrados.

A Tabela 5.1 é um exemplo fictício de uso desse método de criptoanálise para uma amostra de texto plano, utilizando o alfabeto $\mathcal{A} = \{a, b, c\}$. Na tabela, em negrito, esta destacada a discrepância estatística para duas e quatro iterações. Conforme o exemplo, na iteração 2, o símbolo c aparece com mais freqüência que outros. Sempre que surgir no texto cifrado um valor correspondente a 2 iterações, podemos inferir que, muito provavelmente, este valor se refere ao caracter plano c.

O mesmo ocorre com 4 iterações, neste caso, inferimos que o possível símbolo plano é b. Se criarmos mais tabelas desse tipo com outros textos planos, novas tendências para outros símbolos planos irão surgir, e outras se confirmarão. Cruzando-se a informação de várias tabelas, tem-se um mapa geral da relação entre os símbolos planos do alfabeto e o número de iterações, as quais são informações contida no texto cifrado e conhecidas por todos. Com isso, é possível estimar os textos planos, mesmo que a condição inicial do mapa caótico (chave) seja alterada.

No caso do algoritmo CC isso não ocorre porque segundo proposição definida em [26] todos os sítios são uniformemente visitados, independentemente da chave escolhida. A Tabela 5.2 mostra a criptoanálise de entropia aplicada ao CC. Para o teste foi utilizada a chave $K = \{x_0 \rightarrow 2, p \rightarrow 2\}$, isto é, $\sqrt{2}$. O alfabeto utilizado

TABELA 5.1 – Exemplo de tabela de ataque de entropia.

Iterações	Freqüência	Freq. de a	Freq. de b	Freq. de c	Provável Símbolo
1	0,1419	0,0456	0,0483	0,0481	não identificado
2	0,1356	0,0411	0,0420	**0,0524**	símbolo c
3	0,1509	0,0498	0,0475	0,0536	não identificado
4	0,1338	0,0404	**0,0510**	0,0425	símbolo b
5	0,1229	0,0404	0,0413	0,0411	não identificado
6	0,1006	0,0335	0,0342	0,0329	não identificado
7	0,0631	0,0210	0,0213	0,0208	não identificado
8	0,0510	0,0163	0,0154	0,0192	não identificado
9	0,0394	0,0158	0,0115	0,0121	não identificado
10	0,0279	0,0083	0,0085	0,0112	não identificado

foi $\mathcal{A} = \{a, b, c\}$. Foram utilizados no teste dez textos planos não viesados. A Tabela 5.2 mostra os resultados de um dos textos planos do teste. Ela foi limitada a exibir o histograma de 10 iterações. Acima desse valor as taxas de iteração são insignificantes. Todas as outras tabelas têm comportamentos semelhante. Como pode ser observado, os dados extraídos são inconclusivos em termos de criptoanálise, pois as proporções de iterações são muito parecidas para todos os símbolos do alfabeto.

TABELA 5.2 – Tabela de ataque de entropia para o cifrador CC.

Iterações	Freqüência	Freq. de a	Freq. de b	Freq. de c	Provável Símbolo
1	0,1364	0,0433	0,0481	0,0450	não identificado
2	0,1355	0,0445	0,0452	0,0458	não identificado
3	0,1491	0,0528	0,0466	0,0498	não identificado
4	0,1265	0,0426	0,0422	0,0417	não identificado
5	0,1204	0,0403	0,0409	0,0392	não identificado
6	0,1027	0,0351	0,0328	0,0349	não identificado
7	0,0658	0,0221	0,0232	0,0205	não identificado
8	0,0524	0,0156	0,0177	0,0190	não identificado
9	0,0447	0,0162	0,0121	0,0164	não identificado
10	0,0281	0,0092	0,0098	0,0090	não identificado

A propriedade do CC de eqüiprobabilidade de visitação dos sítios não deixa transparecer seu comportamento diferenciado em cada cifração, mesmo quando a

mesma chave é utilizada. A não transparência do comportamento do CC é verificada no experimento descrito a seguir. O experimento mostra os histogramas de cifração de quatro conjuntos de textos com diferentes vieses. Cada conjunto é composto de 10 amostras de textos com os mesmos vieses. Todos os textos têm tamanho de 1MB. A média de histograma $h^o(n)$ foi calculada para cada um dos quatro conjuntos de amostras. Mais precisamente, sendo C^{ij} a versão cifrada de P^{ij}, para cada conjunto de textos planos $P^i = (P^{i1}, P^{i2}, \dots P^{ij})$, com $j = 1, 2, \dots 10$, $i = 1, 2, 3, 4$ e $n \in \mathbb{N}$, foi calculada a média

$$h^o(n) = \frac{1}{10} \sum_{j=1}^{10} h^{ij}(n),$$

onde

$$h_j(n) := \sharp\{c^{ij} : c^{ij} = n, j = 1, ..., L_i\}.$$

A respectiva variância de $\sigma(n)$ também foi calculada. Para cada texto plano, foi utilizada uma chave diferente. O alfabeto utilizado $\mathcal{A}$ foi o ASCII, com d-bloco=3.

O primeiro conjunto foi composto de 10 seqüencias pseudo-aleatórias, com distribuição uniforme de símbolos de $\mathcal{A}$. O segundo conjunto foi composto de 10 textos com seqüencia pseudo-aleatória viesada e distribuição uniforme dos símbolos de $\mathcal{A}$, exceto para o símbolo a, que aparece com freqüência dez vezes maior que os outros símbolos. O terceiro conjunto é parecido com o segundo, mas com a dupla de símbolos ab aparecendo dez vezes mais freqüentemente que qualquer outra dupla de símbolos. E o quarto conjunto segue igual ao terceiro, mas com o trio the aparecendo com dez vezes mais freqüência do que outros trios de símbolos.

A Figura 5.2 mostra os resultados obtidos para cada conjunto de textos. Ela mostra a média dos histogramas $h^o(n)$ bem como $h(n) = h^o(n) \pm \sigma(n)$, sendo $\sigma(n)$, o desvio padrão médio de $h^o(n)$. Desses resultados é possível concluir que o CC não deixa rastros sobre diferenças entre textos planos, mesmo quando elas são marcantes, como no caso dos conjuntos 2,3 e 4. O experimento apresentado é outra indicação que Ataques de Entropia não é factíveis contra o CC, pois os textos planos não transparecem ou dão indícios visíveis de seus padrões nos textos cifrados.

Como complemento, foram gerados os gráficos das Figuras 5.3, 5.4 e 5.5, os quais corroboram com a proposição matemática definida em [26], a qual afirma que os sítios do mapa caótico têm visitação uniforme. Os gráficos foram gerados para o mesmo

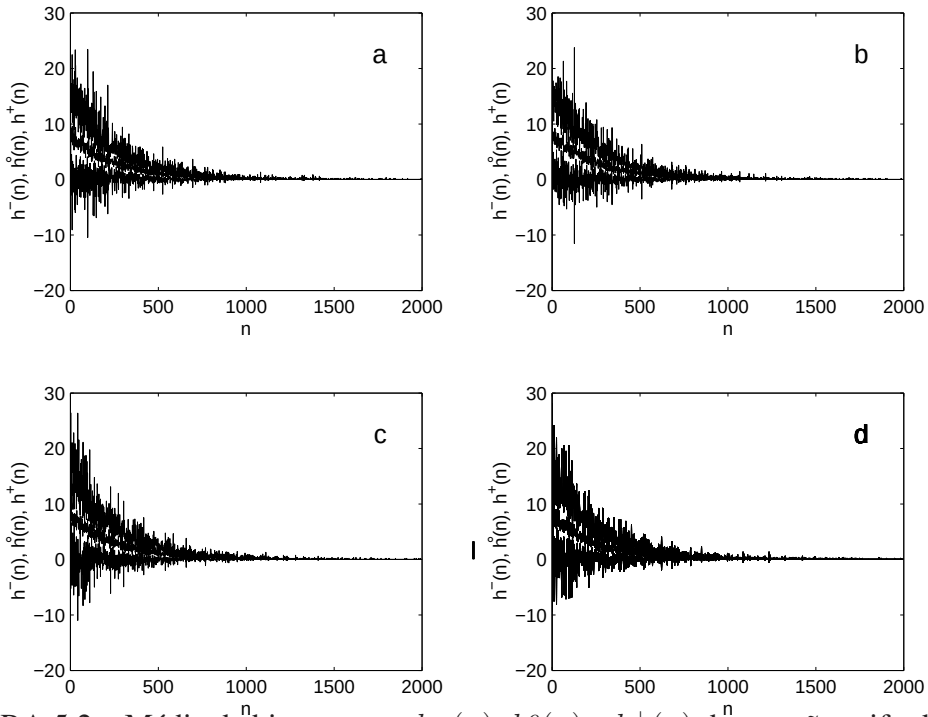

FIGURA 5.2 – Média de histogramas $h^{-}(n)$, $h^{o}(n)$ e $h^{+}(n)$ das versões cifradas de conjuntos de textos viesados e não-viesados. O histograma obtido para o conjunto de textos não-viesado (a) é estatísticamente similar ao conjunto viesado onde o símbolo a aparece com dez vezes mais freqüência do que qualquer outro símbolo (b). Os Histogramas (c) e (d) mostram resultados similares para textos planos com o mesmo viés para os grupos de símbolos ab e the, respectivamente.

alfabeto $\mathcal{A} = \{a, b, c\}$ usado no primeiro experimento desta seção. Eles são histogramas que contabilizam o número de iterações para o arranjo simples do alfabeto $\mathcal{A}$, tomando os símbolos 2 a 2. O Gráfico 5.3 contabiliza as iterações para os arranjos $\mathcal{A} = \{ab, ac\}$. O histograma 5.4 contabiliza as iterações para os arranjos $\mathcal{A}' = \{ba, bc\}$. O histograma 5.5 contabiliza as iterações para os arranjos $\mathcal{A}'' = \{ca, cb\}$. Os histogramas de $\mathcal{A}$ têm comportamento muito similar, o mesmo ocorre com $\mathcal{A}'$ e $\mathcal{A}''$. Mesmo comparando todos os histogramas entre si, nota-se que todos são similares. Esse comportamento do CC comprova a visitação eqüanime de todos os símbolos do alfabeto, independente de suas posições no texto plano, sendo mais uma evidência de que o ataque por Análise da Entropia não tem sucesso contra o CC.

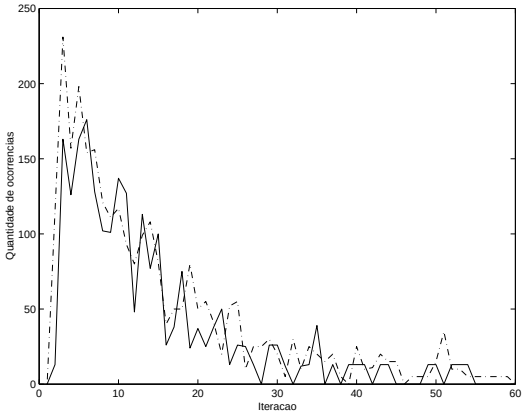

FIGURA 5.3 – Histograma de iterações para cifração de símbolos partindo de "a"para "b"(linha cheia) e de símbolos partindo de "a"para "c"(linha ponto-traço).

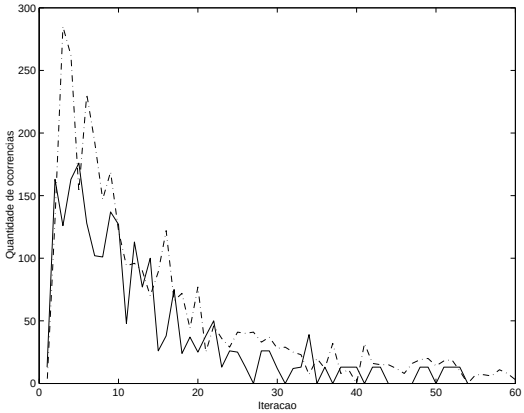

FIGURA 5.4 – Histograma de iterações para cifração de símbolos partindo de "b"para "a"(linha cheia) e de símbolos partindo de "b"para "c"(linha ponto-traço).

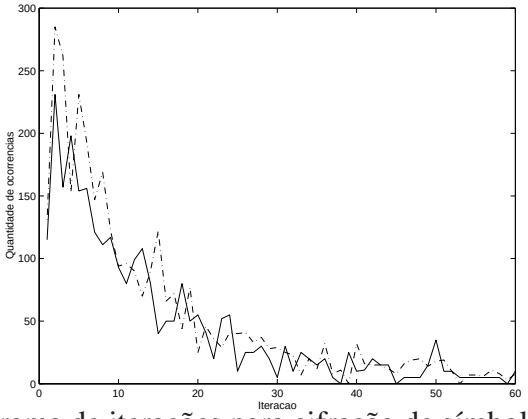

FIGURA 5.5 – Histograma de iterações para cifração de símbolos partindo de "c"para "a"(linha cheia) e de símbolos partindo de "c"para "b"(linha ponto-traço).

One-time pad attack

Segundo o trabalho [8], o cifrador de Baptista, assim como outros da mesma família, têm falhas de segurança quando a mesma chave é utilizada na cifração de vários textos planos. Neste ataque, o criptoanalista gera muitos textos planos com pequenas diferenças e compara os resultados de suas cifrações para inferir o comportamento do mapa caótico. Este problema foi contornado no CC. Em seu processamento o CC implementa um Modo de Operação similar ao CBC. A cada símbolo cifrado, n sítios são percorridos no domínio do mapa caótico. O valor n depende do tipo de símbolo plano que foi cifrado. O próximo símbolo plano inicia a cifração a partir da sítio inicial i adicionado ao somatório das n posições iteradas por todos os símbolos anteriores, isto é, cada símbolo cifrado depende de todos os símbolos anteriormente cifrados. Com isso, aplicando o ataque proposto em [8] e sendo:

$P^1 = (P^1_1, P^1_2, P^1_3, \ldots, P^1_L)$, o texto plano 1;
$P^2 = (P^2_1, P^2_2, P^2_3, \ldots, P^2_L)$, o texto plano 2;
$C^1 = (C^1_1, C^1_2, C^1_3, \ldots, C^1_L)$, a versão cifrada de P^1;
$C^2 = (C^2_1, C^2_2, C^2_3, \ldots, C^2_L)$, a versão cifrada de P^2.

Tem-se as seguintes situações:

- Para $P^1_1 \neq P^2_1$, os textos cifrados são $C^1_1 \neq C^2_1, C^1_2 \neq C^2_2, C^1_3 \neq C^2_3, \ldots, C^1_L \neq C^2_L$.
- Para $P^1_1 = P^2_1, P^1_2 \neq P^2_2$, os textos cifrados são $C^1_1 = C^2_1, C^1_2 \neq C^2_2, C^1_3 \neq C^2_3, \ldots, C^1_L \neq C^2_L$.
- Para $P^1_1 = P^2_1, P^1_2 = P^2_2, P^1_3 \neq P^2_3$, os textos cifrados são $C1_1 = C^2_1, C^1_2 = C^2_2, C^1_3 \neq C^2_3, \ldots, C^1_L \neq C^2_L$ e assim por diante para o restante dos símbolos planos.

Nota-se neste caso que o comportamento do CC pode gerar lacunas de segurança para o *One-time pad attack* pois, sempre que dois textos planos têm a mesma seqüencia inicial de símbolos, seus respectivos textos cifrados serão iguais para essa seqüencia.

Para evitar este comportamento, foi adicionado um novo elemento na cifração dos dados, o qual elimina este comportamento sem comprometer a segurança do mesmo.

O elemento novo consiste em incluir um contador ξ no processo de cifração. Este contador é utilizado no início da cifração como índice do mapa caótico.

Por exemplo, para um texto plano formado pelo conjunto de símbolos do alfabeto ASCII (256 símbolos), e para a chave $\sqrt{2}$, temos a órbita caótica:

TABELA 5.3 – Órbita caótico do CC para chave $\sqrt{2}$ e d-bloco=3, onde $\Delta(x)$ é o valor do x-ésimo d-bloco.

índice x	1	2	3	...	325	326	327	328	329	...
$\Delta(x)$	414	142	421	...	187	387	870	701	012	...

Sendo, $\xi = 1$, o cifrador inicia a cifração no índice 414 ou seja, $\Delta(414) = 196$. Supondo que o primeiro texto finalize a cifração no índice 325, a cifração do segundo texto plano iniciará no índice $325 + (\Delta(329) = 012) = 337$ e assim por diante. Nota-se que o salto do índice 325 para o índice 329 dissipa qualquer relação com o último $d-bloco$ utilizado na cifração do primeiro texto. Ou seja, o salto para iniciar os próximos textos planos sempre se dá em função do tamanho dos $d - blocos$, obedecendo a regra: Tamanho($d - bloco$)+1. Com isso, mesmo utilizando a mesma chave para textos planos iguais ou muito parecidos, os textos cifrados serão sempre diferentes, pois a cifração de cada texto plano iniciará em um ponto diferente do domínio do mapa caótico. O decifrador deve manter o mesmo ξ para que os dados decifrados sejam coerentes com o texto plano. Se o salto ultrapassa a órbita caótica, o processo continua do início da mesma, como uma fila circular.

A introdução deste elemento não gera falhas de segurança pois, conforme o artigo [26] os sítios são igualmente distribuídos e o mapa caótico tem entropia máxima. Desta forma, o conjunto ξ é composto pelos elementos $\xi = \{000, 001, 002, ..., 999\}$ distribuídos igualmente e de forma pseudo-aleatória. É possível adotar tamanhos de ξ diferentes do tamanho dos $d - blocos$.

Sendo assim, o valor 414, utilizado quando $\xi = 1$, é totalmente aleatório, o mesmo ocorre quando $\xi = 337$ ou $\xi = 337 + n$. O criptoanalista não consegue prever qual o número inicial de interações do sistema pois, a cada novo ξ, as chances são as mesmas para qualquer valor do conjunto $\xi = \{000, 001, 002, ..., 999\}$.

Embora este elemento não tenha sido incluído ao CC implementado em software, ele não impacta na velocidade de cifração/decifração. O único custo adicionado com este elemento é o salto inicial de ξ, somente na cifração do primeiro símbolo. Esta nova característica do CC o torna um cifrador probabilístico. Um cifrador probabilístico caracteriza-se por sempre gerar textos cifrados diferentes, mesmo que os textos planos sejam iguais. Com isso, o CC aprimora sua segurança não só contra o ataque *One-time pad attack*, mas também para quaisquer outros possíveis ataques de Texto Plano Escolhido e Adaptado.

Ataque de Jakimoski - Kocarev

Abordado em [9], esse é um Ataque por Texto Plano Conhecido. Ele gera uma tabela com o total de iterações necessárias para alcançar cada símbolo plano. As iterações são computadas desde o início da cifração. Com essa tabela, é possível comparar as iterações necessárias para outros textos planos e estimar o mapa caótico do cifrador.

Por exemplo, sendo $P^1 = \{subject\}$ e $P^2 = \{to\}$, dois textos planos conhecidos, seu respectivo textos cifrados são $C^1 = \{272, 258, 305, 285, 314, 276, 422\}$ e $C^2 = \{254, 267\}$. Com estes dados a Tabela 5.4 é gerada.

TABELA 5.4 – Tabela de Jakimoski - Kocarev comparando os símbolos planos e o total de iterações desde o início da cifração até atingir os sítios relacionados a estes símbolos [9].

n	254	272	521	530	835	1120	1434	1710	2132
símbolo	t	s	o	u	b	j	e	c	t

Com a Tabela 5.4 é possível decifrar textos que contenham símbolos mapeados na tabela. Por exemplo, o texto cifrado $C^3 = \{272, 249\}$ pode ser inferido como sendo $P^3 = \{so\}$. O valor 272 corresponde ao símbolo s e 272+249=521, denotando o símbolo o.

A modificação feita no CC contra o *One-time Pad Attack* elimina qualquer chance de sucesso desse ataque. A construção da tabela de Jakimoski-Kocarev torna-se inviável visto que, a cada cifração de um novo texto plano, o criptoanalista desconhece o ponto

de inicio do somatório de iterações dos símbolos.

Os ataques apresentados até o momento são os principais referentes a cifradores caóticos. Além desses ataques, é possível que a criptoanálise de cifradores caóticos se utilize de técnicas tradicionalmente utilizadas nos cifradores modernos. Alguns dos tipos de ataques mais comuns a cifradores modernos são apresentados resumidamente em 2.4. A seguir são analisados suas aplicabilidade e eficiência sobre o CC.

Dos tipos de ataque apresentados em 2.4, o ataque por escorregamento (slide attack) não é aplicável ao CC, pois não há como separa o processo de cifração em rodadas. O processo de procura de d-blocos, dentro de Γ é único, não existem maneiras de dividi-lo em partes. Existe apenas uma única lógica de cifração que consiste em percorrer o domínio do mapa caótico. Além disso o CC não divide sua chave em partes, utiliza-a de forma completa para gerar o valores do domínio do mapa caótico.

Ataque Linear

O CC é um sistema caótico. É sabido que sistemas caóticos, quando linearizados, podem representar o comportamento real do sistema apenas por um período curto de tempo [45]. A linearização destes sistemas logo destoam do comportamento real devido a sensibilidade as condições iniciais e a descrição inexata do sistema real através de equações lineares.

A cada iteração, o mapa caótico do CC acumula a probabilidade de ocorrências de cada sítio iterado. Isso resulta em comportamentos totalmente diferentes para cada cifração. O ataque linear, como foi pensado originalmente, foca a inspeção das Caixas-S e outros elementos não lineares dos cifradores. No caso do CC, a cifração é baseada totalmente na não linearidade do seu sistema. A cada símbolo cifrado, é injetada uma nova porção de não-linearidade ao texto cifrado. Isso ocorre de forma cumulativa, isto é, cada símbolo cifrado carrega sua não-linearidade somada a não-linearidade acumulada de todos os símbolos cifrados anteriormente. Isso torna inviável o Ataque Linear, visto que a complexidade da relação entre os símbolos cifrados cresce proporcionalmente ao tamanho do texto cifrado.

Ataque Diferencial

Este ataque utiliza o princípio de comparar textos cifrados com a mesma chave, originados de textos planos com pequenas diferenças. Para analisar o comportamento do CC em relação a esse método de criptoanálise, foi realizado um experimento nos mesmos moldes do ataque. O experimento levou em conta um conjunto de 64 textos planos. Cada texto com tamanho de 10 Mbytes. O alfabeto utilizado foi o ASCII com 256 símbolos. A diferença entre os textos planos da amostra era de apenas em um byte, localizado na mesma posição, exatamente na metade de cada texto plano. O teste foi realizado sem repetibilidade dos valores de Γ, isto é, o CC gerou apenas uma órbita caótica, utilizando a chave $K = (\pi_{16}, p)$, sendo $p = 2$ e π_{16}, a aproximação do valor de π em 16 casas decimais. Além disso, não foi utilizada ϵ-segurança.

Devido ao CC utilizar a mesma chave K para cifrar todos os textos planos e, estes se diferenciarem apenas na posição do byte 5242880, o qual esta exatamente na metade dos textos planos, os textos cifrados se diferenciaram a partir desta posição até o seu final. A Tabela 5.5 contém uma pequena amostra dos dados extraídos do teste. Na representação da tabela, todos os símbolos planos foram convertidos em valores entre 0 e 255. A tabela foca exatamente a posição 5242880, que diferencia os textos planos. Observando a tabela, nota-se que os textos cifrados se diferenciam a partir da posição 5242880. O fato importante é que, mesmo com as próximas posições 5242881, 5242882... de seus textos planos exatamente iguais, eles não apresentam qualquer indício de relação.

Nenhuma evidência estatística relevante foi encontrada comparando-se os textos cifrados e seus respectivos textos planos. Como resultado prático, foi calculado desvio padrão dos valores das posições 5242880 dos textos cifrados $\{C^0, C^2, ...C^{63}\}$. O mesmo foi feito para as posições $\{5242881..5242890\}$. Em média o desvio padrão foi 128,27. Neste caso, isso significa uma distribuição não uniforme de dados. Observando as seqüencias geradas, nenhuma similaridade pode ser encontrada a partir da posição 5242880 dos textos cifrados. Portanto, este teste é um indício de que este tipo de ataque é ineficiente contra o CC.

TABELA 5.5 – Amostra do resultado do teste de Ataque Diferencial ao CC. Sendo P^n=textos planos e C^n=textos cifrados.

Identificação do texto	valores de textos planos e cifrados				
P^0	056	105	095	141	028
C^0	099	222	206	354	186
P^1	056	105	096	141	028
C^1	099	222	057	182	078
P^2	056	105	097	141	028
C^2	099	222	069	107	251
P^3	056	105	098	141	028
C^3	099	222	349	331	125
...	...	...	...	...	...
P^{63}	056	105	159	141	028
C^{63}	099	222	361	068	137
posição dos bytes nos textos	**5242878**	**5242879**	**5242880**	**5242881**	**5242882**

Ataque por Correlação

No caso do CC, este ataque deve ser adaptado para verificar a relação entre os bits/bytes cifrados e a chave. Esse ataque é mais efetivo quando relaciona os bits/bytes iniciais do texto cifrado. Embora este tipo de ataque possa ser aplicado a cifradores caóticos, até o momento não existe nenhum ataque conhecido que utilize este método de criptoanálise para cifradores caóticos e, conseqüentemente, para o CC. A ineficiência desse ataque para o CC se deve ao fato dos sistemas caóticos serem altamente sensíveis às condições iniciais. Isso faz com que a relação entre a chave (condição inicial) e o texto plano se distancie muito rapidamente e aumente mais e mais, a cada nova iteração no mapa caótico. Mais ainda, conforme a forma de criptografia caótica estudada neste trabalho, o texto plano influência diretamente no comportamento do sistema caótico, na mesma proporção que a chave. Portanto, mesmo em sistemas onde o criptoanalista tem acesso a muitas mensagens, será muito difícil encontrar uma relação lógica entre chave e textos planos. Um ótimo exemplo disso é o teste aplicado no Ataque Diferencial, o resultado parcial apresentado na Tabela 5.5 mostra que variando apenas um byte dentro de textos planos, os bytes seguintes se diferenciam completamente.

Ataque à memória utilizada pelo CC

Como pode ser visto na Seção 4.1 o CC mantém em memória uma tabela Γ com a seqüencia de valores do domínio do mapa caótico utilizado. Este é um aspecto negativo, pois a área de memória reservada para a órbita utilizada do mapa caótico é passível de investigação. É possível retirar Γ do CC, evitando tal ataque à memória. Porém, isso implica em gerar partes de Γ a cada iteração, visto que ela não está mais em memória. Conseqüentemente, o processo se torna menos eficiente.

A inspeção de máquinas cifradoras é uma forma comum de ataque, principalmente quando as máquinas são computadores pessoais. Este tipo de ataque também ocorre para outros cifradores. Um dos ataques propostos contra o AES explora justamente a análise da memória usada pelo cifrador. O ataque analisa as transições entre rodadas de cifração para identificar a chave e ou prever o comportamento do AES para inferir os textos planos cifrados com a mesma chave. O RC4 também é suscetível a esse tipo de ataque. No caso do RC4 o criptoanalista analisa as transições de seu vetor de inicialização para inferir os textos planos gerados pelo RC4.

Este tipo de ataque fica comprometido quando o *hardware* utilizado é especializado. Um exemplo disso são os HSM (Hardware Security Module). Nesses equipamentos, o acesso é limitado e protegido. Eles são usados para cifrar dados, armazenar chaves e até gerar chaves. Cifrando os dados em um HSM, o cifrador fica imune a ataques à sua memória. Tal recurso pode ser combinado com o CC para aumentar a segurança em sistemas criptográficos. Outra maneira é limitando o acesso a máquina que executa o CC. Neste último caso, a preocupação se limita ao acesso via rede.

Outras considerações sobre segurança

O NIST sugere uma série de testes empíricos que podem ser aplicados a cifradores e geradores de números aleatórios. Estes testes costumam ser usados como base de avaliação do nível de segurança de cifradores em geral [55].

Os testes do NIST focam a análise do comportamento de bits e bytes dentro dos textos cifrados e a relação de bits e bytes entre textos cifrados. Nenhum dos testes empíricos propostos pelo NIST podem ser aplicados ao CC pois, como dito

anteriormente, eles buscam padrões de bits e bytes. No CC, além de a unidade de cifração ser composta de 2 bytes, o significado que ela carrega cria muitos padrões no texto cifrado. Ao contrário do que se possa imaginar, esses padrões não significam falhas de segurança, a medida que o mesmo padrão é repetido para qualquer texto. Para esclarecer melhor este assunto, tomamos os textos planos $P^1 = (abcde)$, $P^2 = (aaaaa)$ e os respectivos textos cifrados $C^1 = \{(0,1),(0,2),(1,2),(0,4),(0,1),(0,5)\}$, $C^2 = \{(0,4),(0,3),(0,2),(0,5),(0,1),(0,3)\}$. Podemos observar muitos padrões dentro de C^1 e C^2 como, a cada dois bytes, um é geralmente zero. Outro padrão neste exemplo são a repetição de muitos valores entre $[0..5]$. Conforme citado no capítulo anterior, na cifração, o CC gera muitos símbolos em torno da faixa de valores próxima ao tamanho do alfabeto utilizado. Os testes do NIST entendem esses padrões como possíveis falhas de segurança, pois não sabem interpretar um conjunto de símbolos qualquer, por exemplo $(0,3)$, como apenas um valor representando um símbolo de texto plano. Em alguns testes são analisados conjuntos de bytes, mesmo nesses existe o problema dos padrões. O problema é que, para o CC, o significado semântico dos símbolos cifrados é diferente do significado interpretado pelos testes do NIST.

Além disso, o fato de o CC gerar uma seqüencia de números altamente tendenciosa, não significa nada, pois a dupla cifrada $\{0,1\}$ pode representar qualquer símbolo de qualquer conjunto de textos planos e, qualquer símbolo plano pode ser representado por qualquer dupla de símbolos cifrados. Essa leitura dos símbolos cifrados não é feita pelos testes do NIST.

5.3 Características do CC

O NIST dividiu os critérios de escolha do AES em três items. O primeiro refere-se à segurança. Os outros dois referem-se a custo e características do algoritmo [56]. O critério de custo engloba os fatores de velocidade, uso de memória, portabilidade para várias plataformas (8, 16, 32 e 64 bits) e tamanho variável das chaves. O critério de características do algoritmo engloba os fatores de simplicidade de implementação, portabilidade para software e *hardware*, paralelismo e adaptabilidade [56, 57]. O critério de segurança, e o fator de velocidade já foram abordada nas seções anteriores. A seguir, o CC é analisado sobre as outras características de avaliação do NIST.

Uso de memória

O uso de memória no CC é dependente da tabela Γ. Esta tabela pode ser maior ou menor conforme o alfabeto utilizado e a quantidade de d-blocos que ela guarda.

Como exemplo, para um ambiente com alfabeto de 256 símbolos, textos planos com tamanho de 1024 bytes e geração do mapa caótico a cada 10 textos cifrados, temos um total de 1024 bytes por texto plano $\times$ 10 textos planos = 10.240 bytes, a serem cifrados. Para este alfabeto, o algoritmo utiliza, em média, 256 posições do domínio do mapa caótico para cada símbolo cifrado. Cada posição do mapa caótico ocupa 4 bytes de memória. No caso do uso de Γ (sem reutilização), temos a seguinte estimativa para a memória requerida:

$10.240 \times 256 \times 4 = 10.485.760 bytes$, ou seja, 10.240KB.

Dependendo da aplicação, esta quantidade de bytes pode ser aceitável, em outras elevada. Em qualquer dos casos, esse tamanho é ajustável. Quanto maior a tabela, menor o tempo gasto em processamento de novos pontos do domínio do mapa, e vice-versa. No modo de uso de Γ sem reutilização, quando todos os sítios de Γ forem utilizados, um novo conjunto de valores é gerado. Mas Γ pode ser utilizada de maneira cíclica, isto é, usadas todas posições de Γ, o CC volta ao início da tabela e reutiliza os mesmos valores para cifrar outros textos. Se a opção for reutilizar os mesmos pontos da tabela para cifração, economiza-se processamento e memória, porém a segurança é diminuída.

Portabilidade para várias plataformas

Uma das grandes vantagens do CC em relação a outros algoritmos caóticos é a sua independência de máquina. Outros algoritmos sofrem problema de degradação do caos, enquanto o CC se mantém independente de plataforma (8, 16, 32 ou 64 bits). O desempenho do CC neste ponto é muito dependente do tamanho do alfabeto. O CC pode ser implementado em uma máquina de 8 bits de barramento, mesmo que o tamanho dos símbolos do alfabeto seja de 16 bits. Neste caso, o desempenho será diminuído devido a

limitação no barramento de dados, o qual seria utilizado duas vezes para transmitir cada dado do CC. No caso do barramento de dados ser maior, o uso do *hardware* pode ser otimizado para aproveitar o máximo do barramento, passando 2 ou mais dados por vez, ou ainda, ignorar esse recurso de *hardware*, utilizando apenas 16 bits de barramento.

Tamanho variável das chaves

O uso de chaves de tamanhos variados objetiva tornar o cifrador mais adaptável a vários meios. Em geral, tamanhos de chaves menores significam processamento mais rápido, porém com menor segurança. Em ambientes onde a segurança não necessita ser tão forte, com memória e processamento limitados, o tamanho da chave pode ser um fator muito importante para aumentar o desempenho. Por definição, a chave no CC é a raiz n-ésima de um número irracional. Portanto, seu tamanho pode variar conforme a necessidade do ambiente onde esta implantado. A maior vantagem do CC neste quesito é que a força da criptografia independe do tamanho da chave utilizada. Ela se mantém constante pois independente do tamanho utilizado, as chaves são sempre números irracionais, os quais sempre mantém as mesmas propriedades e, conseqüentemente, o mesmo comportamento dentro do CC. Isso não ocorre com o AES, Sosemanuk e RC4 e mesmo com outros cifradores, onde o tamanho da chave esta atrelado diretamente à segurança.

Simplicidade de implementação

O CC pode ser dividido em quatro partes, todas elas são simples e de fácil implementação. A Tabela 5.6 mostra os sub-processos nos quais o CC pode ser dividido.

A geração do mapa caótico é a tarefa mais complicada. Ela extrai gradativamente os dígitos de um número irracional. A Geração da tabela Γ consiste em relacionar a órbita do mapa caótico ao alfabeto existente. O teste de segurança do expoente de Lyapunov é uma verificação simples que, após um cálculo pré-determinado verifica se o valor esta dentro do limite de segurança exigido. A procura dentro de Γ dos pontos correspondentes cada símbolo plano (cifração propriamente dita) também é uma tarefa simples. Ela se resume a comparar na linha correspondente ao símbolo plano o valor mais adequado a cifração, isto é, uma tarefa simples de comparação dentro de uma lista

TABELA 5.6 – Lista de sub-processos internos do CC.

Sub-Processo	Descrição do Sub-Processo.
A	Geração da órbita do mapa caótico.
B	Geração da tabela Γ.
C	Teste de segurança do expoente de Lyapunov.
D	Identificação, em Γ, do ponto dentro do domínio do mapa caótico que esta relacioando ao símbolo plano.

de possibilidades. Essa tarefa deve também identificar quando todos os valores de Γ foram utilizados e disparar a geração de novos valores para Γ. Sub-dividindo o CC em diversas partes, fica clara sua simplicidade de implementação. A decifração é ainda mais simples. Ela é composta das fases A e D. A decifração não precisa verificar se o fator de segurança esta adequado e nem precisa gera a tabela Γ, pois cada símbolo cifrado que é lido, aponta diretamente para o índice da órbita gerada pelo mapa caótico que contém o símbolo plano. Portanto, se comparado a outros cifradores o CC tem implementação muito simples, tanto na cifração, quanto na decifração.

Portabilidade para software e hardware

Conforme descrito anteriormente, por ser composto de tarefas simples, a portabilidade do CC para software e *hardware* é viável. Em software, todas os sub-processos que compõe o algoritmo são implementadas sem dificuldades, inclusive com o uso de bibliotecas especializadas para cálculo de números de alta precisão. No caso do *hardware*, todas as tarefas têm ações claras com poucas mudanças no fluxo de processamento, ou seja, cada parte do processo de cifração pode ser subdividida em pequenas tarefas simples. Essas características são essenciais para portar um algoritmo de forma eficiente para *hardware* [3].

Paralelismo

Como algumas sub-tarefas do CC não dependem de resultados prévios de outras sub-tarefas, é possível implementar cada uma delas independente da outra, tornando

o CC altamente paralelizável. Essa característica é muito importante, principalmente para implementações em *hardware*. A forma com que o CC foi concebido permite a criação de um fluxo de processamento pouco complexo e constante, isso diminui consideravelmente o tempo de processamento do algoritmo.

Os Diagramas UML de Estado e Seqüencia das Figuras 5.6 e 5.7 mostram mais claramente como o CC pode ser internamente paralelizável. O Diagrama de Estados da Figura 5.6 mostra graficamente a interação dos quatro processos (A, B, C e D) em que ele é divido. A primeira vista, nota-se a dependência entre os processos A, B, C e D. Embora exista ligação entre os processos, o Diagrama de Estados mostra como o foco de processamento alterna de uma sub-parte para outra conforme a ocorrência de eventos. A dependência vista no diagrama de Estados não é necessariamente indício de seqüencia linear de processamento.

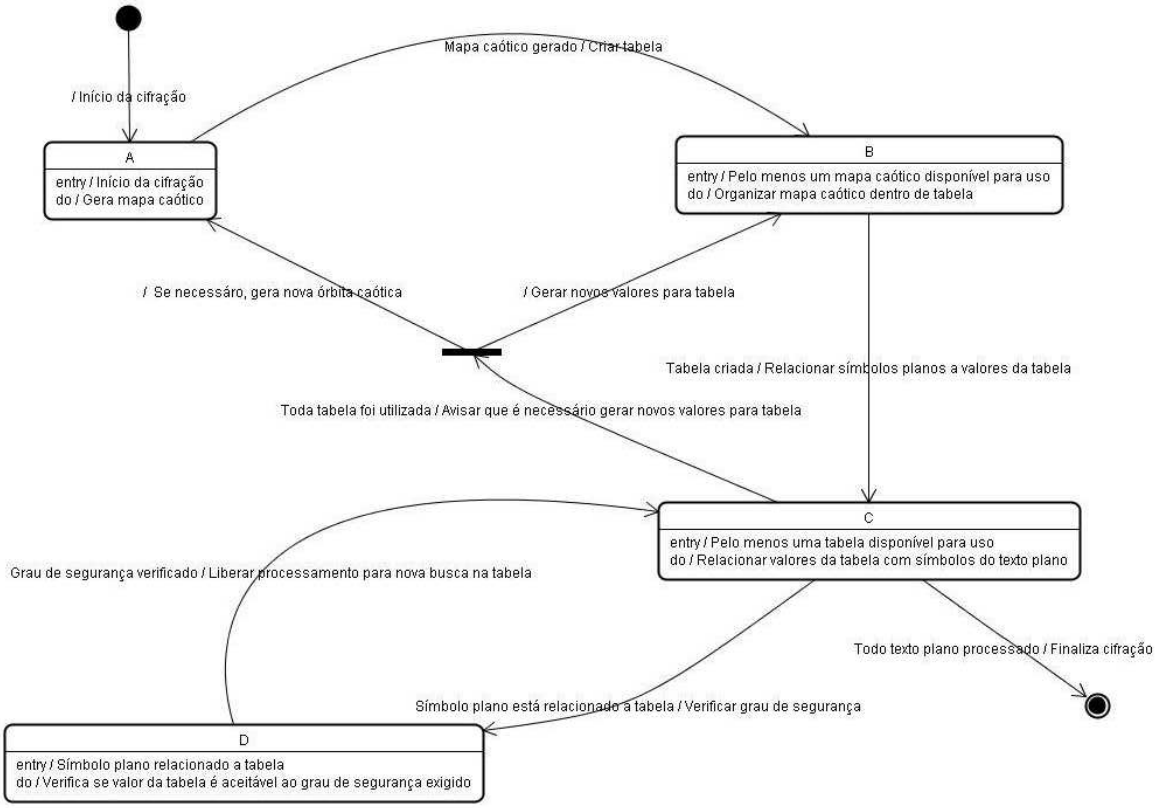

FIGURA 5.6 – Diagrama de Estados do algoritmo CC.

Com o Diagrama de Seqüencia da Figura 5.7 é possível comprovar o paralelismo do CC. No Diagrama de Seqüencia, cada uma das barras verticais representa um sub-processo. A ativação de cada sub-processo se dá através de eventos, os quais são representados por setas. O tempo de processamento é indicado pelos trechos largos em cada uma das barras verticais. Os trechos pontilhados indicam que o processo está inativo, aguardando algum evento. A Figura 5.7 mostra claramente o paralelismo permitido pelo CC. O aumento de paralelismo se dá pela independência de duas duplas de processos (A, B) e (C, D), definidas na Tabela 5.6. Enquanto C e D realizam a cifração de forma rápida e simples, os processos A e B preparam nova tabela para ser utilizada pelo processo C. O gargalo de processamento reduz-se a uma busca linear dentro de Γ, tendo em média dois testes até encontrar o resultado procurado.

Existe uma outra forma de paralelismo que pode ser aplicada ao CC. Devido a suas características, é possível dividir o texto plano em diversas partes e cifrar cada uma delas paralelamente e de forma independente. A utilização deste método de cifração consiste em dividir a tabela Γ em diversas partes iguais. Cada parte de Γ é usada para cifrar um trecho do texto plano. Como exemplo simples de aplicação, o texto plano $P = \{b, a, c, a, d, a\}$ é cifrado em 3 núcleos de processamento que utilizam a tabela Γ representada na Figura 5.8. O texto plano e a tabela Γ são divididos em 3 partes. A cada uma delas, é atribuído um processo independente, digamos A, B e C. Os processos iniciam em sub-partes diferentes de Γ, neste exemplo os pontos de início são $P_A = 2$, $P_B = 17$ e $P_C = 55$. Os valores de P_A, P_B e P_C foram escolhidos neste exemplo para fins de entendimento do processo.

O resultado do processamento paralelo é $C = C^1 =$ (7-2=5, 16-7=9) $+ C^2 =$ (36-17=19, 42-36=6) $+ C^3 =$ (71-55=16, 77-71=6). A decifração utiliza 2 bytes para descrever cada símbolo plano. O primeiro byte corresponde a quantidade de vezes que o número de iterações ultrapassou o valor limite de 8 bits, ou seja, 255 e o segundo corresponde ao restante das iterações (inferiores ao valor de 255). Portanto, $C = (0, 5, 0, 9, 0, 19, 0, 6, 0, 16, 0, 6)$. A decifração também pode ser paralelizável, seguindo o mesmo procedimento da decifração. Este exemplo divide o paralelismo em três processos, mas ele pode ser dividido em tantos quantos o *hardware* suportar.

Nenhum dos cifradores simétricos de fluxo estudados neste trabalho (Sosemanuk, RC4), têm a capacidade de paralelizar seus processos de cifração de forma tão granular como o CC tem. Por serem cifradores de fluxo, o novo estado de cifração depende do

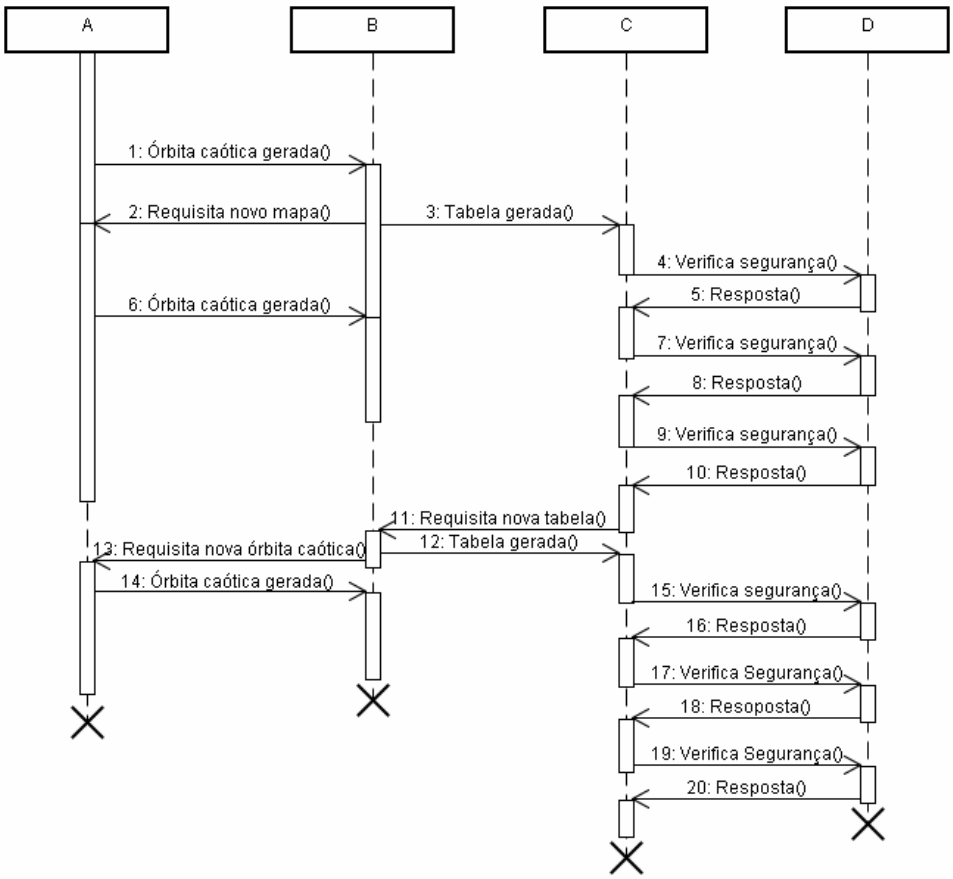

FIGURA 5.7 – Diagrama de Seqüencia do algoritmo CC.

estado anterior e, portanto, a cifração do texto plano não pode ser dividida em processos independentes. A única possibilidade disso ocorrer é se o texto plano for dividido em várias partes e cada uma delas for tratada como textos planos isolados. Sendo assim cada uma das partes do texto plano seria cifrada independentemente da outra. Isso implicaria no uso de chaves ou IVs diferentes para cada cifração, principalmente no caso do RC4 que não pode repetir o mesmo IV mais do que uma vez com a mesma chave. Ao final, um processo de concatenação dos textos cifrados deveria ser utilizado. O mesmo ocorre com o AES que, por usar um Modo de Operação, atrela o estado atual de cifração aos estados anteriores. No caso do AES, se eliminarmos o Modo de Operação, ele consegue paralelizar a operação de cifração com a mesma intensidade do CC. O AES é

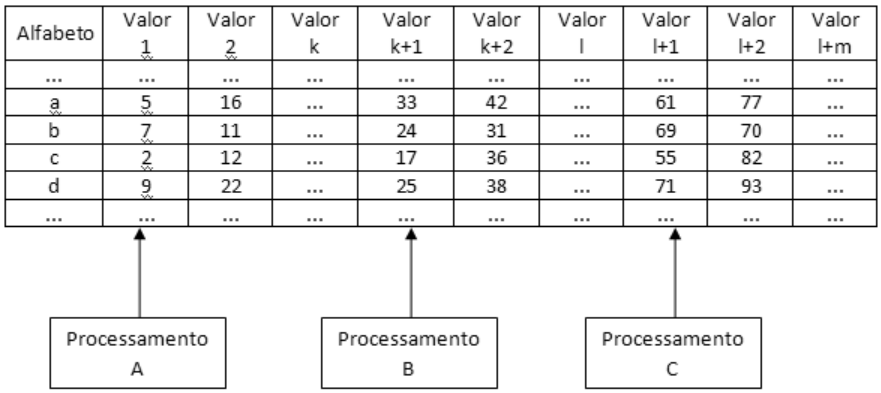

FIGURA 5.8 – Variação do método de procura dentro tabela Γ do cifrador CC, particionando Γ para processamento paralelo de textos planos.

um cifrador de blocos, conforme descrito na Seção 2.2, é muito recomendado combinar um Modo de Operação à cifradores de bloco. No caso do AES, quando ele cifra dados sem utilizar um Modo de Operação, gerar blocos cifrados idênticos para os mesmos blocos de textos planos. Portanto, sem um modo como CBC, CFB, OFB, ou outros mais modernos, sua segurança fica comprometida.

Para o CC, a cifração ocorre com a mesma chave e o algoritmo avança no domínio do mapa caótico praticamente da mesma forma que se estivesse avançando como um único processo. Certamente existem variações que podem ser utilizadas, com uso de diferentes chaves e/ou otimização do consumo dos d-blocos de Γ. No caso de termos diferentes chaves para a cifração, o CC se comportaria da mesma maneira descrita para os algoritmos simétricos de fluxo estudados. O texto plano seria dividido em partes e vários processos independentes cifrariam os sub-textos planos. Ao final um processo de concatenação seria aplicado.

É possível otimizar o uso de valores de Γ. Conforme mostrado na Figura 5.9, todos os processos iniciariam a cifração no início da tabela Γ. Seguindo esse procedimento, devemos levar em conta que o ponto de início de todos os processos é o menor valor dentro da primeira coluna de Γ. Assumindo o valor 2 como o valor de início de cifração, o texto plano $P = (b, a, c, a, d, a)$, corresponderia ao texto cifrado $C = (0, 5, 0, 9, 0, 10, 0, 4, 0, 7, 0, 7)$.

Esse procedimento tem o viés de tornar o processo de cifração menos seguro,

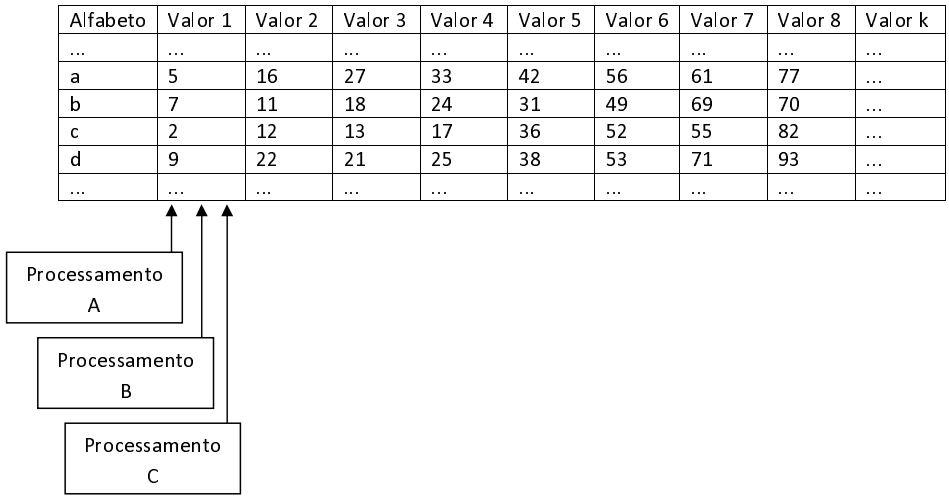

FIGURA 5.9 – Variação do método de procura dentro tabela Γ do cifrador CC, sem particionamento de Γ para processamento paralelo de textos planos.

visto que a forma de cifrar os dados foi modificada e portanto, a entropia máxima do sistema já não é mais garantida.

Adaptabilidade a diferentes ambientes

A adaptabilidade corresponde a todos os fatores nos quais o cifrador pode ser regulado para atuar em diferentes ambientes. No caso do CC, existem três características importantes que são ajustadas conforme a situação onde ele é aplicado. A primeira é a ϵ-segurança, identificada pelo expoente de lyapunov e já abordada em 4.1. Esta característica permite regular a sensibilidade do sistema caótico. Quanto mais sensível o sistema, mais incerteza é injetada nele e portanto, maior a sua segurança. O aumento da ϵ-segurança traz consigo também o aumento do tempo de cifração. O gráfico da Figura 5.10 mostra o comportamento do CC a medida que aumentam o expoente de Lyapunov e o tamanho dos d-blocos. Como esperado, quando a restrição da ϵ-segurança $l(x) \geq l_0$ é usada, em tempo de execução, a eficiência do CC diminui. As equações (10) e (11) definidas em [7] dão uma idéia da dependência do tempo de cifração em função de l_0. Na prática, o tempo de cifração dá um salto quando l_0 deixa de ser zero. Sendo l_0>0,

a medida que ele aumenta, o tempo de cifração cresce gradativamente. Entre $l_0 = 0.1$ e seu valor máximo, $l_0 = 1$, o tempo de cifração aumenta em $26,66\%$. É importante observar que o uso de $l_0 = 0$ não torna o CC inseguro ou passível de algum ataque conhecido à cifradores caóticos e análise estatística. O aumento de l_0 apenas torna mais imprevisível o comportamento do mapa caótico, o que aumenta mais a incerteza do sistema caótico, tornando o CC ainda mais seguro.

Nota-se também que o desempenho do CC é praticamente independe do tamanho dos d-blocos. Por exemplo, quando $d = 1$, ou seja, d-blocos formados de um dígito, o maior alfabeto possível é $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. O tempo de cifração com $\mathcal{A}$ e d = 1, 2, 3,... é o mesmo que o tempo de cifração para um alfabeto de 256 símbolos e com d = 3, 4, 5,....

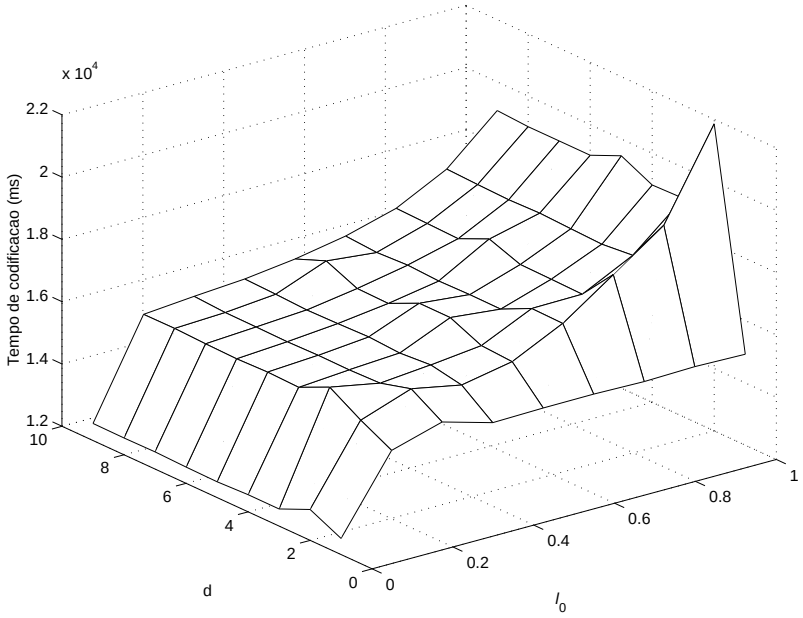

FIGURA 5.10 – Gráfico que mostra a variação no tempo de codificação (milisegundos), em função dos tamanhos dos d-blocos (d) e do expoente de lyapunov (l_0), para um texto de 10MB de tamanho.

O experimento mostrado na Figura 5.10 foi realizado no sistema operacional Linux em modo de segurança e sem interface gráfica. Os tempos colhidos são resultado da média de cifração de 10 textos não viesados, compostos pelos alfabeto ASCII e com tamanho de 10MB.

A segunda característica passível de regulagem é o tamanho da tabela Γ. A tabela pode assumir qualquer valor que o ambiente comportar. Uma tabela grande possibilita maior velocidade de cifração, a medida que o cálculo de novos mapas caóticos é menos freqüente. Seu viés é o maior consumo de memória. Por outro lado, em ambientes com memória restrita, pode-se reduzir o tamanho de Γ. O custo de usar o mínimo de memória possível é o aumento do processamento. Tabelas pequenas são rapidamente utilizadas no processo de cifração e portanto, novos dígitos de $\sqrt[p]{x_0}$ têm de ser introduzidos em Γ às custas de novos cálculos, com maior freqüência. Tal possibilidade de maior ou menor uso de memória e, maior ou menor uso de processamento, se reflete na adaptabilidade do algoritmo a diversos ambientes e suas restrições.

Por fim, a terceira característica regulável no CC é o número de vezes que a mesma chave é utilizada em diferentes cifrações. Esta característica influencia diretamente a velocidade de cifração e a segurança. Lembra-se novamente que o uso repetido do mesmo mapa caótico reduz a taxa de geração de novos mapas a medida que novos textos planos são cifrados. É possível que o uso demasiadamente elevado do mesmo mapa caótico para cifrar diferentes textos planos seja passível de exploração estatística por parte de criptoanalistas. Não existe um número exato de quantos textos planos podem ser cifrados para o mesmo mapa caótico. Essa medida depende do tamanho do mapa caótico e do tamanho dos textos planos.

5.4 Limitações do CC

O CC é um algoritmo que contempla muitas das qualidades desejáveis em criptografia. Mesmo assim, a exemplo de todos os outros cifradores existentes, ele tem limitações. Uma das principais limitações é quanto ao tamanho do texto gerado. Como dito na Seção 4.3, mesmo com as melhorias nesta área, o texto cifrado continua com tamanho maior do que o usual da maioria dos cifradores modernos. Esse é um problema que não restringe o uso do CC, mas merece atenção pois, talvez, limite os ambientes de aplicabilidade do CC. Outro fator que limita o nicho de atuação do CC é o ataque por exploração da memória usada. Essa memória é utilizada principalmente pela Tabela de Endereçamentos. Portanto, atualmente, o uso do CC não é recomendado em computadores pessoais com fácil acesso de pessoas suspeitas. Por fim, a velocidade

de cifração do CC deve ser melhorada. Entre os cifradores modernos, o CC tem tempos de cifração próximos ao usual, mesmo assim, não tem os melhores tempos de cifração. Esse é um dos fatores que tem ampla chance de ser melhorado, pois a implementação do cifrador não é a mais otimizada possível e abre espaço para que através de implementação de código se atinja velocidades até superiores aos mais rápidos cifradores modernos.

Todas as características limitantes do CC aqui apresentadas são evidentemente fatores que restringem os ambientes uso do CC, porém estão longe de serem características que inviabilizem seu uso. Lembrando que o CC é um algoritmo que ainda está em estudo e pode ser melhorado e suas limitações minimizadas.

5.5 Resumo dos resultados

Como resumo prático, as Tabelas 5.7, 5.8 e 5.9 condensam o comparativo de todos os quesitos abordados neste capítulo. Elas abordam prós e contras do CC e faz algumas comparações em relação aos outros algoritmos estudados neste trabalho. A Tabela 5.7 compara o desempenho do CC em relação ao AES, RC4 e Sosemanuk. A Tabela 5.8 aborda aspectos de segurança. Por fim, a Tabela 5.9 aborda características outras, que não segurança e velocidade do CC.

TABELA 5.7 – Resumo comparativo de desempenho do CC em relação aos outros algoritmos estudados.

Desempenho				
Aspectos	**CC**	**RC4**	**Sosemanuk**	**AES**
Velocidade de Cifração em Software	Ótimo desempenho se comparado a cifradores caóticos, ver Seção 5.1, perde para cifradores tradicionais.	Desempenho inferior aos outros cifradores tradicionais.	Tem os melhores resultados.	Próximo ao melhor resultado.
Velocidade de Decifração em Software	Boa, inferior apenas ao Sosemanuk.	Inferior a todos os outros cifradores.	Tem os melhores resultados.	Desempenho razoavelmente bom.
Velocidade de Cifração em Hardware	Não analisado	Não analisado.	Não analisado.	Não analisado.

TABELA 5.8 – Resumo comparativo da segurança do CC em relação aos outros algoritmos estudados.

Segurança				
Aspectos	**CC**	**RC4**	**Sosemanuk**	**AES**
Ataques a cifradores caóticos	diferentemente de outros cifradores caóticos, o CC é resistente aos ataques caóticos conhecidos.	não aplicado	não aplicado	não aplicado
Ataques tradicionais	Para os ataques abordados nesse trabalho, o CC passou em todos os testes aplicados.	Existem diversos ataques conhecidos.	Ao menos um trabalho propõe um ataque contra chaves de 256 bits [58].	Alguns ataques propostos, mas todos sem uso prático ou inviáveis.
Tempo de mercado	Não tem tempo de mercado. Ataques eficazes podem vir a surgir.	Muito tempo de mercado. Sua segurança foi muito bem explorada.	Pouco tempo de mercado. Outros ataques podem vir a surgir.	Muito tempo de mercado. Sua segurança foi muito bem explorada.

TABELA 5.9 – Resumo comparativo das características do CC em relação aos outros algoritmos estudados.

Aplicações e uso				
Aspectos	**CC**	**RC4**	**Sosemanuk**	**AES**
Adaptabilidade	Regulável em espaço de memória, segurança e velocidade. Texto cifrado pode ficar entre 60% 100% maior que o texto plano.	tamanho do vetor de inicialização regulável para aumentar segurança.	sem regulagem de espaço de memória. Número de rodadas variável para cifrar dados, o que altera segurança e velocidade.	número de rodadas regulável, afetando segurança e velocidade.
Uso de memória	Adaptável conforme ambiente	Uso de pouca memória	Uso de pouca memória.	Uso de pouca memória.
Portabilidade para várias plataformas	Sim.	Sim.	Sim.	Sim.
Tamanho variável das chaves	Não se aplica.	$1 \leq$ tamanho da chave ≤ 2048 bits.	$128 \leq$ tamanho da chave ≤ 256 bits. A segurança é garantida apenas para chaves de 128 bits.	128 bits, 192 bits ou 256 bits.
Simplicidade de implementação	Fácil implementação em software e *hardware*.	Muito fácil implementação em software e *hardware*.	Dificuldade média de implementação em software e *hardware*.	Dificuldade média de implementação em software e *hardware*.
Portabilidade para software e *hardware*	Sim.	Sim.	Sim.	Sim.
Paralelismo	Paralelizável.	Paralelizável.	Paralelizável.	Paralelizável.

Capítulo 6

Conclusão

A troca e armazenamento de dados via meios eletrônicos vem aumenta ao ritmo da evolução tecnológica. A diversidade de equipamentos e a conectividade cada vez maior entre eles incentiva este hábito. Muitos desses dados são informações sigilosas, as quais necessitam de proteção contra o acesso de terceiros. A criptografia é uma das ferramentas utilizadas para prover sigilo à esses dados. Ela pode ser combinada com outras tecnologias, ou utilizada isoladamente. Um dos instrumentos necessários para o uso de criptografia são o uso de algoritmos criptográficos. A exemplo de outros ramos da criptografia, eles também estão em constante evolução para atender o aumento das trocas seguras de informações via meios eletrônicos. Prova disso foi o esforço do NIST que, após décadas utilizando e recomendando o uso do cifrador DES, promoveu um concurso para selecionar um novo cifrador simétrico de blocos. Esse cifrador, denominado AES, veio para atender as novas exigências do mundo atual em termos de cifração simétrica de dados. Com o mesmo intuito, mais recentemente, um convênio de empresas, instituições e universidades chamado *ECRYPT*, vem focando esforços para selecionar um bom cifrador simétrico de fluxo. As características buscadas para cifradores modernos são basicamente segurança, velocidade, baixo consumo memória, uso de pouco processamento e adaptabilidade a diferentes ambientes de cifração. A busca pela melhor combinação dessas características levou a especulação dos mais diversos métodos de cifração. Um dos métodos que vem sendo pesquisado é o uso de mapas caóticos para cifrar dados.

Este trabalho investigou cifradores caóticos baseados no modelo proposto por

Baptista, mais especificamente, estudou o cifrador nomeado por CC. Para tanto, foi feita uma revisão sobre criptografia, teoria do caos e o uso de mapas caóticos na cifração de dados. Este estudo envolveu a análise das características buscadas pelo NIST e pelo *ECRYPT*, comparou também, o desempenho do CC com outros cifradores modernamente usados: AES, RC4 e Sosemanuk. Este trabalho também propôs a inclusão de melhorias no CC. Essas melhorias visam tornar o CC mais competitivo e próximo aos cifradores utilizado atualmente.

A primeira melhoria proposta foi na geração do texto cifrado. No algoritmo original, o texto cifrado tem o dobro do tamanho do texto plano. A modificação proposta na forma de armazenar o texto cifrado reduziu ele para 1.6 vezes do tamanho do texto plano. Em comparação aos cifradores modernos, este tamanho ainda não é o ideal porém, é uma redução considerável, se comparada ao tamanho do texto cifrado do algoritmo original.

Outra importante melhoria foi a inclusão da tabela de endereçamentos. O uso dessa tabela trouxe, como melhoria imediata, um aumento considerável na velocidade de cifração. Além disso, no processo anterior, o desempenho de cifração era diretamente proporcional ao tamanho do alfabeto cifrado. Alfabetos de 256 símbolos tinham, em média, 256 iterações antes de cifrar um símbolo. Da mesma forma, alfabetos de 10 símbolos realizavam, em média, 10 iterações e assim por diante. Atualmente, realiza-se 2 iterações, em média, para cifrar cada símbolo. A taxa de cifração se tornou constante e independe do tamanho do alfabeto. A velocidade de cifração se aproximou muito mais dos cifradores comercialmente utilizados. A inclusão da tabela de endereçamentos trouxe ainda outras vantagens. O tamanho dessa tabela é fixo em linhas, pois depende somente do tamanho do alfabeto, e variável em colunas. Essa variabilidade permite regular o uso de memória do CC de forma muito mais efetiva. Devido à independência do alfabeto para cifrar os dados e, a baixa taxa de iterações necessárias para cifrar cada dado, tem-se um controle muito mais fino de quais serão as possíveis posições de memória necessárias para cifrar os próximos dados. Isso permite estimar e alocar de forma eficaz a memória que vai conter as órbitas do mapa caótico, conforme a disponibilidade do ambiente onde ele é executado. O uso da memória também é mais inteligente, visto que, como as requisições são em ordem e, geralmente, com os mesmos saltos no endereçamento requisitado, é sabido com muito mais precisão qual a próxima área de memória será requisitada pelo cifrador. Essas características

reduzem as falhas de paginação por parte do sistema operacional, que utiliza algoritmos específicos como FIFO, LRU, entre outros para controlar o uso de memória [59]. Isto é, a medida que os dados requisitados pelo CC estejam com muito mais freqüência disponíveis, primeiramente, em memória Cache, reduz-se os acessos desnecessários a memória RAM e principalmente a disco rígido. Com isso, a taxa de acesso a dados é muito mais efetiva do que no processo antigo.

Outra melhoria proposta para o CC é a paralelização de seu processo de cifração e decifração. Foi mostrado que o CC pode ser dividido em 4 sub-processos simples, que podem executar paralelamente, maximizando o uso do processador. Outra forma de paralelismo proposta é quanto a cifração de diferentes partes do texto de forma paralela. Neste caso foram sugeridas duas formas de implementação e relatadas suas vantagens e desvantagens entre si e em relação ao método original. O processamento paralelo de diversas partes do texto plano é particularmente melhor aproveitado quando o CC é implantado em *hardware*. Múltiplos núcleos do cifrador podem ser instanciados, todos compartilhando a mesma tabela de endereçamento e os mesmos textos planos. A redução no tempo de cifração e decifração é diretamente proporcional a quantidade de núcleos de processamento instanciados.

As melhorias até o momento apresentadas afetam principalmente a adaptabilidade do CC a diversos ambientes e a sua velocidade de processamento. Em termos de segurança, a contribuição deste trabalho para o CC é o uso de um mecanismo que torna o cifrador estatístico, isto é, independente dos textos planos e da chave utilizada, o CC nunca gera o mesmo texto cifrado. O funcionamento desse mecanismo é baseado na propriedade do CC de eqüiprobabilidade na visitação de sítios do domínio do mapa utilizado. O adendo dessa característica não afeta a forma de cifrar os dados e tão pouco a velocidade de cifração e decifração. Esse mecanismo é de implementação simples, mas traz benefícios fundamentais em termos de segurança, sendo decisivo contra, pelo menos, um ataque proposto a cifradores caóticos baseados no modelo de Baptista e, no mínimo, dificulta outras formas de criptoanálise, inclusive ataques tradicionalmente conhecidos à cifradores comerciais.

Além das melhorias apresentadas acima, o CC tem em sua essência outras características como controle do grau de segurança, independência de máquina, fácil implementação, entropia máxima e espaço de chaves ilimitado. Ao contrário do CC, todos os cifradores comerciais têm um espaço de chaves grande, porém limitado. Além

disso, o tamanho das chaves nesses cifradores têm influencia direta na segurança e na velocidade, enquanto que no CC esses fatores são invariantes para qualquer chave utilizada.

Para avaliar a aplicação prática do CC, foram realizados diversos testes de desempenho e segurança. Todos os testes foram realizados em ambiente controlado, com diversas amostras. Os testes de desempenho foram aplicados também aos cifradores AES, RC4 e Sosemanuk, com o objetivo de comparar o CC a cifradores comerciais utilizados atualmente.

Embora não tenha atingido o melhor desempenho, na cifração, o CC atingiu níveis de desempenho próximos aos outros. Enquanto que na decifração, ele superou dois dos três cifradores comparados a ele, mesmo com uma implementação de código não totalmente otimizada e sem os recursos de paralelismo propostos. Isso leva a crer que, implementações mais eficazes do CC e outros cifradores caóticos nos moldes do CC, podem atingir os mesmos padrões de desempenho alcançados pelo AES, RC4 e Sosemanuk, inclusive com a possibilidade de superá-los em termos de velocidade de cifração e decifração.

Quanto à segurança, foram realizados diversos testes empíricos para medir o quanto o CC deixa transparecer os textos planos. Em todos os testes o CC apresentou resultados satisfatórios quanto à segurança, o que vem de encontro com a teoria que define o cerne do CC, a qual afirma que ele tem máxima entropia. As análises quanto a segurança no uso da chave e contra a criptoanálise do CC indicam que o CC é, possivelmente, um cifrador seguro para os dias atuais, principalmente depois de adicionada a propriedade de cifrador estatístico. É claro que, no quesito segurança, testes empíricos e uma análise de segurança não são suficientes para provar que ele é totalmente seguro. São necessárias outras avaliações da comunidade científica e até de empresas de segurança e órgãos de segurança para comprovar que não existem ataques eficientes contra o CC. Esse tipo de análise mais profunda vem com o tempo e com o uso do CC, a exemplo do que ocorre com qualquer outro cifrador.

Em se tratando de uso prático de algoritmos criptográficos, também foi analisada a adaptabilidade do CC a diferentes ambientes e, quando possível, comparando as características do CC ao AES, RC4 e Sosemanuk. Nesse aspecto o CC apresentou bons resultados. Ele contemplou bem os diversos aspectos analisados, ficando no mesmo nível dos cifradores AES, RC4 e Sosemanuk.

Diante de tudo o que foi apresentado, conclui-se que cifradores caóticos que utilizam metodologia similares ao CC podem ter uso prático. Em especial, o CC é bastante promissor para uso comercial. Acredita-se que o CC pode ter seu nicho de mercado em equipamentos que evitam o acesso a sua memória, como por exemplo os HSM. E, seu uso, especialmente indicado a ambientes que necessitem rápida decifração, onde o tamanho dos textos cifrados não é problema, ou seja, onde o canal de dados tem sobras de recursos em relação a banda requerida ou, no caso de cópias de segurança de dados, em áreas de armazenamento que suportem a capacidade mínima de armazenar 1.6 vezes mais o tamanho dos textos planos.

Por fim, trabalhos futuros apontam para o aprofundamento do estudo da segurança de cifradores caóticos similares ao CC, principalmente quanto ao acesso a tabela de endereçamento; pesquisa de maneiras de reduzir ainda mais o tamanho dos textos cifrados; geração de novas chaves de cifração de forma automática; aperfeiçoamento do código gerado para o CC, extraindo o máximo do *hardware* em que esteja executando; pesquisa de formas de transformar o CC em um cifrados de chave pública e por fim, implementar o CC em *hardware* para estudar mais a fundo seu desempenho em um *hardware* dedicado.

Glossário

d-blocos	Quantidade de dígitos necessários para definir um sítio do mapa caótico. Esta quantidade de dígitos está relacionada ao tamanho do alfabeto. Um alfabeto de 256 símbolos necessita, no mínimo, 3 dígitos para relacionar cada sítio do mapa caótico com um símbolo diferente, ou seja, d-bloco=3 . Já um alfabeto de 40 símbolos precisa, no mínimo, d-blocos=2, e assim por diante., 1
Chave	No escopo deste trabalho, quando a chave de cifração é igual a chave de decifração, utiliza-se apenas o termo (chave) [10]., 1
Chave de cifração	Chave de entrada de um cifrador que, combinada com o texto plano, gera o texto cifrado [10]., 1
Chave de decifração	Chave de entrada de um decifrador que, combinada com o texto cifrado, recupera o texto plano [10]., 1
Cifrador	Algoritmo que tem a função de proteger informações sigilosas, alterando os símbolos que a compõe, sem perda de informação [10]., 1
Cifrar	Transformar o texto plano em texto cifrado., 1
Codificar	No escopo deste trabalho, é sinônimo de cifrar., 1

Criptoanálise Ramo da criptologia que estuda os métodos de recuperar informações de textos cifrados sem o conhecimento da chave [10]., 1

Criptografia Ramo da criptologia que estuda os métodos para restringir acesso a informações sigilosas utilizando cifradores [10]., 1

Criptologia É a ciência que estuda as técnicas para restringir e obter acesso a informações sigilosas [10]., 1

Decifrador Algoritmo que tem a função de recuperar informações sigilosas protegidas por um cifrador, utilizando a chave de decifração [10]., 1

Decifrar Recuperar do texto cifrado seu respectivo texto plano., 1

Decodificar No escopo deste trabalho, é sinônimo de decifrar., 1

Decriptar Sinônimo de decifrar., 1

Desempenho No escopo deste trabalho, significa a velocidade de cifração e decifração de dados., 1

Emissor Quem envia a mensagem cifrada., 1

Encriptar Sinônimo de cifrar., 1

HSM Equipamento especializado em criptografia que armazena e gera chaves, criptografa e decriptografa dados. Ele pode ainda, rotear pacotes cifrados por um algoritmo para outro canais de dados, que utilizam algoritmos de criptografia diferentes., 1

Implementar No escopo deste trabalho, implementar significa encontrar uma forma de codificar um algoritmo teórico para um computador de forma a extrair a máxima velocidade com mínimo uso de memória mantendo a máxima segurança possível do algorítmo teórico., 1

Primitivas Funções algorítmicas simples utilizadas de forma isolada ou combinadas para compor um cifrador [12]., 1

Receptor Destinatário da mensagem cifrada., 1

Sistema criptográfico É o conjunto composto pelo cifrador, decifrador e as respectivas chaves de cifração e decifração., 1

Sistema de Segurança Conjunto de equipamentos, dispositivos, sistema criptográfico e regras que fornecem segurança à informações sigilosas, restringindo o acesso a elas em um ambiente [1]., 1

Texto cifrado Todo tipo de dado (texto, imagem, som, arquivo, etc) modificado por um cifrador [10]., 1

Texto plano Todo tipo de dado (texto, imagem, som, arquivo, etc) original, o qual não foi cifrado [10]. Neste trabalho é adotada a nomenclatura Texto Plano (P) ao invés de Texto Claro para facilitar a simbologia e não confundindo com Texto Cifrado (C)., 1

UML	Linguagem gráfica, não proprietária e padrão para a elaboração da estrutura de projetos de software., 1

Bibliografia

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Prentice-Hall, 2 ed., 2002.

[2] A. Berent, "Aes (advanced encryption standard) simplified," tech. rep., ABI Software Development, 2003.

[3] E. Moreno, D. Pereira, and R. Chiaramonte, *Criptografia em Software and Hardware*. Novatec, 2005.

[4] NIST, "Advanced encryption standard (aes)," tech. rep., National Institute of Standards and Technology NIST. Federal Information Processing Standards Publication 197.

[5] C. B. et al., "Sosemanuk, a fast software-oriented stream cipher." Disponível em http://www.ecrypt.eu.org/stream/sosemanukp3.html, acessado em abril de 2008, 2004.

[6] M. S. Baptista, "Cryptography with chaos," *Physics Letters A*, no. 240, pp. 50–54, 1998.

[7] L. P. L. Oliveira and M. Sobottka, "Cryptography with chaotic mixing," *Chaos Solitons and Fractals*, May 2008.

[8] G. Álvarez, F. Montoya, M. Romera, and G. Pastor, "Cryptanalysis of an ergotic chaotic cipher," *Physics Letters A*, no. 311, pp. 172–179, 2003.

[9] S. Li, X. Mou, Z. Ji, J. Zhang, and Y. Cai, "Performance analysis of jakimoski-kocarev attack on a class of chaotic cryptosystems," *Physics Letters A*, vol. 307, no. 1, pp. 22–28, 2003.

[10] B. Schneier, *Applied Cryptography*. John Wiley and Sons, 2 ed., 1996.

[11] D. Khan, *The Codebreakers: The History of Secret Writing*. McMillan Publishing Co, 1967.

[12] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*. Boca Raton, EUA: CRC Press, Inc., 1996.

[13] C. Shannon, "Communication theory of secrecy systems," *Bells Systems Technical Journal*, no. 4, 1949.

[14] S. Paine, *Criptografia e Segurança - O Guia oficial RSA*. Campus, 2002.

[15] J. N. et al., "Report on the development of the advanced encryption standard (aes)," *Journal of Research of the National Institute of Standards and Technology*, vol. 106, 2001.

[16] M. Hermelin and K. Nyberg, "Correlation properties of bluetooth combiner generator," *2th International Conference on Information Security and Cryptology*, pp. 17–29, 2000.

[17] L. Pecora and T. Carroll, "Synchronization in chaotic systems," *Phys. Rev. Lett.*, vol. 64, pp. 821–824, Feb 1990.

[18] J. Fridrich, "Symmetric ciphers based om two-dimensional chaotic maps," *Int. J. of Bifurcation and Chaos*, vol. 8, no. 6, pp. 1259–1284, 1998.

[19] U. Feldmann, M. Hasler, and W. Schwarz, "Communication by chaotic signals: The inverse system approach," *Int. J. Circuit Theory Appl.*, no. 5, pp. 551–579, 1996.

[20] G. Alvarez, G. Monotoya, and M. Romera, "Chaotic cryptosystems," *Proc. IEEE 33rd Annual Int. Carnahan Conf. Security Technology*, pp. 332–338, 1999.

[21] E. Biham, “Cryptanalysis of the chaotic-map cryptosystem suggested at EUROCRYPT ’91,” *Lecture Notes in Computer Science*, vol. 547, pp. 532–534, 1991.

[22] D. Wheeler and R. Matthews, “Supercomputer investigations of a chaotic encryption algorithm,” *Cryptologia*, vol. XV, no. 2, pp. 140–152, 1991.

[23] S. L. et al., “Problems with a probabilistic encryption scheme based on chaotic systems.” accepted by Int. J. Bifurcation and Chaos in July 2002, initially scheduled to be published in vol. 13, no. 10, 2003, preprint available online at http://www.hooklee.com/pub.html, 2003.

[24] G. Jakimoski and L. Kocarev, “Chaos and cryptography: block encryption ciphers based on chaotic maps,” *IEEE Trans. Circuits Systems I Fund. Theory Appl.*, vol. 48, no. 2, pp. 163–169, 2001.

[25] W. Wonga, L. Lee, and K. Wong, “A modified chaotic cryptographic method,” *Computer Physics Communications*, no. 138, pp. 234–236, 2001.

[26] M. Sobottka and L. P. L. Oliveira, “Dynamic properties of an exact algorithm for square root calculation,” *Phisica D*, Sep 2006.

[27] J. Daemen and V. Rijmen, “The rijndael block cipher.” Disponível em http://www.nist.gov/aes, acessado em janeiro de 2008, 1999.

[28] ECRYPT, “estream.” Disponível em http://www.ecrypt.eu.org/stream/, acessado em abril de 2008.

[29] A. F. Webster and S. E. Tavares, “On the design of S-boxes,” pp. 523–534, 1986.

[30] C. Adams and S. Tavares, “The structured design of cryptographically good s-boxes,” *J. Cryptol.*, vol. 3, no. 1, pp. 27–41, 1990.

[31] K. Nyberg, “Perfect nonlinear s-boxes,” *Advances in Cryptology - EUROCRYPT 1991.*, pp. 378–386, 1991.

[32] M. Robshaw, “Stream ciphers,” tech. rep., RSA Laboratories Technical Report TR-701, 1995.

[33] E. Biham and A. Shamir, *Differential cryptanalysis of the data encryption standard*. London, UK: Springer-Verlag, 1993.

[34] A. Biryukov and D. Wagner, “Slide attacks,” *Proceedings of Fast Software Encryption FSE 1999*, no. 1636, pp. 245–259, 1999.

[35] A. Biryukov and D. Wagner, “Advanced slide attacks,” *Advances in Cryptology EUROCRYPT 2000*, no. 1807, pp. 589–606, 2000.

[36] M. Matsumi, “Linear cryptanalysis method for des cipher,” *Advances in Cryptology - EUROCRYPT 1993*, pp. 386–397, 1993.

[37] J. D. Golic, “Correlation properties of a general binary combiner with memory,” *J. Cryptology*, vol. 9, no. 2, pp. 111–126, 1996.

[38] J. Daemen and V. Rijmen, “Aes proposal: Rijndael.” Disponível em http://citeseer.csail.mit.edu/daemen98aes.html, acessado em janeiro de 2008, 1998.

[39] A. M. Muñoz, “Seguridad europea para eeuu, algoritmo criptográfico rijndael.” Disponível em http://www.kriptopolis.org/docs/rijndael.pdf, acessado em abril de 2008, 2004.

[40] R. Anderson, E. Biham, and L. Knudsen, “Serpent: A proposal for the advanced encryption standard.” Disponível em http://www.cl.cam.ac.uk/, acessado em abril de 2008.

[41] P. Souradyuti and B. Preneel, “Analysis of non-fortuitous predictive states of the rc4 keystream generator.” Avaliable in http://www.cosic.esat.kuleuven.be/publications/article-86.pdf, last access 2008, June.

[42] L. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, “Analysis methods for (alleged) rc4,” pp. 327–341, 1998.

[43] I. Mantin and A. Shamir, “A practical attack on broadcast rc4,” pp. 152–164, 2001.

[44] S. R. Fluhrer and D. A. McGrew, “Statistical analysis of the alleged rc4 keystream generator,” pp. 19–30, 2001.

[45] K. Alligood, T. D. Sauer, and J. A. Yorke, *Chaos an Introduction to Dynamical Systems*. Addison-Wesley, 1996.

[46] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems, 2nd Edition*. Westview Pr (Short Disc), March 2003.

[47] S. Lian, J. Sun, and Z. Wang, “Security analysis of a chaos-based image encryption algorithm.” Disponível em http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0608119, acessado em dezembro de 2007.

[48] S. Li, G. Chen, K. Wong, X. Mou, and Y. Cai, “Baptista-type chaotic cryptosystems: problems and countermeasures,” *Physics Letters A*, no. 332, pp. 368–375, 2004.

[49] S. Li, X. Mou, Y. Cai, Z.Ji, and J. Zhang, “On the security of a chaotic encryption scheme: problems with computerized chaos in finite computing precision,” *Computer Physics Communications*, no. 153, pp. 52–58, 2003.

[50] S. Behnia, N. A. Akhshani, A. Akhavan, and H. Mahmodi, “Chaotic cryptographic scheme based on composition maps.” Disponível em http://www.citebase.org/abstract?id=oai:arXiv.org:nlin/0601051, acessado em janeiro de 2008.

[51] Y. C. Bo Mi, Xiaofeng Liao, “A novel chaotic encryption scheme based on arithmetic coding,” no. 38, 2009.

[52] B. D, W. Dei, and et al., “Cryptographic c++ class library crypto++.” Disponível em www.cryptopp.com, acessado em outubro de 2007.

[53] B. Schneier and D. Whiting, "A performance comparison of the five aes finalists.," in *AES Candidate Conference*, pp. 123–135, 2000.

[54] R. Venugopalan, "Encryption overhead in embedded systems and sensor network nodes: modeling and analysis," *International Symposium on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 188–197, 2003.

[55] NIST, "Statistical test suit: Nist special publication 800-22." Disponível em http://csrc.nist.gov/groups/ST/toolkit/index.html, acessado em maio de 2008, 2001.

[56] A. Lee, "Guideline for implementing cryptography in the federal government," tech. rep., National Institute of Standards and Technology, 2000.

[57] J. Nechvatal and et al., "Report on the development of the advanced encryption standard (aes)," tech. rep., Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce, 2000.

[58] H. Ahmadi, T. Eghlidos, and S. Khazaei, "Improved guess and determine attack on sosemanuk," tech. rep., eSTREAM, ECRYPT Stream Cipher Project.

[59] A. Silberschatz and P. B. Galvin, *Operating System Concepts*. John Wiley & Sons, Inc., fifth edition ed., 1999.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo