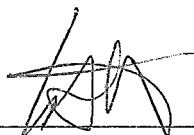


O RESFRIAMENTO SIMULADO NO PROJETO ÓTIMO DE AUTÔMATOS  
CELULARES PARA A GERAÇÃO DE CHAVES EM CRIPTOGRAFIA DE FLUXO

Saulo Moraes Villela

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS  
PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE  
FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS  
NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM  
ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:



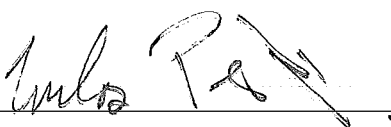
---

Prof. Luís Alfredo Vidal de Carvalho, D.Sc.



---

Prof. Luís Felipe Magalhães de Moraes, Ph.D.



---

Prof. Carlos Eduardo Pedreira, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2006

VILLELA, SAULO MORAES

O Resfriamento Simulado no Projeto Ótimo de Autômatos Celulares para a Geração de Chaves em Criptografia de Fluxo [Rio de Janeiro] 2006

XI, 66 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2006)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Autômatos Celulares
2. Criptografia de Fluxo
3. Resfriamento Simulado

I. COPPE/UFRJ    II. Título (série)

À minha mãe, à minha irmã e  
aos meus avós *Herbert (in  
memoriam)* e *Teresinha*.

# Agradecimentos

Estou certo de que esta seção se não foi a mais, com certeza foi uma das mais difíceis de serem escritas. Isso porque expressar meus agradecimentos a todas as pessoas que contribuíram para que este sonho se tornasse real não é nada fácil. Não se consegue muita coisa sozinho na vida. Ainda bem que pude contar com vocês. Muito obrigado!

Agradeço, primeiramente, a Deus, pela bênção da vida. Tê-Lo como guia é sempre fundamental para se obter sucesso na vida.

Agradeço a minha mãe, por ter sempre me incentivado a buscar meus objetivos. Por ter se privado de alguns de seus sonhos para que eu pudesse realizar os meus.

Agradeço a minha irmã Tatyana, por acreditar em mim e sempre me apoiar em minhas decisões.

Agradeço a meu avô Herbert (*in memoriam*), pelos ensinamentos e lições de vida e a minha avó Teresinha por estar sempre presente me incentivando.

Agradeço a todos os meus familiares e amigos, presentes em todas as ocasiões, seus cuidados, força, carinho e atenção tornaram possíveis este momento.

Agradeço ao Professor Luís Alfredo Vidal de Carvalho, por ter me presenteado com este apaixonante tema de pesquisa e ter sido, além de mestre e orientador, amigo nesta longa caminhada.

Agradeço aos professores e funcionários do PESC pelo suporte necessário.

Agradeço à Viviane, com quem convivo desde a graduação, companheira e amiga que, com sua atenção e carinho, não me deixou esmorecer diante de alguma dificuldade. À Graziella e Ricardo, pelas incansáveis ajudas com o projeto. Àqueles com quem convivi no Rio, em especial Carla, Luciana, Rodrigo, Jairo e Ariadne, os quais me fizeram sentir em “família”. Ao “sócio” Vinicius, pela amizade e apoio na reta final.

A todas as outras pessoas que colaboraram direta ou indiretamente nessa minha conquista, agradeço sinceramente.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

## O RESFRIAMENTO SIMULADO NO PROJETO ÓTIMO DE AUTÔMATOS CELULARES PARA A GERAÇÃO DE CHAVES EM CRIPTOGRAFIA DE FLUXO

Saulo Moraes Villela

Março/2006

Orientador: Luís Alfredo Vidal de Carvalho

Programa: Engenharia de Sistemas e Computação

Criptografia (kriptós = escondido, oculto; grápho = grafia) é a arte ou ciência de escrever em cifra ou em códigos, de forma a permitir que somente o destinatário a decifre e compreenda, ou seja, criptografia transforma textos originais em uma informação alterada, chamada texto cifrado ou simplesmente cifra, que usualmente tem a aparência de um texto randômico ilegível. A criptografia se tornou um requisito básico nessa era de conectividade eletrônica global para assegurar o armazenamento de dados e transmissões contra a possibilidade de interceptação de mensagens e fraudes eletrônicas.

Autômatos celulares têm sido estudados como uma opção de técnica de criptografia. No presente trabalho, utilizamos autômatos celulares para gerar chaves para uma criptografia de fluxo. Para isso, fizemos o uso da meta-heurística Resfriamento Simulado. Extensivos testes nos mostraram diversos resultados e, com eles, conseguimos observar a enorme dificuldade que se é encontrar boas chaves. Combatemos, com isso, os resultados obtidos por trabalhos anteriores.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

THE SIMULATED ANNEALING IN THE OPTIMAL PROJECT OF CELLULAR  
AUTOMATA FOR GENERATING KEYS IN STREAM CRYPTOGRAPHY

Saulo Moraes Villela

March/2006

Advisor: Luís Alfredo Vidal de Carvalho

Department: Systems and Computing Engineering

Cryptography is both an art and a science of secret writing in such a way that only the intended receiver of a message understands its contents. In other words, cryptography transforms original texts into ciphers, or coded texts that appears as nonsense for most of the readers. Cryptography has become a basic requirement in this age of global electronic connectivity to secure data storage and transmission against the possibility of message eavesdropping and electronic fraud.

Cellular automata have been studied as cryptography algorithms. The present work presents a cellular automata model for the generation of keys to flow cryptography with the use o the Simulated Annealing heuristics. Exhaustive tests have shown interesting results, mainly the difficulty of finding good keys. Our experiments deny previous results reported in the literature which establish rules for good key cellular automata generators.

# Sumário

<b>Agradecimentos</b>	<b>iv</b>
<b>Resumo</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Motivação e objetivos .....	1
1.2 Organização .....	4
<b>2 Autômatos celulares</b>	<b>5</b>
2.1 Conceitos essenciais.....	5
2.2 Classificação dos autômatos .....	8
<b>3 Criptografia</b>	<b>10</b>
3.1 Conceitos básicos .....	10
3.2 Algoritmos de fluxo .....	22
<b>4 Resfriamento Simulado</b>	<b>34</b>
4.1 O algoritmo de Metropolis.....	36
4.2 O algoritmo de Kirkpatrick.....	38
4.3 O conceito de entropia .....	41
4.4 Algoritmos Genéticos .....	45
<b>5 O Método proposto</b>	<b>47</b>
5.1 O método.....	47
5.2 Resultados .....	51
5.2.1 Comparação com Algoritmos Genéticos .....	55
5.3 Experimentos .....	56
5.4 VoIP .....	58
<b>6 Considerações finais</b>	<b>60</b>
<b>Referências Bibliográficas</b>	<b>63</b>

# Lista de Figuras

- 2.1 Atualização dos estados do autômato celular de uma dimensão do tempo  $t$  para o tempo  $t + 1$ , aplicando o mapeamento determinístico  $F_f$ , sendo  $f$  a regra de atualização local..... 6
- 2.2 Vizinhanças de von Neumann e Moore, respectivamente. Em ambas temos  $r = 1$ ..... 7
- 2.3 Exemplo de regra de atualização de raio = 1, com vizinhança  $\delta = 3$  e 8 combinações de 3 *bits*, gerando 8 *bits* de saída ordenados lexicograficamente. O binário 01011010 é a regra de atualização 90 expressa em decimal. .... 7
- 2.4 Os 20 primeiros passos do comportamento das 256 regras de 8 *bits* (vizinhança  $\delta = 3$ ). Da esquerda pra direita, de cima pra baixo, começando pela regra 0 até a regra 255. .... 8
- 2.5 Exemplos de padrões evolutivos de autômatos celulares de uma dimensão. As figuras (a)–(d) mostram a evolução de autômatos celulares baseados em quatro regras de atualização diferentes a partir de estados iniciais aleatórios. Nesses casos, temos dois estados possíveis  $S = \{0, 1\}$  e vizinhança  $\delta = 3$ . .... 9
  
- 3.1 Nos algoritmos criptográficos simétricos a mesma chave utilizada na encriptação é utilizada na desencriptação. O transmissor da mensagem e seu receptor devem acordar entre si previamente o valor numérico desta chave e verificar que a mesma seja mantida conhecida apenas por ambos.... 13
- 3.2 Um algoritmo criptográfico unidirecional consome pouco tempo na encriptação de uma mensagem, mas exige um tempo muito grande para que a mensagem original seja restaurada, supondo-se a chave desconhecida..... 16



3.3	Na nova concepção de Diffie, uma chave de conhecimento público faria a encriptação das mensagens de um transmissor qualquer para um receptor específico, dono da chave pública. Já a chave privada, de conhecimento apenas do receptor, faria a desencriptação de todas as mensagens enviadas a ele. ....	19
3.4	Os algoritmos de bloco são aqueles que operam sobre conjuntos de <i>bits</i> (blocos) da mensagem. Geralmente, os blocos possuem 64 <i>bits</i> e o último bloco da mensagem original precisa ser completado com <i>bits</i> zero (ou <i>bits</i> um). ....	21
3.5	Os algoritmos de fluxo são aqueles que operam sobre a mensagem <i>bit a bit</i> , não exigindo a formação prévia de blocos de <i>bits</i> e nem a complementação com <i>bits</i> zero ao final da mensagem original. ....	21
3.6	A operação matemática $L$ pode ser utilizada como forma de criptografar um <i>bit</i> . ....	23
3.7	Os algoritmos de fluxo se utilizam de uma operação matemática $L$ que atua sobre <i>bits</i> individualmente e os criptografa com o auxílio de uma chave de fluxo aleatoriamente gerada com o mesmo tamanho da mensagem original. ....	24
3.8	No processo de desencriptação dos algoritmos de fluxo se utiliza a operação matemática inversa $L_{inv}$ sobre a mensagem criptografada e sobre a chave de fluxo do processo de encriptação. ....	24
3.9	O gerador de chaves de fluxo possui três elementos: Um Módulo de Saída que gera individualmente cada <i>bit</i> da chave de fluxo em função do módulo chamado de Estado Interno; um Módulo de Atualização que altera o Estado Interno após a geração de cada <i>bit</i> , permitindo, assim, que o Módulo de Saída produza um novo <i>bit</i> para a chave de fluxo. A chave-semente estabelece os parâmetros específicos de funcionamento dos três elementos, fazendo com que o gerador comporte-se de um modo determinado. Em consequência, uma chave de fluxo particular é produzida para cada uma das chaves-semente utilizadas. ....	27
3.10	Nos algoritmos de fluxo síncronos, a chave de fluxo da encriptação deve ser gerada em total sincronia com a chave de fluxo da desencriptação ou a mensagem original não será resgatada pelo receptor. A chave-semente é, obviamente, a mesma, pois algoritmos de fluxo são algoritmos criptográficos simétricos. ....	29

3.11	O gerador de chaves de fluxo dos algoritmos assíncronos possui apenas dois elementos: Um Módulo de Saída que gera individualmente cada <i>bit</i> da chave de fluxo em função do módulo chamado de Estado Interno; o Estado Interno é alterado pelos $n$ últimos <i>bits</i> da mensagem criptografada, permitindo, assim, que o Módulo de Saída produza um novo <i>bit</i> para a chave de fluxo. A chave-semente estabelece os parâmetros específicos de funcionamento dos dois elementos, fazendo com que o gerador comporte-se de um modo determinado. Em conseqüência, uma chave de fluxo particular é produzida para cada uma das chaves-semente utilizadas. ....	31
3.12	Nos algoritmos de fluxo assíncronos, as chaves de fluxo de encriptação e de desencriptação dependem dos <i>bits</i> da mensagem criptografada. Como tanto o transmissor quanto o receptor têm acesso a esta mensagem, eles podem gerar chaves de fluxo iguais e sincronizadas sem a necessidade de uma sincronização explícita, como nos algoritmos síncronos. A chave-semente é, obviamente, a mesma, pois algoritmos de fluxo são algoritmos criptográficos simétricos. ....	32
4.1	O Método de Monte Carlo no cálculo da área $S$ . ....	35
4.2	Perfil de uma função objetivo com máximo local. ....	40
4.3	Equilíbrio térmico e cristalização no Resfriamento Simulado. ....	41
5.1	Simulação com as regras 150, 85, 153, 105, 85, 150, 165 e 90, respectivamente. Com entropia 7,9937, o autômato pode ser classificado como caótico. Estão representados 100 passos da simulação. ....	51
5.2	Resultados obtidos usando as regras 150, 85, 153, 105, 85, 150, 165 e 90 em posições aleatórias. Todos, também com 100 passos. Abaixo dos passos, o valor da entropia em cada simulação. ....	53
5.3	Resultados obtidos escolhendo regras a partir da tentativa de criação de uma heurística das posições das soluções ótimas encontradas na Tabela 5.2. Em todas as simulações estão representados 100 passos e abaixo deles, o valor de cada entropia. ....	55
5.4	Resultados obtidos usando as regras 90, 105, 150 e 165 em posições aleatórias. Todos, também com 100 passos. Abaixo dos passos, o valor da entropia em cada simulação. ....	56

## Lista de Tabelas

5.1	Exemplos de soluções ótimas (entropia superior a 7,8).....	52
5.2	As regras e suas posições de 471 exemplos de soluções ótimas.....	54

# Capítulo 1

## Introdução

### 1.1 Motivação e objetivos

Hoje em dia, as redes globais são caracterizadas pelo enorme crescimento da necessidade do armazenamento e transmissão de informações digitais. Com isso, organizações públicas e privadas têm se tornado crescentemente dependentes de técnicas criptográficas para garantir a segurança e autenticidade em diversas áreas, especialmente em transações de comércio eletrônico [1]. A criptografia tem uma longa história [2, 3] e diversas técnicas criptográficas diferentes já foram sugeridas: uma excelente revisão é dada em [4]. Aqui, nós iremos descrever a aplicação de alguns tipos de autômatos celulares neste domínio.

Autômatos celulares (ACs) foram previamente usados como mecanismos de encriptação por Wolfram [5] e por Nandi et al. [6]. Outros autores usaram ACs para criptografia de chave-pública, mas seus trabalhos não serão discutidos aqui, uma vez que usamos ACs para sistemas simétricos, onde as chaves de encriptação e desencriptação são a mesma ou podem ser calculadas uma a partir da outra. Nosso

esquema de encriptação é baseado na geração de seqüências de *bits* pseudo-aleatórios por autômatos celulares.

Autômatos celulares [7, 8] são sistemas dinâmicos nos quais espaço e tempo são discretos. Um autômato celular consiste de um vetor de células, cada uma podendo ser em um de um finito número de possíveis estados, atualizado sincronamente em passos de tempo discretos, de acordo com uma regra interativa local. Aqui, nós iremos somente considerar autômatos Booleanos, para cada estado da célula  $s \in \{0,1\}$ . O estado de uma célula no passo seguinte de tempo é determinado pelos estados atuais de uma vizinhança acerca das células. O vetor celular é  $d$ -dimensional, onde  $d = 1, 2, 3$  é usado na prática; no nosso trabalho, nós concentraremos em  $d = 1$ , isto é, vetores unidimensionais. A regra contida em cada célula é essencialmente uma máquina de estado finito, usualmente especificada na forma de uma tabela de regras (também conhecida como função de transição), com uma entrada para cada possível configuração de vizinhança de estados. A vizinhança celular de uma célula consiste no próprio estado e nos estados das células adjacentes. Para ACs unidimensionais, uma célula é conectada a  $r$  vizinhos locais (células) em cada lado onde  $r$  é referente ao raio (então, cada célula tem  $2r + 1$  vizinhos). Quando consideramos o reticulado de tamanho finito, condições de contorno periódicas são freqüentemente aplicadas, resultando em um reticulado circular no caso unidimensional. Autômatos celulares não-uniformes (também conhecidos como não-homogêneos) são iguais aos uniformes com a exceção que as regras não precisam ser idênticas para todas as células. Atualização assíncrona dos valores das células e transições de estados probabilísticas também é possível, mas não será usada aqui.

Um dos pontos mais importantes dos autômatos celulares é a classificação de suas regras de atualização  $f$ , e, conseqüentemente, dos próprios autômatos. O interesse nesta

questão recebeu sua atenção inicial a partir do estudo dos autômatos celulares infinitos unidimensionais por Stephen Wolfram, que resultaram na descoberta empírica de que, desconsiderando o estado inicial do autômato,  $f$  consistentemente recai dentro de uma das quatro possíveis categorias qualitativas: (i) a evolução do autômato leva a uma configuração homogênea, i.e., uma configuração na qual todas as células possuem o mesmo estado; (ii) a evolução leva a um ponto fixo não-homogêneo ou a um ciclo; (iii) a evolução leva a uma sucessão caótica de configurações; ou (iv) a evolução leva a estruturas espaço-temporais localizadas que são algumas vezes “persistentes”. Embora concebida inicialmente para o caso unidimensional, não existe, a princípio, uma razão pela qual esta classificação qualitativa não possa ser aplicada aos casos de dimensões maiores. De fato, estudos similares para o caso bidimensional apareceram em [9] pouco tempo depois.

O Resfriamento Simulado é uma meta-heurística de busca onde uma solução experimental é modificada aleatoriamente. Uma “energia” é definida para representar quão boa é a solução. O objetivo é encontrar a melhor solução minimizando, ou maximizando, esta energia. Mudanças que conduzem a uma energia mais baixa, ou mais alta no caso de maximização, são aceitas imediatamente; um aumento, ou decréscimo, é aceito probabilisticamente. A probabilidade  $p$  é dada por  $p = e^{-\Delta f / KT}$ , onde  $\Delta f$  é a variação “energética”,  $K$  é a constante de Boltzmann e  $T$  é a “temperatura”. Inicialmente a “temperatura” é elevada correspondendo a um líquido ou estado derretido onde as mudanças grandes são possíveis e ela é reduzida progressivamente usando um resfriamento que permite assim mudanças menores até que o sistema se solidifica em uma solução baixa, ou alta, da energia.

Com isso, propusemos um método que tem como objetivo gerar chaves para uma criptografia de fluxo, a partir de autômatos celulares, utilizando, para isso, o algoritmo

de Resfriamento Simulado, ou seja, tentamos gerar autômatos “ótimos” que seriam boas chaves em uma criptografia de fluxo.

## 1.2 Organização

No primeiro capítulo introduzimos os objetivos do presente trabalho, bem como sua motivação, destacando a importância da criptografia no cenário atual.

Com o intuito de aproximar o leitor dos conceitos fundamentais para o bom entendimento dos assuntos relativos à simulação, abordamos no capítulo 2 os temas mais importantes de autômatos celulares, incluindo uma revisão da classificação de Wolfram. Mostramos alguns conceitos e técnicas de criptografia no capítulo 3, aprofundando em criptografia de fluxo. Já no capítulo 4, apresentamos o funcionamento do algoritmo de Resfriamento Simulado (*Simulated Annealing*).

Tendo, a partir de então, uma proximidade maior desses conceitos, descrevemos, no capítulo 5, os passos do método proposto, juntamente com seus resultados e testes realizados para garantir a qualidade dos mesmos.

Finalmente, no capítulo 6, apresentamos as conclusões do trabalho, bem como as divergências em relação a trabalhos anteriores e algumas dificuldades encontradas. Propomos, também, algumas possibilidades de estudos na área.

# Capítulo 2

## Autômatos celulares

### 2.1 Conceitos essenciais

Autômatos celulares são sistemas dinâmicos, discretos no tempo, que possuem unidades que podem assumir um número finito de estados. Essas unidades são denominadas células, cujos estados evoluem no tempo como resultado da interação com outras células. Desde sua introdução por von Neumann [10], há aproximadamente cinco décadas, os autômatos celulares têm adquirido um *status* cada vez mais proeminente como uma ferramenta de modelagem em várias áreas de pesquisa, e foi até mesmo apontado como uma abstração central na modelagem do processo fundamental da natureza [11].

Para um conjunto de estados possíveis  $S = \{0, \dots, s-1\}$  e para um inteiro  $t \geq 0$ , um autômato celular com  $n$  células evolui do tempo  $t$  para o tempo  $t+1$  através da atualização sincronizada de todos os  $n$  estados pela aplicação de um mapeamento determinístico  $F_f : S^n \rightarrow S^n$ , como está ilustrado na Figura 2.1. Este mapeamento  $F_f$  é global em sua natureza, ou seja, atende a todas as células, e depende da regra de atualização local  $f$  que indica como cada estado individual deve ser atualizado, dado o



estado da célula no tempo  $t$  bem como o estado das células que estão dentro de uma vizinhança de tamanho  $\delta$ . A regra de atualização  $f$  é então o mapeamento de  $S^{1+\delta}$  em  $S$ .

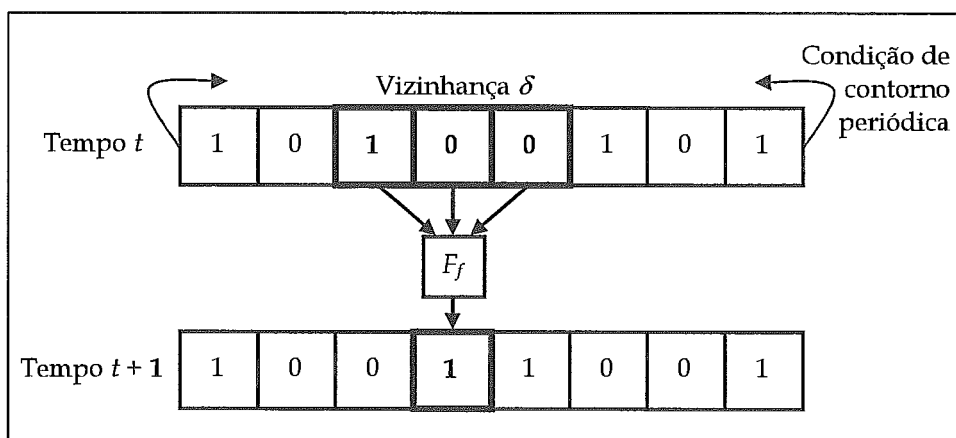


Figura 2.1: Atualização dos estados do autômato celular de uma dimensão do tempo  $t$  para o tempo  $t + 1$ , aplicando o mapeamento determinístico  $F_f$ , sendo  $f$  a regra de atualização local.

A vizinhança de um autômato celular é determinada por um reticulado multidimensional subjacente de acordo com vários critérios possíveis. Por exemplo, um vizinho de uma célula relativo a uma certa dimensão do reticulado pode ser uma das células que estão afastadas  $r > 0$  posições da célula em questão ao longo desta dimensão, mas nenhuma posição afastada em nenhuma outra dimensão, sendo  $r$  sempre referenciado como o raio da regra de atualização naquela dimensão.

Para o raio  $r = 1$  em todas as dimensões, caracteriza-se o que é conhecido como vizinhança de von Neumann (c.f., Figura 2.2). Um outro exemplo de vizinhança seria permitir que duas células fossem vizinhas uma da outra sempre que uma pudesse alcançar a outra sem ultrapassar nenhuma fronteira ao longo de uma certa dimensão, sendo esta fronteira demarcada pelo raio  $r$  da regra de atualização naquela dimensão. Para o raio  $r = 1$  esta é chamada de vizinhança de Moore (c.f., Figura 2.2). Quando  $n$  é finito, é comum considerar que o reticulado tenha bordas cilíndricas, como mostrado na Figura 2.1, ou seja, permitir que toda célula tenha exatamente duas vizinhanças mais

próximas ao longo de cada dimensão (o vizinho da esquerda da primeira célula é a última célula e o vizinho da direita da última célula é a primeira célula).

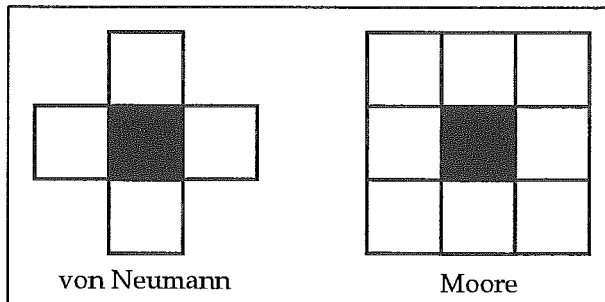


Figura 2.2: Vizinhanças de von Neumann e Moore, respectivamente. Em ambas temos  $r = 1$ .

Em [12], Wolfram propôs um esquema de numeração para os autômatos unidimensionais de raio  $r = 1$ , no qual os *bits* de saídas são ordenados lexicograficamente, como na regra de atualização da Figura 2.3, e são lidos da direita para a esquerda para formar um número na base decimal entre 0 e 255. Por exemplo, a regra apresentada na Figura 2.3 (em binário: 01011010) é a regra de atualização 90. Este esquema de numeração, a partir de uma ordenação lexográfica dos *bits* de saída, é também adotado para autômatos celulares unidimensionais de raio  $r > 1$ . O número de regras de atualização distintas nestes casos é dado por  $2^{2^{2+2r}}$ .

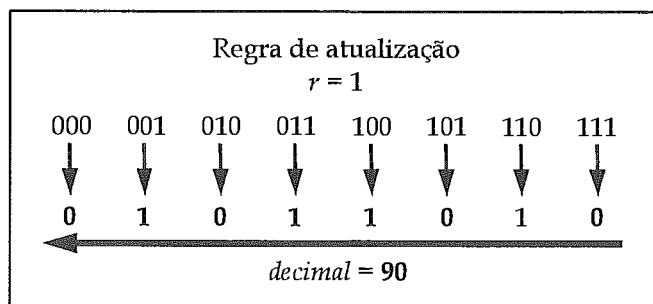


Figura 2.3: Exemplo de regra de atualização de raio  $r = 1$ , com vizinhança  $\delta = 3$  e 8 combinações de 3 *bits*, gerando 8 *bits* de saída ordenados lexicograficamente. O binário 01011010 é a regra de atualização 90 expressa em decimal.

Uma notação similar é adotada para o caso de autômatos celulares bidimensionais sob a vizinhança de von Neumann, entretanto, neste caso, as entradas são lidas em uma

ordem própria; célula, norte, leste, sul, oeste. O número de regras distintas neste caso é dado por  $2^{2^{1+2(\eta+\eta_2)}}$ . Já as regras de atualização de autômatos celulares sob a vizinhança de Moore são totalísticas, isto é, os *bits* de saída são obtidos através da soma dos *bits* da vizinhança e não pelo valor discreto de cada um. Neste caso, o número de regras distintas é  $2^{2^{1+2(\eta+2\eta_1)(1+2\eta_2)}}$ .

A Figura 2.4 ilustra o comportamento de cada uma das 256 regras possíveis de 8 *bits*.

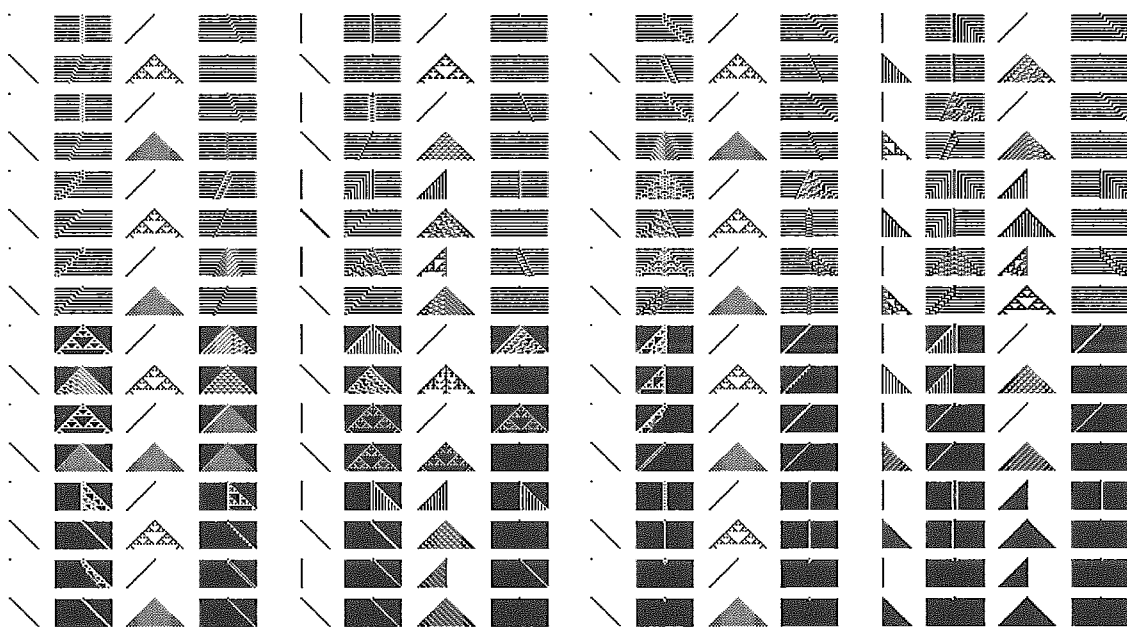


Figura 2.4: Os 20 primeiros passos do comportamento das 256 regras de 8 *bits* (vizinhança  $\delta = 3$ ). Da esquerda pra direita, de cima pra baixo, começando pela regra 0 até a regra 255.

## 2.2 Classificação dos autômatos

Cada regra de atualização leva a padrões que diferem em detalhes. Entretanto, todos os padrões tendem a cair em apenas quatro classes qualitativas conforme Wolfram mostra em [11].

Estas classes de comportamento podem ser caracterizadas empiricamente como segue:

- Classe 1 – evolução leva a estados homogêneos que, por exemplo, tem células com valor 0;
- Classe 2 – evolução leva a um conjunto de estruturas estáveis e periódicas que são separadas e simples;
- Classe 3 – evolução leva a padrões caóticos;
- Classe 4 – evolução leva a estruturas complexas e, algumas vezes, de longa vida.

As Figuras em 2.5 mostram padrões evolutivos de autômatos celulares de uma dimensão, com  $n = 100$  células, na medida em que  $t$  é incrementado. Nos casos apresentados, os dois estados possíveis são representados graficamente por pontos com coloração preta ( $s = 1$ ) e branca ( $s = 0$ ). Cada caso leva a um padrão evolutivo diferente que pode ser classificado qualitativamente segundo as quatro classes de Wolfram.

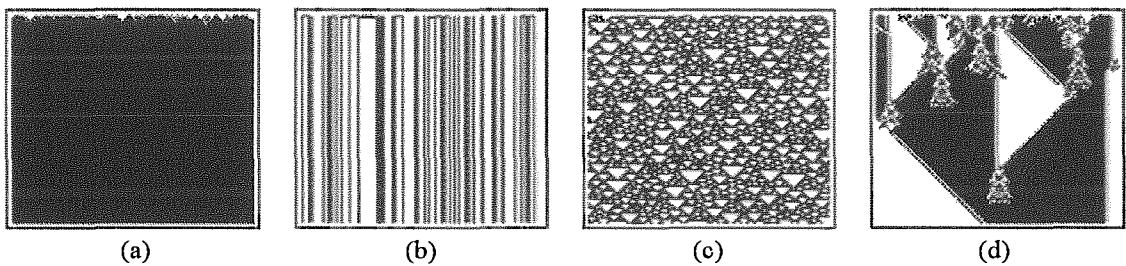


Figura 2.5: Exemplos de padrões evolutivos de autômatos celulares de uma dimensão. As figuras (a)–(d) mostram a evolução de autômatos celulares baseados em quatro regras de atualização diferentes a partir de estados iniciais aleatórios. Nesses casos, temos dois estados possíveis  $S = \{0, 1\}$  e vizinhança  $\delta = 3$ .

A existência de somente quatro classes qualitativas [13] indica, segundo Wolfram, a universalidade do comportamento do autômato celular e muitas características dos autômatos dependem somente da classe que o autômato pertence e não de detalhes precisos da sua evolução.

Estaremos interessados, como será visto no capítulo 5, em autômatos cuja classificação, segundo Wolfram, se enquadre na classe 3, uma vez que nas demais, mesmo a classe 4 levando a padrões complexos, facilmente se consegue descobrir qual será o comportamento nas próximas transições de tempo.

# Capítulo 3

## Criptografia

### 3.1 Conceitos básicos

O computador digital é uma máquina que opera sobre apenas dois símbolos básicos denominados *bit 1* (*binary item one*) e *bit 0* (*binary item zero*). Qualquer número, letra ou palavra que utilizamos em nosso idioma precisa ser transformado em uma seqüência de *bits* para ser representado e processado no computador digital. Costumamos chamar a cada seqüência de 8 *bits* de um “*byte*”. Várias são as formas de representar números e letras em *bits* ou *bytes*. Por exemplo, se utilizarmos uma padronização denominada ASCII (American Standard Code for Information Interchange), a letra “A” maiúscula é representada internamente no computador pelo *byte* composto pelos *bits* 01000001. Já a letra “a” minúscula é representada no código ASCII pelo *byte* 01100001. Outros códigos existem, porém, atualmente, o ASCII é um dos mais utilizados. Os números que costumamos utilizar em nosso dia-a-dia estão representados em uma notação decimal, pois são escritos com dez símbolos possíveis (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9). Eles também, para serem representados no computador digital, precisam ser transformados em uma notação binária, obviamente composta de zeros e uns, pois estes são os únicos símbolos

que o computador é capaz de representar. Assim, por exemplo, o número 100 em notação decimal deve ser representado internamente no computador em notação binária pelo *byte* 01100100.

O fato de o computador digital representar qualquer informação em notação binária deve-se à facilidade de se convencionar que em um circuito eletrônico a existência de corrente elétrica em um certo condutor representa o *bit* 1 enquanto sua inexistência representa o *bit* 0. Nos meios magnéticos utilizados pelo computador digital como memória de longo prazo, por exemplo, os disquetes, também é fácil convencionar que se um ponto está magnetizado ele representa o *bit* 1 enquanto que se não está representa o *bit* 0. A representação binária facilita muito a construção de circuitos eletrônicos capazes de atuar sobre os *bits* realizando operações aritméticas e lógicas com velocidade. Um computador digital dos mais simples atualmente pode realizar milhões de somas, subtrações, multiplicações e divisões em um segundo apenas. Além disto, pode realizar operações lógicas como decidir se um número é maior, menor ou igual a outro, inverter o valor de um *bit* (transformar o *bit* zero em *bit* um ou vice-versa), entre outras, igualmente na velocidade de milhões por segundo.

As operações elementares de aritmética e lógica realizadas pelo computador digital precisam ser organizadas de forma a gerar procedimentos eficazes em executar alguma tarefa específica, de preferência com eficiência. Chamamos de algoritmo a um conjunto de operações (lógicas e aritméticas) que são executadas pelo computador obedecendo a restrições relativas de precedência de maneira a realizar uma tarefa determinada em tempo finito. Se as restrições de precedência para a execução das operações exigem que apenas uma operação seja executada de cada vez, o algoritmo é dito seqüencial enquanto que se algumas operações podem ser executadas ao mesmo tempo em que outras, o algoritmo é dito paralelo. Algoritmos paralelos tendem a ser mais rápidos que

os seqüenciais na execução de uma mesma tarefa, mas exigem computadores especiais com vários processadores nos quais as operações são executadas ao mesmo tempo. Ambos os algoritmos são utilizados em criptografia e sua capacidade de embaralhar e trocar *bits* velozmente faz de qualquer mensagem criptografada computacionalmente um texto incompreensível e impossível de ser descriptografado manualmente.

Os algoritmos computacionais fizeram sua estréia na criptografia em 1943 como conseqüência do esforço de guerra para decifrar códigos criptográficos [3]. Na década de 60, com o preço do computador mais acessível, muitas empresas, universidades e governos em todo o mundo passaram a se utilizar desta máquina e preocupações quanto à segurança dos dados armazenados e transmitidos começaram a surgir. Algoritmos criptográficos começaram a ser utilizados em larga escala e, devido à falta de padronização, problemas de comunicação entre os usuários de criptografia surgiram. As empresas, principalmente os bancos, buscaram a partir de então algum algoritmo criptográfico confiável pela maioria dos usuários e de fácil utilização que pudesse ser adotado como um padrão, resolvendo os conflitos entre os usuários da técnica.

Na época, um determinado algoritmo criptográfico computacional estava se difundindo amplamente em uso, sendo um bom candidato à padronização. Tratava-se do código Lucifer, desenvolvido por Horst Feistel no laboratório da IBM em New York. O código Lucifer foi chamado de um algoritmo simétrico [4] porque a chave utilizada no processo de encriptação era a mesma utilizada no processo inverso de desencriptação. O transmissor da mensagem precisava acertar de comum acordo com o receptor da mesma um número grande (56 *bits*) que seria utilizado como uma chave, sem a qual o algoritmo não podia ser executado. A garantia da segurança da mensagem criptografada se encontrava no fato de a chave ser mantida secreta pelo transmissor e pelo receptor da mensagem, isto é, só ser conhecida pelas duas partes interessadas na troca de

mensagens. Destarte, os algoritmos simétricos passaram a ser conhecidos também como algoritmos de chave secreta (Figura 3.1).

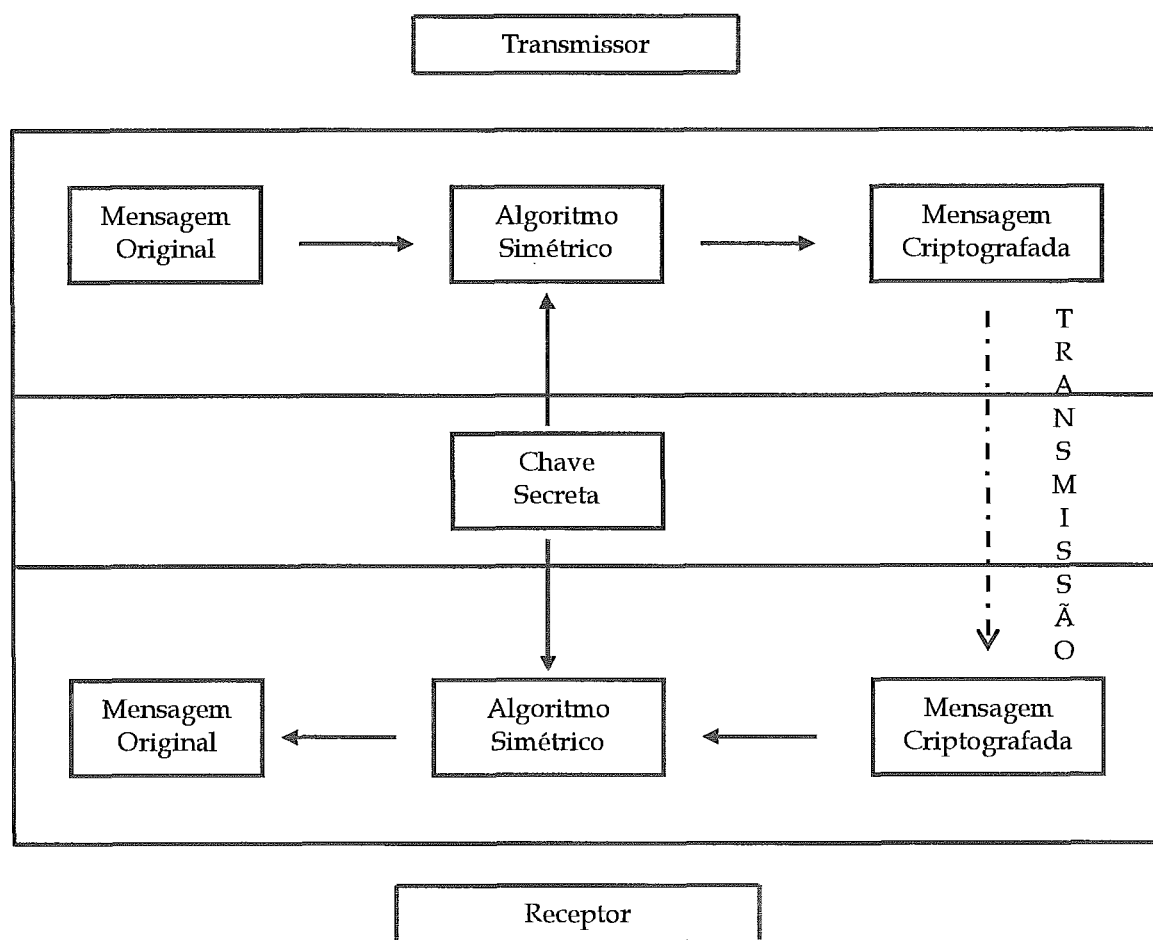


Figura 3.1: Nos algoritmos criptográficos simétricos a mesma chave utilizada na encriptação é utilizada na desencriptação. O transmissor da mensagem e seu receptor devem acordar entre si previamente o valor numérico desta chave e verificar que a mesma seja mantida conhecida apenas por ambos.

O algoritmo Lucifer e seu variante, o Data Encryption Standard, solucionaram a importante questão da uniformização de um método criptográfico entre os usuários. Entretanto, conforme mais e mais empresas passaram a se utilizar dos algoritmos criptográficos computacionais, a necessidade de troca de chaves secretas entre cada par de usuários que desejavam se comunicar aumentou enormemente.

O problema da distribuição de chaves secretas foi resolvido provisoriamente através de emissários de alta confiança das empresas que viajavam através do país ou do mundo



levando em suas “maletas 007” fitas de papel, cartões perfurados, documentos ou discos magnéticos nos quais estavam registradas as chaves a serem utilizadas nas comunicações entre as partes nas próximas semanas. Os governos dos países ricos, grandes usuários de criptografia, realizavam verdadeiras operações de guerra com escoltas armadas e todo o tipo de medidas de segurança para realizar a distribuição de chaves entre seus órgãos administrativos internos e também entre estes e suas embaixadas no exterior.

Se o problema da distribuição de chaves existiu muito tempo antes da criptografia computacional ter surgido, sendo sempre superado sem grandes custos pelo envio dos emissários especiais, agora, com um número enorme de pares desejando se comunicar secretamente utilizando o computador, a logística dos emissários não era mais aceitável por questões de tempo e dinheiro. Desta forma, a distribuição de chaves passou a ser um problema central, estudado por muitos pesquisadores em Criptografia.

Por volta de 1975, um poderoso trio da Universidade de Stanford conseguiu elegantemente resolver o problema da distribuição de chaves, além de revolucionar toda a criptografia. O genial estudante de pós-graduação Whitfield Diffie aliou-se aos professores Martin Hellman e Ralph Merkle e propuseram um algoritmo que permitia que duas pessoas estabelecessem uma chave de encriptação sem ter que trocá-la por qualquer meio de comunicação.

Sua idéia pode ser ilustrada por um exemplo, supondo-se que um indivíduo 1 deseje se comunicar com um indivíduo 2. Os dois, através de algum meio de comunicação, digamos o telefone, acordam entre si o valor de dois números  $A$  e  $B$ . Após isto, o indivíduo 1 escolhe, a seu gosto, um número  $x_1$  e o mantém em segredo e o indivíduo 2 escolhe, também a seu prazer, um número  $x_2$ , igualmente mantido secreto por ele. O indivíduo 1, de posse dos parâmetros acordados  $A$  e  $B$  e do seu número secreto  $x_1$ ,

utiliza o algoritmo de Diffie e calcula um número intermediário que chamaremos de  $I_1$ . Da mesma forma, o indivíduo 2, de posse dos mesmos parâmetros inicialmente acertados  $A$  e  $B$  e do seu número secreto  $x_2$ , também utiliza o algoritmo de Diffie e calcula um número intermediário que chamaremos de  $I_2$ . Agora, novamente os dois indivíduos entram em contato telefônico e o indivíduo 1 informa ao indivíduo 2 o valor de seu número intermediário  $I_1$  enquanto o indivíduo 2 passa o valor de  $I_2$  para o seu parceiro, o indivíduo 1. A ligação telefônica entre os dois é finalizada e o indivíduo 1, tendo agora o número  $I_2$ , utiliza novamente o algoritmo de Diffie e calcula o valor da chave  $K$ . Ao mesmo tempo, o indivíduo 2 possuindo em suas mãos o número  $I_1$ , também se utiliza do algoritmo de Diffie e calcula o valor da chave  $K$ . Os dois indivíduos possuem, finalmente, uma chave criptográfica  $K$  que poderão utilizar para se comunicar secretamente através de qualquer algoritmo criptográfico simétrico que desejem.

Note que em nenhum momento a chave secreta  $K$  foi transmitida pela linha telefônica e, portanto, não houve risco de que algum intruso interceptasse a ligação e descobrisse a chave. O algoritmo de distribuição de chaves de Diffie-Hellman-Merkle, como ficou conhecido este procedimento, tem o cuidado de trocar as mínimas informações necessárias ao cálculo da chave secreta, mas nunca troca a chave propriamente dita. É fácil entender que o algoritmo precisa de cinco parâmetros para realizar o cálculo correto da chave, são eles  $A$ ,  $B$ ,  $I_1$ ,  $I_2$  aliados a  $x_1$  ou a  $x_2$ . Os parâmetros  $A$ ,  $B$ ,  $I_1$ ,  $I_2$ , são transmitidos pelas partes envolvidas por algum meio de comunicação e, sendo assim, podem ser interceptados por um intruso. No entanto, os parâmetros  $x_1$  e  $x_2$  são escolhidos isoladamente e independentemente por cada um dos indivíduos envolvidos e mantidos em segredo por estes e nunca transmitidos por meio de comunicação algum.

Um olhar mais atento para o algoritmo de Diffie-Hellman-Merkle nos mostra que o intruso, de posse dos quatro parâmetros  $A$ ,  $B$ ,  $I_1$  e  $I_2$ , pode calcular os parâmetros secretos  $x_1$  ou  $x_2$  e, com isto, também calcular a chave secreta  $K$ . Ocorre, porém, que o volume de cálculos que este intruso tem de executar para descobrir  $x_1$  ou  $x_2$  é tão grande que só alcançaria o intento de calcular a chave secreta  $K$  após um tempo muito longo. O algoritmo de Diffie-Hellman-Merkle possui esta propriedade pois se utiliza do conceito de unidirecionalidade. Chamamos de algoritmo criptográfico unidirecional ao algoritmo que criptografa uma mensagem rapidamente desde que a chave de encriptação seja dada e que para descriptografar a mensagem, supondo desconhecida a chave, consome um tempo em tentativas e erros tão grande que, do ponto de vista prático, torna a descriptação impossível (Figura 3.2). Assim, um intruso que intercepte os parâmetros  $A$ ,  $B$ ,  $I_1$  e  $I_2$ , transmitidos pela linha telefônica, levará tanto tempo realizando cálculos para descobrir o parâmetro secreto  $x_1$  (ou  $x_2$ ) que a mensagem criptografada já terá cumprido seu papel e estará inútil.

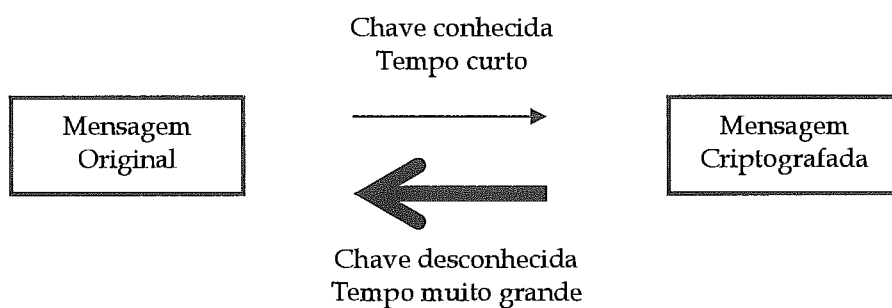


Figura 3.2: Um algoritmo criptográfico unidirecional consome pouco tempo na encriptação de uma mensagem, mas exige um tempo muito grande para que a mensagem original seja restaurada, supondo-se a chave desconhecida.

Cabe notar que toda mensagem criptografada tem um tempo de vida útil. Se um criptoanalista norte-americano decifrasse a mensagem criptografada japonesa que informava aos comandantes nipônicos a data e a hora exatas do ataque a Pearl Harbor

minutos após ao ataque ter sido realizado, de nada adiantaria. Na realidade foi isto que aconteceu! Um outro exemplo ilustrativo é o de um suposto ladrão que consegue descriptar uma mensagem criptografada que informa os detalhes de uma grande transferência de dinheiro entre dois bancos um mês após a mesma ter ocorrido e o dinheiro já ter sido, inclusive, gasto. Destarte, uma mensagem criptografada precisa estar protegida apenas pelo tempo mínimo necessário para o uso devido de seu conteúdo, ou seja, pelo tempo apenas de sua vida útil. Em outras palavras, qualquer método criptoanalítico que só consegue descriptar uma mensagem secreta após a sua vida útil não serve para nada.

Não podemos, todavia, esquecer do detalhe de que o tempo consumido em cálculos computacionais depende, obviamente, do computador utilizado. Um algoritmo criptográfico unidirecional atualmente, tomando-se como medida os computadores hoje existentes, pode deixar de ser unidirecional dentro de alguns anos ou décadas quando computadores mais velozes existirem.

Retornando ao genial Diffie, podemos afirmar, com toda a certeza, que seu algoritmo de distribuição de chaves destruiu o dogma milenar da criptografia que afirmava que se duas partes desejavam trocar mensagens secretas, elas deveriam em algum momento se arriscar a entrar em contato, direto ou indireto, para trocar uma chave de encriptação, residindo neste fato o ponto mais frágil de todo o processo, devido à possibilidade de interceptação desta essencial chave por um terceiro. Após Diffie, duas partes podem acordar entre si uma chave secreta sem o medo da interceptação da mesma, pois esta é calculada pelas partes e não transmitida como o foi durante toda a História.

Uma vez matematicamente provado ser possível a comunicação criptografada entre pares sem a necessidade de troca de chaves, o próximo passo de Diffie era desenvolver

um algoritmo criptográfico no qual este novo princípio estivesse embutido. Se tal algoritmo existisse, qualquer pessoa poderia se comunicar secretamente com a outra sem nunca terem algum tipo de contato prévio para sequer acordar o valor de alguma chave. Diffie não sabia ao certo se este algoritmo seria realmente possível, mas, acreditando que sim, especificou algumas propriedades do mesmo. Para ele, qualquer indivíduo que desejasse utilizar este suposto algoritmo deveria escolher para si um par de chaves. Uma das chaves seria a chave pública, capaz de criptografar mensagens a serem enviadas para aquele indivíduo, enquanto a outra chave seria a chave privada, útil apenas para que o indivíduo descriptografasse as mensagens para ele enviadas. A chave pública, como o nome especifica, deveria ser de conhecimento de todos para que qualquer um que desejasse criptografar mensagens para o seu dono o fizesse sem precisar de nenhum contato prévio com o mesmo. A chave pública seria divulgada ao lado do nome de seu proprietário em um catálogo, como uma lista telefônica, ou mesmo em sua *homepage* na internet, atualmente. Como a chave pública só tem a função de criptografar mensagens para o seu proprietário, não podendo descriptografá-las, não haveria o menor problema em que ela fosse amplamente divulgada. Já a chave privada seria guardada secreta pelo seu proprietário, nunca devendo ser divulgada ou transmitida, pois, de posse da mesma, qualquer intruso seria capaz de descriptografar mensagens enviadas ao verdadeiro proprietário.

O par chave pública / chave privada seria escolhido por cada indivíduo não podendo haver dois indivíduos no mundo com o mesmo par de chaves. Certamente que para o correto funcionamento deste novo algoritmo, as chaves pública e privada deveriam guardar uma relação matemática precisa entre si e uma não poderia ser escolhida independentemente da outra (Figura 3.3). Ao contrário dos algoritmos criptográficos simétricos que se utilizam de uma mesma chave para encriptação e desencriptação, a

idéia de Diffie representava um algoritmo assimétrico, pois a chave de encriptação (chave pública) seria diferente da chave de desencriptação (chave privada).

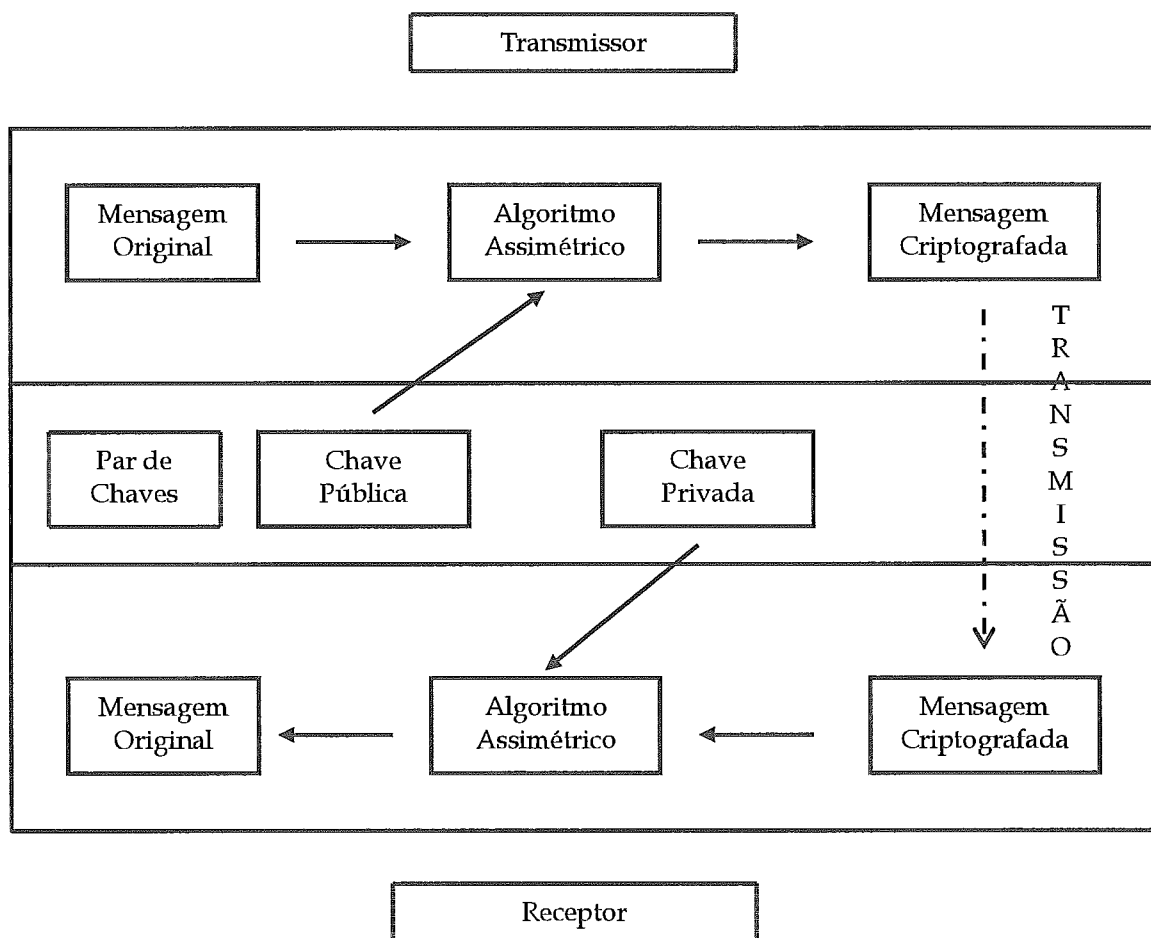


Figura 3.3: Na nova concepção de Diffie, uma chave de conhecimento público faria a encriptação das mensagens de um transmissor qualquer para um receptor específico, dono da chave pública. Já a chave privada, de conhecimento apenas do receptor, faria a desencriptação de todas as mensagens enviadas a ele.

A idéia de se criar uma nova criptografia com a associação “chave pública – encriptação / chave privada – desencriptação” era altamente sedutora, mas não havia, na época, indícios de que tal processo pudesse ser realizado. O trio Diffie, Hellman e Merkle fez várias tentativas para encontrar um algoritmo unidirecional que atendesse a este quesito, mas falharam em todas.

Em 1977, um outro trio sinérgico, composto pelos professores do MIT (*Massachusetts Institute of Technology*), Leonard Adleman, Adi Shamir e Ron Rivest descobriu um algoritmo unidirecional que atendia às especificações de Diffie. O algoritmo foi nomeado RSA [14, 15] devido às iniciais de Rivest, Shamir e Adleman. O RSA passou a ser um dos principais marcos na Criptografia, criando a nova Criptografia de Chave Pública, utilizadíssima atualmente e, ao que parece, em muitos anos ainda por vir.

Até aqui vimos que a evolução dos algoritmos criptográficos converge para a existência de “duas” criptografias: a de chave secreta e a de chave pública. A primeira delas se utiliza de algoritmos simétricos com chaves de encriptação e desencriptação iguais e mantidas secretas. Já a segunda delas se utiliza de algoritmos assimétricos que possuem uma chave pública de encriptação diferente da chave privada de desencriptação. O desenvolvimento posterior das pesquisas na área tornou possível a existência de muitos algoritmos simétricos que podem ser classificados em algoritmos de bloco e algoritmos de fluxo, de acordo com a forma com que operam sobre as mensagens.

Os algoritmos simétricos de bloco trabalham sobre um conjunto de  $n$  bits da mensagem de cada vez chamado bloco. Normalmente,  $n$ , o tamanho do bloco, é de 64 bits, ou seja, a mensagem a ser criptografada (ou descriptografada) precisa ser inicialmente dividida em blocos de 64 bits cada um para, posteriormente, ser operada pelo algoritmo de bloco. Se uma mensagem não possui um número de bits múltiplo de 64, então o último bloco precisa ser completado com bits zero (ou bits um) até alcançar os 64 bits necessários à formação de um bloco (Figura 3.4).

Em contraponto, os algoritmos de fluxo operam sobre cada bit da mensagem individualmente (Figura 3.5).

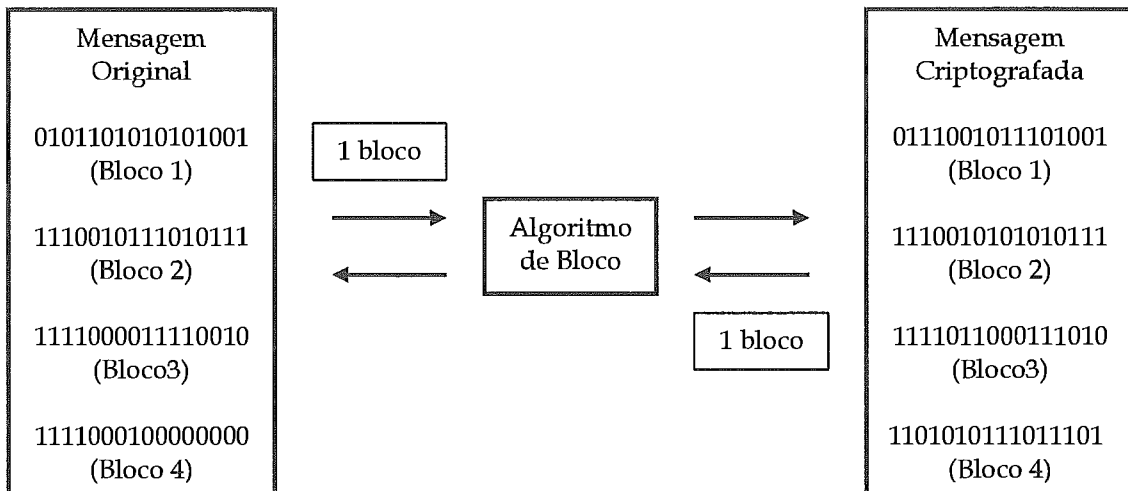


Figura 3.4: Os algoritmos de bloco são aqueles que operam sobre conjuntos de *bits* (blocos) da mensagem. Geralmente, os blocos possuem 64 *bits* e o último bloco da mensagem original precisa ser completado com *bits* zero (ou *bits* um).

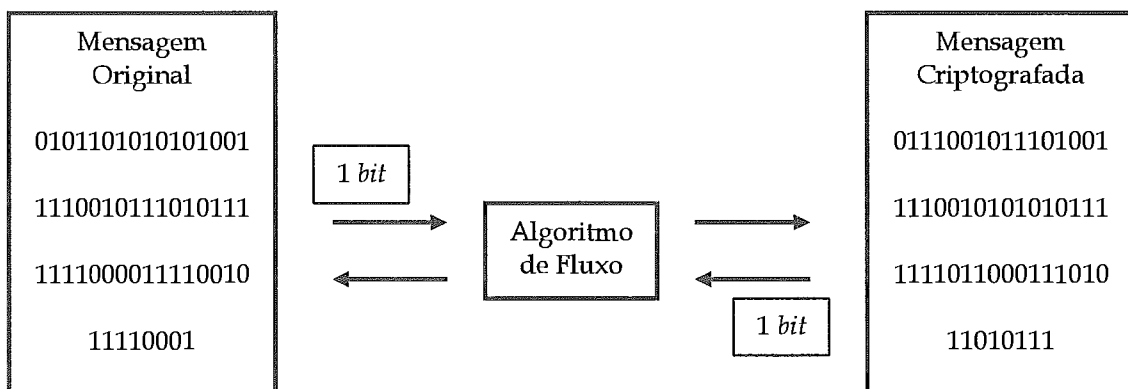


Figura 3.5: Os algoritmos de fluxo são aqueles que operam sobre a mensagem *bit a bit*, não exigindo a formação prévia de blocos de *bits* e nem a complementação com *bits* zero ao final da mensagem original.

Estes algoritmos são úteis quando o transmissor da mensagem precisa enviar para o receptor um fluxo contínuo de dados criptografados. Por exemplo, o caso de um computador que está permanentemente transmitindo dados criptografados para outro computador é típico para o uso de algum algoritmo criptográfico de fluxo.

Estes algoritmos também são necessários quando um transmissor não tem como acumular os dados para permitir a formação de blocos, seja por falta de tempo ou de



espaço. Seria o caso de um computador com pouca memória ou com a necessidade de enviar rapidamente os dados criptografados para o outro computador receptor. Os algoritmos de fluxo serão analisados detidamente na próxima seção, enquanto para os algoritmos de blocos uma excelente revisão é encontrada em [15, 16].

## 3.2 Algoritmos de fluxo

Como já comentado na seção anterior, os algoritmos de fluxo (*stream ciphers*) [16] são algoritmos de chave simétrica que operam sobre cada *bit* da mensagem individualmente. Estes algoritmos são úteis quando o transmissor da mensagem precisa enviar para o receptor um fluxo contínuo de dados criptografados, como, por exemplo, um computador que está permanentemente transmitindo dados criptografados para outro computador. Estes algoritmos também são necessários quando um transmissor não tem como acumular os dados para permitir a formação de blocos, seja por falta de tempo ou de espaço, como um computador com pouca memória ou com a necessidade de enviar rapidamente os dados criptografados para o outro computador receptor.

Nos algoritmos de fluxo, alguma operação matemática, que chamaremos de  $L$ , precisa ser realizada com cada *bit* da mensagem separadamente de forma a alterar-lhe o valor, criptografando-o. A operação  $L$  é normalmente uma função “ou-exclusivo”. É suficiente para isto que convençionemos que um dos *bits* a serem operados por  $L$  pertença à mensagem original a ser encriptada enquanto o outro *bit* necessário à operação seja uma chave de encriptação. O resultado da operação  $L$  sobre o *bit* da mensagem e sobre a chave de encriptação é o terceiro *bit* que representa um *bit* da mensagem criptografada (Figura 3.8).

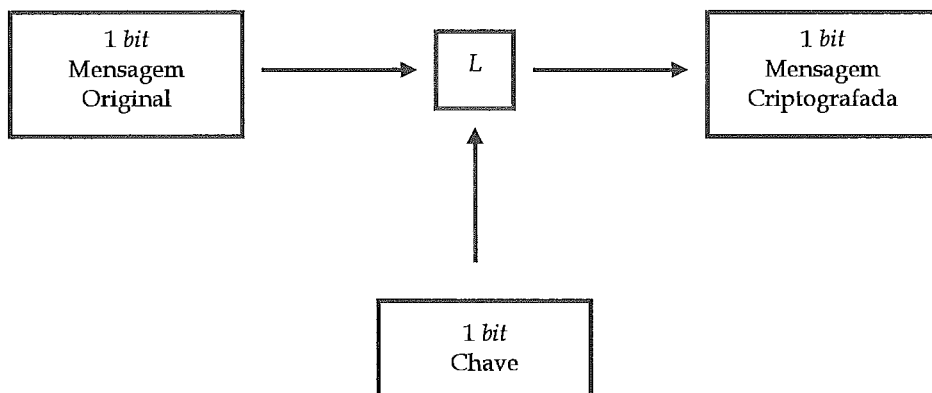


Figura 3.6: A operação matemática  $L$  pode ser utilizada como forma de criptografar um *bit*.

Certamente que a chave de encriptação não pode ser constante (sempre *bit* 1 ou sempre *bit* 0) pois, se assim for, o resultado da operação  $L$  sobre o *bit* da mensagem original será previsível e o *bit* da mensagem criptografada poderá ser facilmente descriptado. Destarte, a chave de encriptação deve ser constituída de *bits* que variam no tempo, encriptando cada *bit* da mensagem original diferentemente. O ideal, na realidade, seria que a chave de encriptação fosse uma seqüência de *bits* aleatoriamente gerados e, portanto, imprevisível para qualquer intruso desejoso de criptanalizar a mensagem.

Como conseqüência do que foi exposto acima, e observando a Figura 3.6, a chave de encriptação deve possuir o mesmo número de *bits* da mensagem original, pois, só assim, cada *bit* da mensagem poderá ser encriptado por um *bit* da chave diferente. A chave de encriptação é, então, uma seqüência aleatória de  $n$  *bits*, na qual  $n$  é o número de *bits* da mensagem original. Costuma-se chamar esta seqüência de *bits* de chave de fluxo. Basicamente, todos os algoritmos de fluxo possuem, conseqüentemente, a estrutura indicada na Figura 3.7.

O processo de descriptação é semelhante ao de encriptação, bastando utilizar-se a operação matemática inversa  $L_{inv}$  sobre a mensagem criptografada e com a mesma chave de fluxo (Figura 3.8).

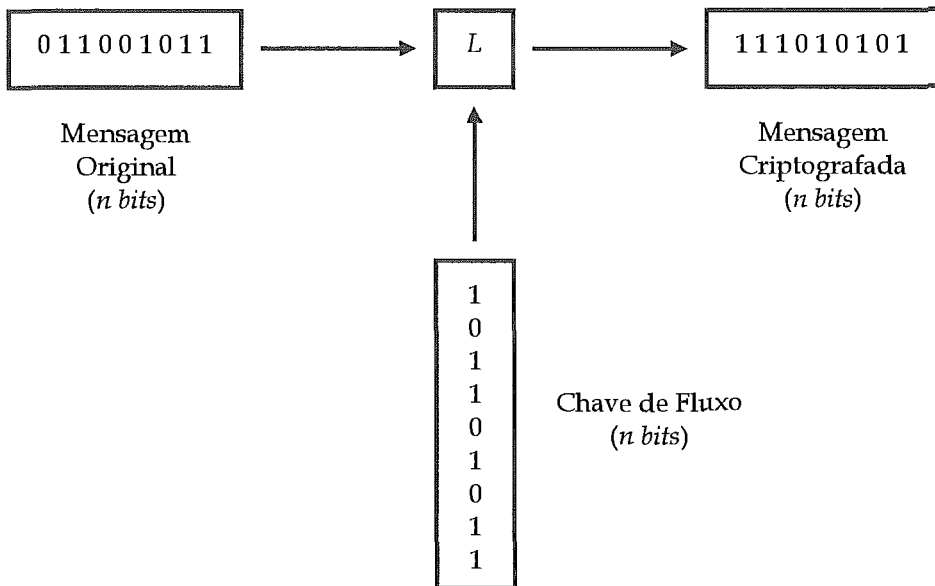


Figura 3.7: Os algoritmos de fluxo se utilizam de uma operação matemática  $L$  que atua sobre *bits* individualmente e os criptografa com o auxílio de uma chave de fluxo aleatoriamente gerada com o mesmo tamanho da mensagem original.

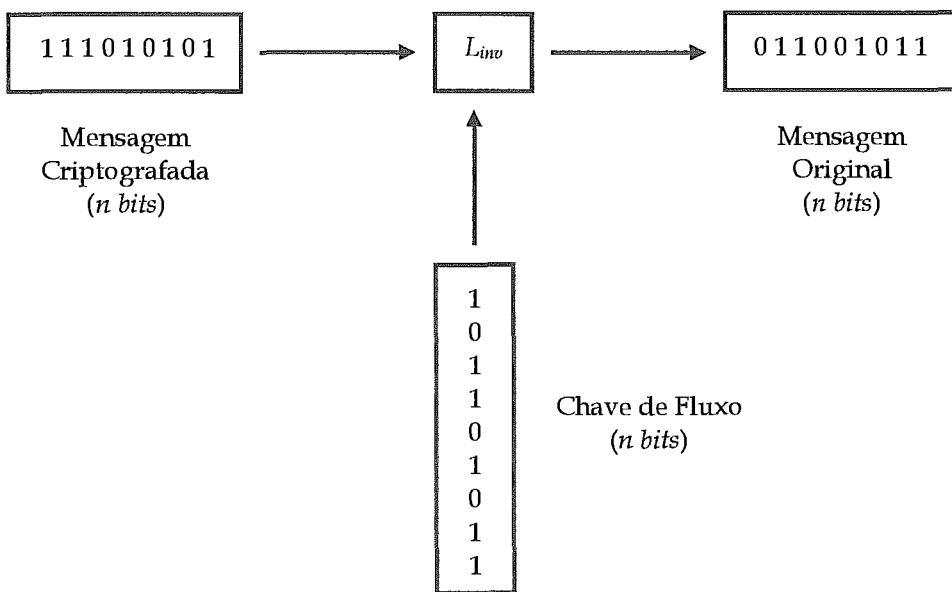


Figura 3.8: No processo de descriptação dos algoritmos de fluxo se utiliza a operação matemática inversa  $L_{inv}$  sobre a mensagem criptografada e sobre a chave de fluxo do processo de encriptação.

Como na descriptação a chave de fluxo utilizada deve ser a mesma usada no processo de encriptação, um problema conceitual e operacional surge: se a chave de fluxo da encriptação for realmente aleatória, ela nunca poderá ser gerada novamente

para o processo de descriptação. Certamente que estamos assumindo que a chave de fluxo não é guardada em algum arquivo ou transmitida para o receptor da mensagem, pois isto seria muito prejudicial à segurança do algoritmo criptográfico. A chave de fluxo deve, então, ser gerada no ato de encriptação e, posteriormente, reproduzida, no ato de descriptação. Daí, a chave de fluxo não poder ser efetivamente aleatória, mas sim, uma seqüência de *bits* denominada de pseudo-aleatória, isto é, uma seqüência que possui algumas propriedades da aleatoriedade, mas que pode ser reconstruída quando se desejar.

No capítulo 5, discutiremos como gerar números pseudo-aleatórios, mas o que importa agora é que fique compreendido que a segurança dos algoritmos de fluxo reside na capacidade da chave de fluxo simular, emular, imitar com bastante perfeição uma seqüência de *bits* aleatoriamente gerada sem ser realmente aleatória.

Do que foi visto até aqui, podemos concluir que os algoritmos de fluxo necessitam de alguma máquina, ou processo, gerador de chaves de fluxo para poderem funcionar. Este gerador de chaves de fluxo deve ser capaz de produzir, para a encriptação, e reproduzir, para a descriptação, a mesma chave de fluxo com características de aleatoriedade suficientes para garantir a segurança do algoritmo.

Em adição, a chave de fluxo não deve ser a mesma para criptografar todas as mensagens. É importante que seja possível gerar chaves de fluxo diferentes quando se deseja criptografar mensagens também diferentes. Esta possibilidade aumenta sobremaneira a segurança dos algoritmos de fluxo. Assim, o gerador de chaves de fluxo precisa ser capaz de produzir um número muito grande de seqüências pseudo-aleatórias de *bits* para poder criptografar um sem-número de mensagens com segurança.

Os geradores de chave de fluxo, em geral, possuem uma chave-semente que dispara o processo de geração de uma chave de fluxo específica, dentre todas as possíveis de

serem geradas pelo mesmo. Quando o transmissor deseja criptografar uma mensagem, ele escolhe uma chave-semente e encripta a mensagem com a seqüência pseudo-aleatória de *bits* (chave de fluxo) produzida pelo gerador de chaves de fluxo. O receptor da mensagem deve conhecer a chave-semente escolhida pelo transmissor, pois só assim poderá iniciar o processo de geração da mesma chave de fluxo e descriptar a mensagem. Note que estamos tratando aqui de algoritmos simétricos ou de chave secreta, sendo por isto necessário o acerto de uma mesma chave entre o transmissor e o receptor da mensagem.

A vantagem na existência desta chave-semente está no fato de que ela possui um tamanho pequeno em relação ao tamanho da chave de fluxo, que precisa possuir sempre o mesmo número de *bits* da mensagem a ser tratada. Por conseguinte, a chave-semente pode ser facilmente armazenada e transmitida para o receptor que, de posse da mesma, gerará a chave de fluxo correspondente e realizará a descriptação. Ou seja, que um grande segredo (a chave de fluxo) é mantido seguro através de um pequeno segredo (a chave-semente), respeitando um dos princípios básicos da criptografia e do bom senso.

A chave-semente tem o papel de determinar os parâmetros internos do gerador de chaves de fluxo, especificando o seu funcionamento e obrigando-o a gerar uma chave de fluxo particular. No interior do gerador de chaves de fluxo encontramos um registrador (uma memória) capaz de armazenar um certo número de *bits* e recebendo o nome de estado interno. Com base neste estado interno, um outro módulo do gerador, denominado módulo de saída, gera um *bit* da chave de fluxo. Finalmente, um terceiro elemento do gerador de chaves de fluxo, o módulo de atualização, atualiza o estado interno, alterando-lhe o valor para que um novo *bit* da chave de fluxo possa ser produzido pelo módulo de saída, futuramente.

A Figura 3.9 ilustra o interior de um gerador de chaves de fluxo conforme descrito anteriormente.

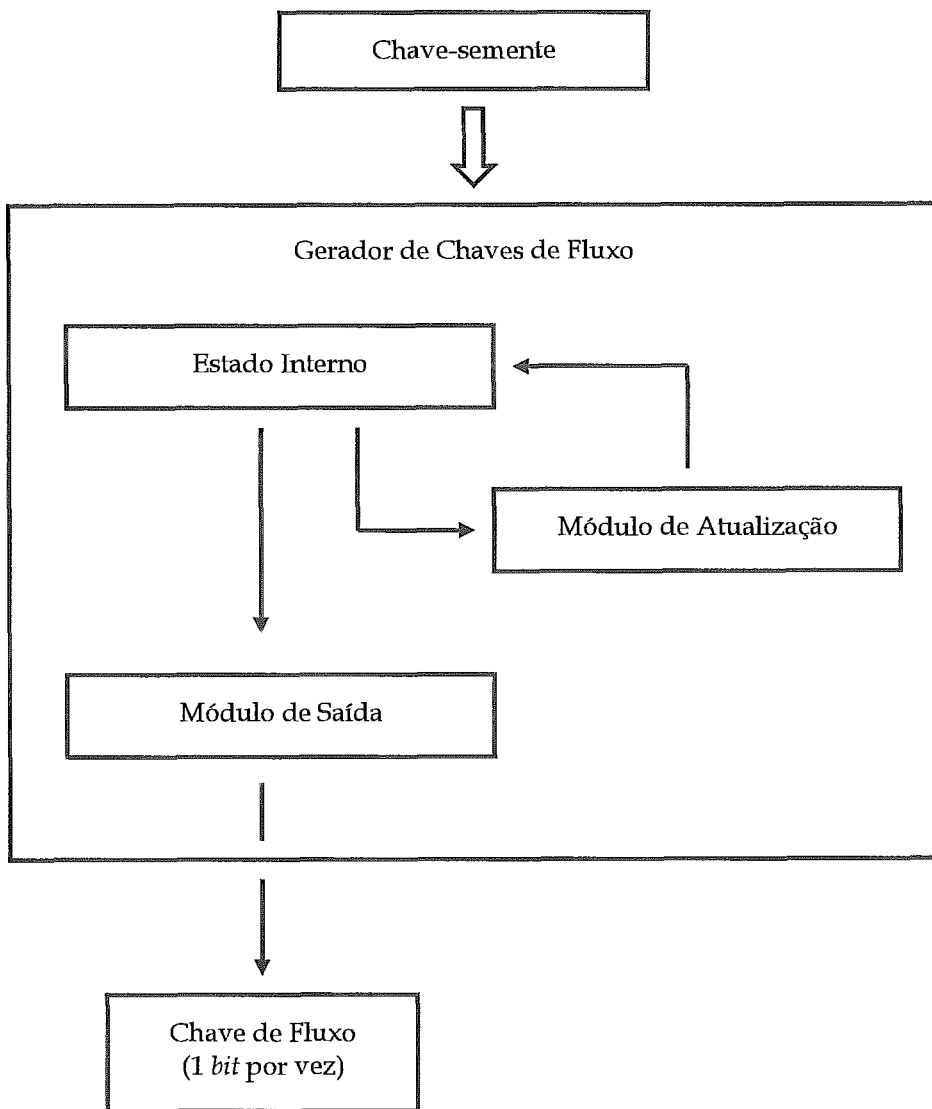


Figura 3.9: O gerador de chaves de fluxo possui três elementos: Um Módulo de Saída que gera individualmente cada *bit* da chave de fluxo em função do módulo chamado de Estado Interno; um Módulo de Atualização que altera o Estado Interno após a geração de cada *bit*, permitindo, assim, que o Módulo de Saída produza um novo *bit* para a chave de fluxo. A chave-semente estabelece os parâmetros específicos de funcionamento dos três elementos, fazendo com que o gerador comporte-se de um modo determinado. Em conseqüência, uma chave de fluxo particular é produzida para cada uma das chaves-semente utilizadas.

Os algoritmos fluxo possuem diferentes “modos de operação”. Destarte, existem os algoritmos de fluxo ditos síncronos e os assíncronos. Nos algoritmos de fluxo síncronos,

a chave de fluxo utilizada na encriptação da mensagem original deve estar em perfeita sincronia com a chave de fluxo usada na descriptação. Somente desta forma cada *bit* encriptado poderá ser descriptado corretamente. Se, por um incidente, um *bit* da mensagem criptografada for perdido durante a transmissão, o processo de descriptação se fará erradamente a partir deste momento, pois o receptor da mensagem a estará descriptando com *bits* da chave de fluxo que não correspondem aos utilizados na encriptação. A chave de fluxo do receptor estará defasada de um *bit* em relação à chave de fluxo do transmissor.

É preciso que o receptor da mensagem esteja sempre atento ao processo de descriptação porque, ao primeiro sinal de que a mensagem está sendo incorretamente descriptada, ele deve interromper o processo e entrar em contato com o transmissor da mensagem para que ambos, de alguma forma, re-sincronizem suas chaves de fluxo. Normalmente, o transmissor e o receptor convencionam entre si a inserção na mensagem criptografada de algum tipo de marcador (uma seqüência de *bits* 1, por exemplo) de forma que, ao se detectar algum erro de descriptação, ambos retornem ao ponto marcado e re-sincronizem suas chaves de fluxo a partir daí. Isto evita que os processos de transmissão e descriptação sejam repetidos desde o primeiro *bit* da mensagem por causa da ocorrência de alguma falha de transmissão.

A vantagem dos algoritmos de fluxo síncronos reside no fato de que a troca de valor de um *bit* da mensagem criptografada só provoca erro na descriptação deste mesmo *bit*. Em outras palavras, não há propagação de erros. Lembre-se de que a troca de valor de um *bit* é um erro muito mais freqüente do que a perda de um *bit* durante transmissão.

A Figura 3.10 ilustra o funcionamento de um algoritmo de fluxo síncrono.

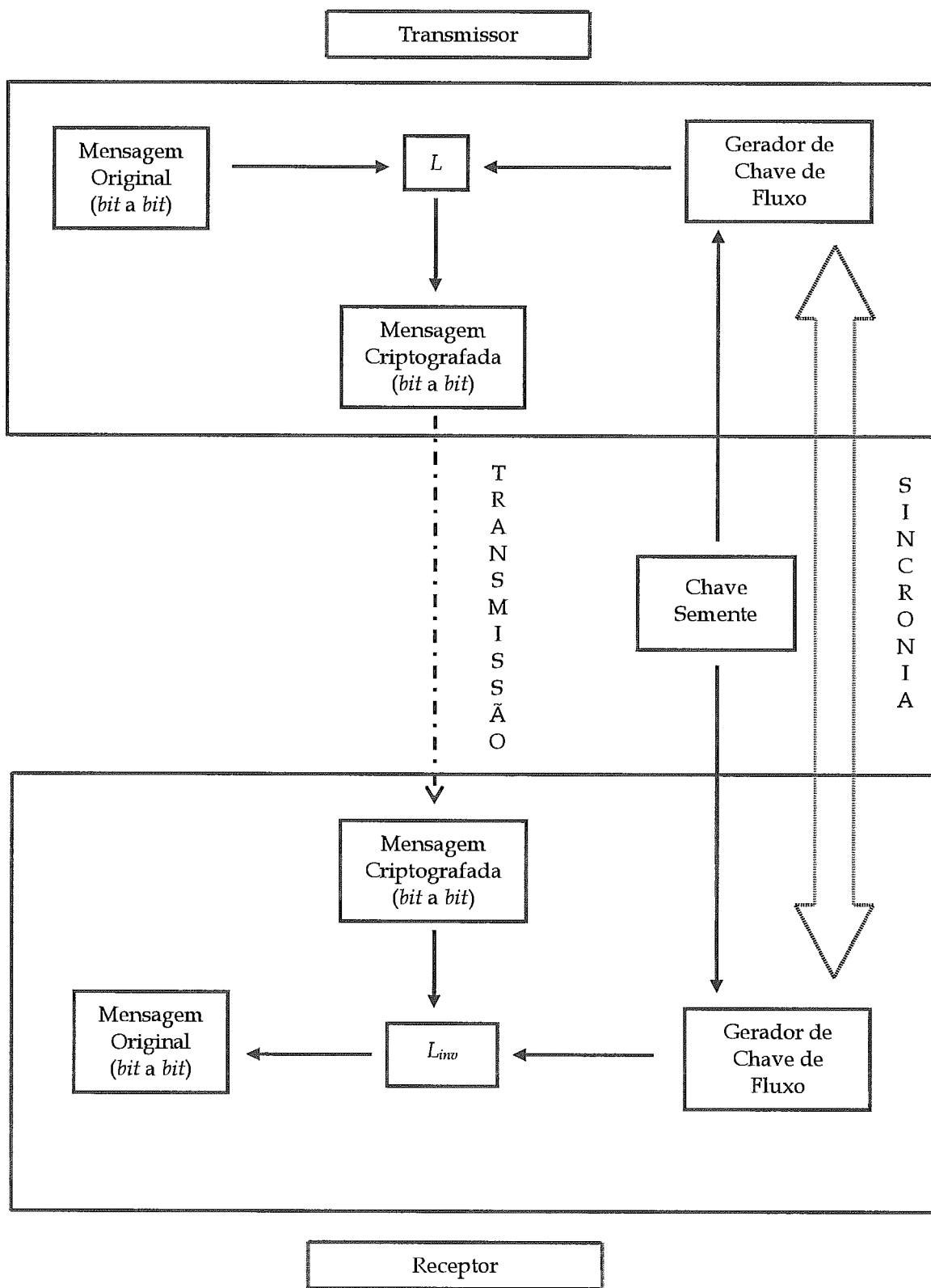


Figura 3.10: Nos algoritmos de fluxo síncronos, a chave de fluxo da encriptação deve ser gerada em total sincronia com a chave de fluxo da descriptação ou a mensagem original não será resgatada pelo receptor. A chave-semente é, obviamente, a mesma, pois algoritmos de fluxo são algoritmos criptográficos simétricos.



Os algoritmos de fluxo assíncronos, também chamados de auto-sincronizantes, diferem dos síncronos no detalhe de que a chave de fluxo depende dos últimos  $n$  bits da mensagem criptografada ( $n$ , um inteiro maior que um). Sendo assim, o transmissor da mensagem e o receptor da mesma não precisam se preocupar em sincronizar suas chaves de fluxo, pois a cada *bit* criptografado (ou descriptografado) o *bit* da chave de fluxo está completamente determinado pelos  $n$  bits da mensagem criptografada que acabaram de passar pelo canal de transmissão.

Operacionalmente, esta idéia é implementada suprimindo-se o módulo de atualização do gerador de chaves de fluxo e fazendo-se com que os últimos  $n$  bits que transitaram pelo canal de transmissão sejam diretamente alimentados no seu estado interno. A responsabilidade do módulo de saída do gerador de chaves de fluxo nos algoritmos de fluxo assíncronos é maior do que nos síncronos. Isto se dá porque o estado interno do gerador pode ser determinado por um intruso a qualquer instante, caso ele tenha acesso aos últimos *bits* da mensagem criptografada que foram transmitidos. A Figura 3.11 ilustra o gerador de chaves de fluxo utilizado nos algoritmos de fluxo auto-sincronizantes.

Uma questão interessante se põe quando pensamos na forma de se encriptar os primeiros  $(n - 1)$  bits da mensagem original. Como cada *bit* da chave de fluxo depende dos últimos  $n$  bits criptografados, não há como se gerar o primeiro *bit* desta chave sem que já se tenha  $n$  bits criptografados. Este problema se resolve facilmente inserindo-se no processo um vetor aleatório de  $n$  bits chamado de vetor de inicialização. Este vetor é inserido no início da mensagem criptografada, servindo apenas para gerar o primeiro *bit* da chave de fluxo com o qual o primeiro *bit* da mensagem original será criptografado. A mensagem criptografada é transmitida para o receptor com este vetor de inicialização à frente. Assim, ao chegar a mensagem criptografada no receptor, este, com o auxílio do

seu gerador da chave de fluxo, poderá gerar o primeiro *bit* da chave de fluxo semelhantemente ao que fez o transmissor da mensagem.

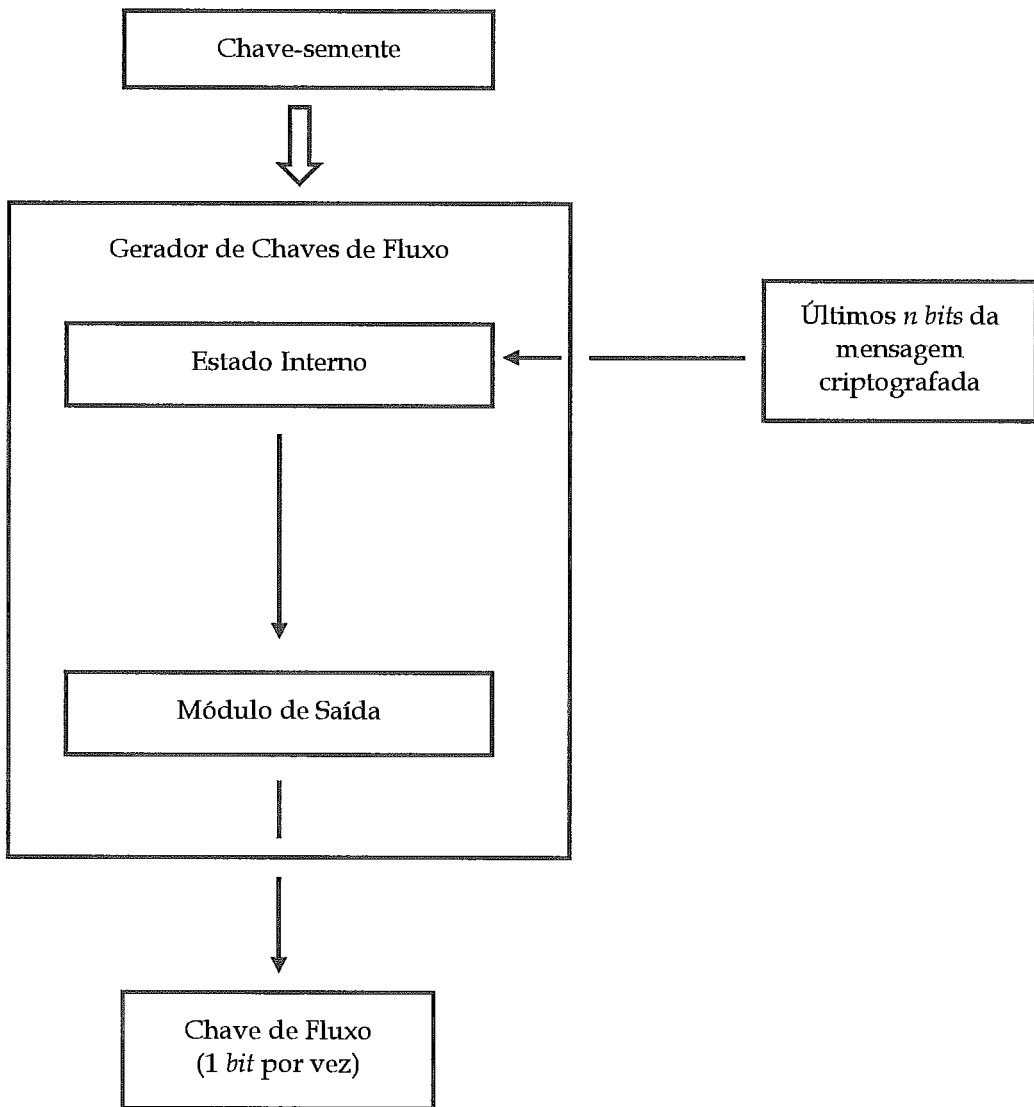


Figura 3.11: O gerador de chaves de fluxo dos algoritmos assíncronos possui apenas dois elementos: Um Módulo de Saída que gera individualmente cada *bit* da chave de fluxo em função do módulo chamado de Estado Interno; o Estado Interno é alterado pelos  $n$  últimos *bits* da mensagem criptografada, permitindo, assim, que o Módulo de Saída produza um novo *bit* para a chave de fluxo. A chave-semente estabelece os parâmetros específicos de funcionamento dos dois elementos, fazendo com que o gerador comporte-se de um modo determinado. Em conseqüência, uma chave de fluxo particular é produzida para cada uma das chaves-semente utilizadas.

A Figura 3.12 representa o funcionamento dos algoritmos de fluxo assíncronos.

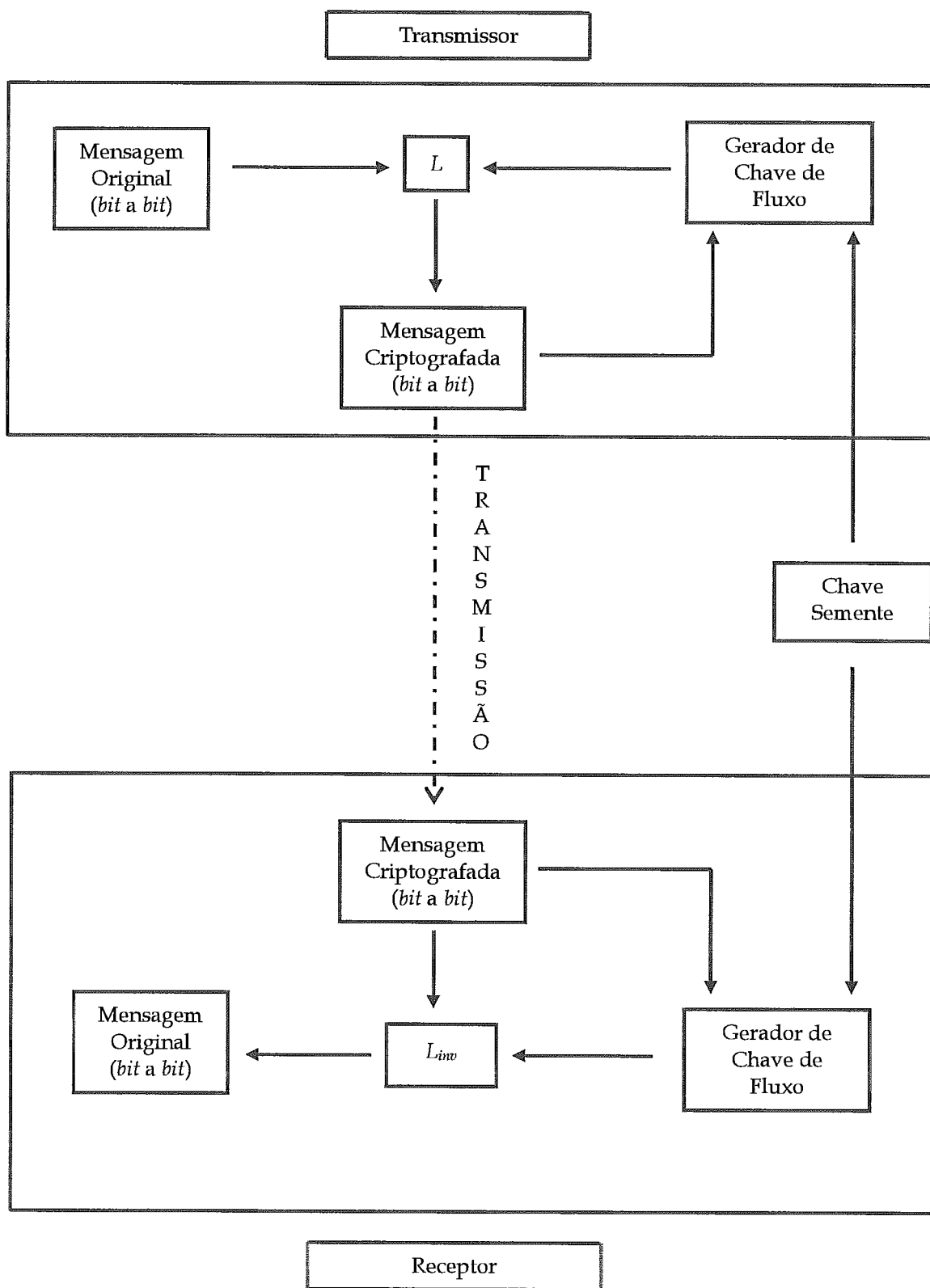


Figura 3.12: Nos algoritmos de fluxo assíncronos, as chaves de fluxo de encriptação e de desencriptação dependem dos *bits* da mensagem criptografada. Como tanto o transmissor quanto o receptor têm acesso a esta mensagem, eles podem gerar chaves de fluxo iguais e sincronizadas sem a necessidade de uma sincronização explícita, como nos algoritmos síncronos. A chave-semente é, obviamente, a mesma, pois algoritmos de fluxo são algoritmos criptográficos simétricos.

Note que se ocorrer um erro de transmissão, no qual um *bit* for perdido ou alterado, haverá um erro de descriptação nos próximos  $n$  *bits*, pois o estado interno do gerador armazena sempre os últimos  $n$  *bits* transmitidos. Após  $n$  *bits* da mensagem criptografada, o estado interno já não armazena mais o *bit* errado, sendo, então, capaz de produzir o *bit* correto da chave de fluxo. É exatamente isto que possibilita aos algoritmos de fluxo assíncronos se re-sincronizarem automaticamente. O custo desta auto-sincronia após erros é a não recuperação de  $n$  *bits* da mensagem criptografada.

## Capítulo 4

### Resfriamento Simulado

Um procedimento computacional da chamada “matemática experimental” muito utilizado é o Método de Monte Carlo [17, 18], nome este relacionado à sua operação de natureza probabilística, que muito se assemelha aos jogos de azar dos cassinos de Monte Carlo, umas das divisões do Principado de Mônaco. Suas bases teóricas são conhecidas há muito tempo, mas devido à dificuldade de aplicação, apenas com a construção dos primeiros computadores eletrônicos se pôde, efetivamente, aplicá-lo com todas as suas capacidades.

A idéia central do Método de Monte Carlo na solução de um problema com incógnita  $q$  é descobrir uma variável  $X$  cuja média  $\hat{X}$  seja igual ao valor da referida incógnita e, sendo assim, pode-se abstrair o problema em si e resolvê-lo, indiretamente, através do cálculo da média desta variável:

$$q = \hat{X}$$

Certamente, a média  $\hat{X}$  pode ser aproximada, tanto quanto se deseja, pelo cálculo da média de um grande número de valores  $x_i$  que, após sorteados através de um mecanismo capaz de gerar a aleatoriedade da variável  $X$  e devidamente contados, se apresentam com frequência  $f_i$ ,  $1 \leq i \leq k$ , ou seja:

$$\hat{X} \approx \frac{\sum_{i=1}^k x_i f_i}{\sum_{i=1}^k f_i} \quad (4.1)$$

Naturalmente, sendo  $p_i$  a probabilidade de ocorrência do valor  $x_i$  na variável aleatória  $X$ , a equação (4.1) pode ser escrita como:

$$\hat{X} \approx \sum_{i=1}^k x_i p_i \quad (4.2)$$

Considere, por exemplo, o Método de Monte Carlo no cálculo da área de uma figura plana  $S$  que pode ser inscrita em um quadrado de lado  $l$ , como indicado na Figura 4.1. Tomando-se duas variáveis aleatórias  $x$  e  $y$ , responsáveis, respectivamente, pela geração equiprovável de abscissas e ordenadas no plano da figura  $S$  no intervalo entre zero e  $l$ , através do sorteio de um número  $k$  de seus valores, podemos contar o número  $N_s$  de pontos contidos no interior da figura  $S$  e, intuitivamente, perceber que a razão entre a sua área e a área do quadrado é proporcional à relação entre o número  $N_s$  e o total de pontos sorteados:

$$\frac{S}{l^2} = \frac{N_s}{k} \quad (4.3)$$

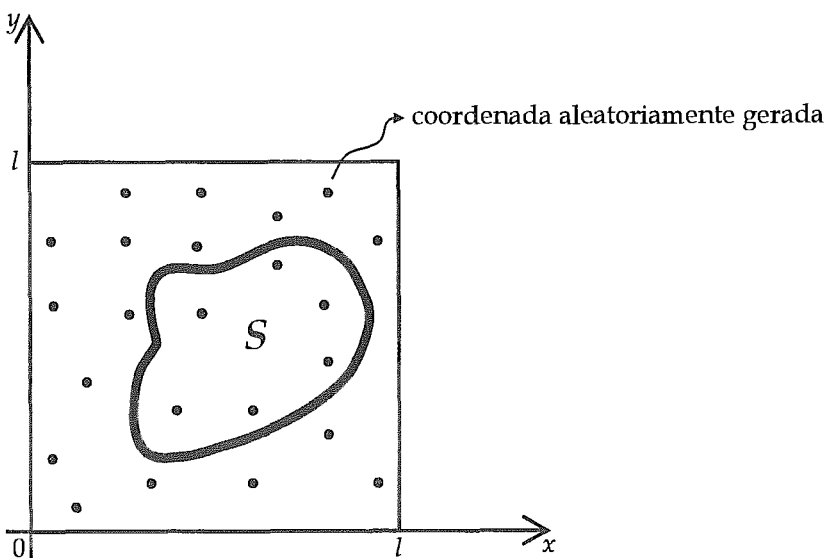


Figura 4.1: O Método de Monte Carlo no cálculo da área  $S$ .

## 4.1 O algoritmo de Metropolis

O Método de Monte Carlo também pode ser utilizado para o cálculo de propriedades de sistemas termodinâmicos como, por exemplo, um gás perfeito em equilíbrio à temperatura  $T$  e contido em um recipiente fechado. Sabe-se da Termodinâmica que, para tal sistema, a probabilidade  $p$  relativa a uma energia  $E$  é dada pela distribuição de Boltzmann:

$$p = e^{-E/KT}, \quad (4.4)$$

onde  $K$  é a constante de Boltzmann. A energia total  $E_t$  deste sistema estatístico é a soma das energias médias de suas  $N$  moléculas, o que em analogia com a equação (4.2) é escrito, desprezando-se a constante de normalização, como:

$$E_t = \sum_{i=1}^N \int_0^{\infty} E_i e^{-E_i/KT_i} dE_i \quad (4.5)$$

Notamos da distribuição de Boltzmann que, para temperaturas muito altas, há uma probabilidade muito próxima da unidade de se encontrar moléculas com todos os níveis energéticos; enquanto que em temperaturas baixas, apenas moléculas com pequenos níveis de energia são detectadas com frequência. Desta forma, para um gás em equilíbrio à temperatura  $T$ , não há a necessidade de considerarmos, para efeito de cálculo da energia média, todos os valores de energia possíveis, mas apenas aqueles que possuem uma probabilidade de ocorrência significativa. Baseado neste princípio, Metropolis [17], em 1953, estabeleceu um método resultante de uma modificação do Método de Monte Carlo, capaz de determinar propriedades termodinâmicas de sistemas mecânicos estatísticos, onde, ao invés da soma da energia de cada molécula ponderada pela sua probabilidade de ocorrência, considerou apenas a soma dos valores energéticos com ocorrência mais provável. Dada uma configuração inicial  $C_0$  do sistema e um

temperatura de equilíbrio  $T_0$ , o algoritmo de Metropolis escolhe aleatoriamente um molécula  $M_i$ ;  $1 \leq i \leq N$  e valores  $\Delta x$  e  $\Delta y$  através dos quais movimenta a molécula  $M_i$  de sua posição atual, gerando uma diferente configuração  $C_k$  do sistema. A variação  $\Delta E_t$  da energia total do sistema entre as duas configurações é calculada por  $\Delta E_t = (E_t(C_k) - E_t(C_{k-1}))$  onde  $E_t(C_k)$  é a energia do sistema na configuração  $C_k$ . Sendo a variação energética  $\Delta E_t$  negativa, a configuração  $C_k$  é imediatamente aceita como a nova configuração do sistema; do contrário a aceitação da configuração recentemente gerada é decidida por sorteio com probabilidade  $p = e^{-\Delta E_t / kT}$  e, no caso de rejeição, a nova configuração é igual à anteriormente existente. Após a determinação da nova configuração, a energia média  $\hat{E}_t$  do sistema é calculada aproximadamente por:

$$\hat{E}_t = \frac{1}{n} \sum_{i=1}^n E_t(C_i), \quad (4.6)$$

na qual  $n$  é o número de configurações aceitas segundo o processo descrito acima. Metropolis demonstrou que seu método escolhe configurações obedecendo à distribuição de Boltzmann, conforme esperado para sistemas estatísticos deste tipo.

É importante notar que, como o algoritmo de Metropolis permite o cálculo da energia de um gás em equilíbrio à temperatura  $T$ , pode-se simular o resfriamento deste sistema até a mudança de fase para o estado líquido e, continuando a redução de temperatura, emular o processo de solidificação. Sabe-se da Cristalografia que, se esta diminuição de temperatura é suficientemente lenta, o sólido formado assume uma estrutura cristalina representante do estado mais organizado conhecido e cuja energia é a menor que tal pode atingir. Nos casos de formação cristalina a partir do vapor com decréscimo de temperatura mais rápido do que o necessário, forma-se uma estrutura sólida conhecida como o “vidro” que é, a grosso modo, um cristal repleto de



imperfeições de organização, e cuja energia é superior à mínima, atingida somente nos cristais.

## 4.2 O algoritmo de Kirkpatrick

Foi observando a semelhança entre a formação de cristais a partir do vapor e os processos de minimização global de funções matemáticas que Kirkpatrick [19], em 1983, estabeleceu o método do Resfriamento Simulado (*Simulated Annealing*), que é basicamente a aplicação sucessiva do método de Metropolis para uma seqüência de temperaturas decrescentes, com o objetivo de gerar uma configuração de energia mínima. Como no presente trabalho, temos o objetivo de chegarmos a uma solução com energia máxima, adotaremos como padrão a maximização da solução, porém, entenda que também é possível minimizá-la. Certamente Kirkpatrick não estava interessado no estudo de formação de cristais, porém, associando cada configuração do sistema termodinâmico de Metropolis à uma solução de um problema de Otimização Combinatória e a energia total do gás à função objetivo do referido problema, percebeu que poderia maximizar globalmente o objetivo, desde que exercesse um correto controle sobre a taxa de resfriamento da simulação.

O Resfriamento Simulado, como consequência do método de Metropolis, se assemelha ao Método do Gradiente ao aceitar, imediatamente, uma solução viável que aumente o valor da função objetivo, porém não apresenta o problema de convergência a máximos locais, pois, para cada temperatura, existe uma probabilidade, dada pela distribuição de Boltzmann, de que a função objetivo diminua seu valor na próxima solução viável gerada (Figura 4.2).

Seja  $D$  um conjunto finito,  $f$  uma função matemática dita função objetivo e  $D \subseteq D^n$  o conjunto das chamadas soluções viáveis, o problema de Otimização Combinatória é:

$$\begin{aligned} & \max f(x) \\ & \text{sujeito a } x \in D, \end{aligned}$$

e estabelecemos o método do Resfriamento Simulado por:

---

Seja  $x_0 \in D$  uma solução viável inicial,

$T_0$  uma temperatura inicial,

e  $r$  uma taxa de decréscimo da temperatura;

$k := 0$  ;  $x_k := x_0$  ;  $T_k := T_0$ ;

**enquanto** o sistema não está “cristalizado” faça

**início**

**enquanto** o sistema não atinge o “equilíbrio térmico” à temperatura  $T_k$  faça

**início**

selecione aleatoriamente uma solução viável  $x' \in D$ ;

$\Delta f := f(x') - f(x_k)$ ;

se  $\Delta f \geq 0$  então  $x_k := x'$ ;

do contrario  $x_k := x'$  com probabilidade  $p = e^{-\Delta f/T_k}$ ;

**fim**;

$T_{k+1} := r(T_k)$  ;  $x_{k+1} := x_k$  ;  $k := k + 1$ ;

**fim**;

---

a solução é  $x_k$ .

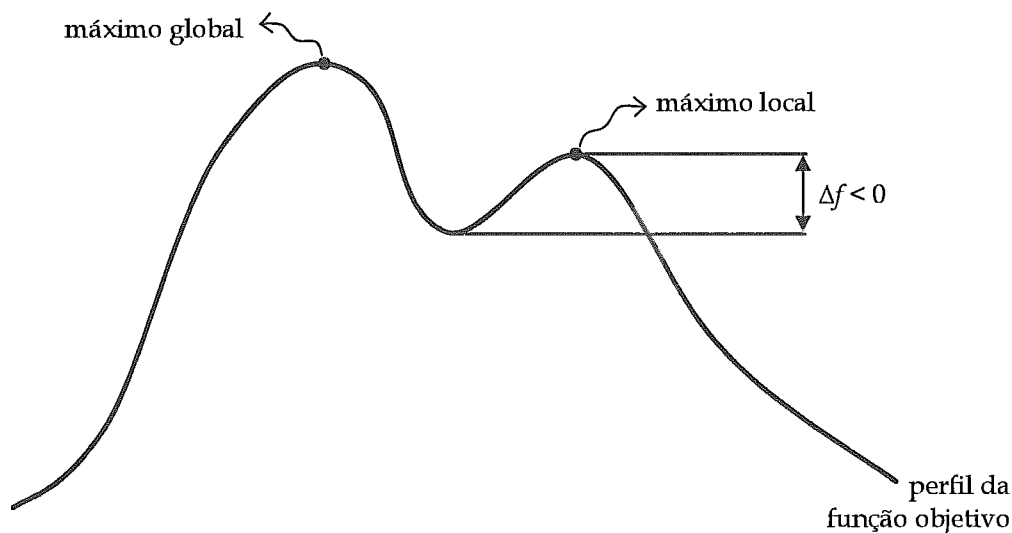


Figura 4.2: Perfil de uma função objetivo com máximo local.

Para a determinação completa do algoritmo é preciso definir quando o sistema está “cristalizado” e atinge o “equilíbrio térmico”, duas características perfeitamente compreensíveis nos sistemas termodinâmicos, porém ausentes e sem análogos nos problemas de Otimização Combinatória. Semelhantemente ao método de Metropolis, no Resfriamento Simulado o sistema atinge o “equilíbrio térmico” quando a média aproximada da função objetivo para uma dada temperatura estabiliza-se, sofrendo apenas pequenas flutuações ao redor de seu valor médio exato e conhecido. Já a “cristalização” do sistema ocorre quando, em analogia ao crescimento de cristais a partir de vapor, ao reduzir-se a temperatura, a média da função objetivo não aumenta, significando, na melhor hipótese, que o sistema atingiu seu máximo global, representante da estrutura cristalina do sistema termodinâmico (Figura 4.3). A taxa de decréscimo da temperatura, também necessária ao método e dependente do problema de Otimização Combinatória considerado, deve ser suficientemente pequena para garantir a “cristalização” do sistema na configuração do máximo global, porém não tão baixa que inviabilize a aplicação do método por requerer um número extremamente grande de configurações simuladas.

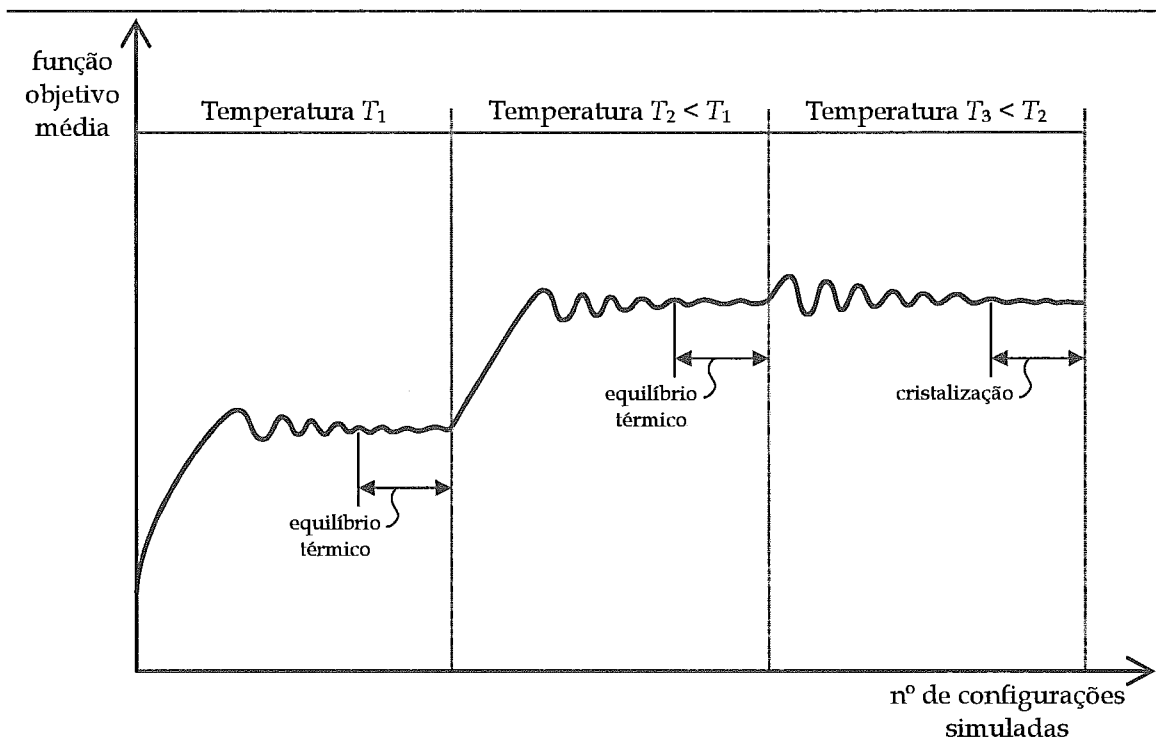


Figura 4.3: Equilíbrio térmico e cristalização no Resfriamento Simulado.

### 4.3 O conceito de entropia

Para medir a homogeneidade de determinados grupos, alguns algoritmos se utilizam do conceito de variância ou de entropia. Na Física, especialmente na Termodinâmica, o conceito de entropia está associado à desordem. Nas leis da Termodinâmica encontramos a afirmação de que qualquer máquina térmica, isto é, qualquer máquina que gere trabalho a partir de calor, só pode funcionar se houver uma fonte de temperatura mais alta e uma fonte de temperatura mais baixa às quais a máquina esteja conectada. Em outras palavras, sempre é preciso uma diferença de temperaturas entre as quais possa fluir energia (calor) de uma fonte quente para uma fonte fria, sendo parte desta energia em fluxo transformada no trabalho útil realizado pela máquina. Uma vez que a célula dos animais e plantas e as estrelas do Universo são máquinas térmicas, a importância desta Lei não deve ser subestimada, pois é a própria essência da vida. Quando estamos em um ambiente quente, nosso corpo transpira eliminando calor,

carregado pelo suor, de forma a manter a diferença de temperatura entre a fonte quente e a fonte fria e permitir o funcionamento das células. O sol e as outras estrelas que emitem calor são fontes quentes de energia. O petróleo é outra fonte quente que usamos em nossas usinas, turbinas e motores. A glicose e os lipídios são outras fontes quentes de energia que nossas células “queimam” para gerar calor e depois trabalho. Muitas são as fontes de calor no Universo e elas possuem diferentes temperaturas. Por isto, o Universo está sempre em constante fluxo energético. Se essa diversidade de temperaturas diminuir, o fluxo energético diminuirá proporcionalmente. Faz parte do processo de gerar trabalho a partir do calor que sempre alguma parte do calor se perca na própria máquina (atrito, aquecimento das partes etc.) não gerando trabalho útil. Essa energia perdida nunca mais será recuperada e chamamos de “entropia”. Entropia é uma medida de diversidade, degeneração, desordem, desorganização e caos. A entropia mínima (zero), representando a organização total, é definida como aquela de um cristal de forma geométrica perfeita a uma temperatura de zero absoluto na qual nenhum átomo se “movimenta”. A entropia zero significa, portanto, inércia, ordem total e morte. A cada dia fluem muitas formas de energia das fontes quentes para as fontes frias do Universo. E, sendo assim, as fontes quentes se esfriam (ou se esgotam) e as fontes frias se esquentam até que as temperaturas se igualem. Esses fluxos de energia têm suas entropias, pois sempre há alguma energia perdida não transformada em trabalho útil. Dessa forma, a entropia do Universo aumenta sempre até alcançarmos a desordem ou degeneração total. Neste dia, todas as temperaturas se igualarão e nenhum trabalho poderá ser gerado no Universo. O Universo terá morrido! Concluindo, a entropia é uma medida de desordem e degradação. Seu conceito é muito importante para a Física e também para a Ciência da Computação.

O conceito de entropia como energia perdida na Termodinâmica foi elegantemente estendido para o de quantidade de informação na Ciência da Computação. Seja a informação de que “o Sol nascerá amanhã”. Qual seria a “quantidade” ou o “valor” desta informação para qualquer um de nós? Se o Sol nasce mesmo todos os dias, esta informação não significa nada! O fenômeno do nascer do Sol é tão regular, organizado e entendido que não necessitamos de nenhuma outra informação complementar. Portanto, a frase “o Sol nascerá amanhã” possui quantidade de informação nula. Mas e a frase “o Sol nascerá amanhã trinta segundos mais cedo do que hoje”? Esta frase já possui um valor informacional maior, pois se o fenômeno do nascer do Sol é regular e organizado a um nível macroscópico, ele não o é a um nível mais detalhado e microscópico. Se o Sol nascerá amanhã trinta segundos mais cedo que hoje, é sinal de que estamos caminhando em direção ao verão. Isto é uma informação adicional que não existia antes! Se medirmos o horário do nascer do Sol todos os dias, veremos que o fenômeno não é tão regular e organizado quanto pode parecer a princípio como na inútil frase “o Sol nascerá amanhã”. Sistemas organizados (como um cristal perfeito a zero grau absoluto) são tão previsíveis que não necessitam de informação nenhuma para seu entendimento. Ou seja, sistemas com entropia zero não necessitam de informação, pois já são entendidos. Porém, se o fenômeno é complexo, como o horário que o Sol nasce a cada dia, torna-se mais difícil entendê-lo e, conseqüentemente, qualquer informação sobre o fenômeno passa a ter valor. Em outras palavras, sistemas desorganizados, caóticos, com muita entropia, necessitam de muita informação para seu esclarecimento. Chamamos de entropia da informação a quantidade de informação adicional necessária para se entender um fenômeno ou sistema.

A quantidade de informação  $Q$  sobre um fenômeno é medida pela sua probabilidade de ocorrência  $p$ . Quanto mais improvável o fenômeno (baixa probabilidade de

ocorrência), maior o valor da informação capaz de prevê-lo. A relação entre a quantidade da informação  $Q$  e a probabilidade  $p$  é inversa:

$$Q = -\log_2 p \quad (4.7)$$

Se a probabilidade do Sol nascer amanhã é 1 (total certeza do fato), a quantidade  $Q$  desta informação é nula pois o logaritmo de 1 é zero. Porém, se tentarmos prever em que horário o Sol nascerá amanhã, nossa probabilidade de acerto será pequena e, conseqüentemente, o valor  $Q$  será grande, pois o logaritmo de um número pequeno é um número grande – negativo, é claro, mas que multiplicado pelo sinal de menos presente na definição de  $Q$ , resulta em um número positivo e grande. Concluindo, a quantidade de informação necessária para entendermos que o Sol nascerá amanhã é zero, pois já sabemos disto. No entanto, a quantidade de informação necessária para sabermos o horário exato do nascimento do Sol é grande, pois o fenômeno é dificilmente previsível.

Definimos a entropia da informação de um fenômeno ou conjunto de eventos  $X = \{E_1, E_2, \dots, E_n\}$  como a quantidade média de informação necessária para representar este conjunto, com probabilidades associadas e distintas. Se um conjunto contém  $n$  fatores ou eventos  $E_i$ , onde  $P(E_i) = p_i$ ;  $i = 1, 2, 3, \dots, n$ , a entropia da informação sobre este conjunto é a média da quantidade de informação que necessitamos para prever cada evento:

$$H(X) = -\sum_{i=1}^n p_i \log_2 p_i \quad (4.8)$$

onde  $p_i$  é probabilidade de ocorrência do evento  $E_i$ . O logaritmo usado é na base dois, pois estamos trabalhando com *bits*.

Se um fenômeno complexo e desorganizado depende de vários eventos de difícil previsão, a quantidade de informação para prevermos cada evento é alta e, certamente, a

média destas quantidades também será alta, resultando em uma grande entropia da informação. A entropia é máxima quando a distribuição (probabilidade dos eventos) é uniforme, ou seja, os eventos são equiprováveis. Por outro lado, um fenômeno que dependa de muitos eventos facilmente previsíveis, terá uma entropia de informação baixa, pois a média de baixas quantidades de informação será um número pequeno.

Se na Termodinâmica, a entropia é a perda presente de trabalho útil irrecuperável no futuro, na Ciência da Computação, a entropia é a falta de conhecimento no presente que deve ser suprida no futuro. Na Termodinâmica, a entropia é a desordem por degradação enquanto na Ciência da Computação a entropia é a desordem por falta de conhecimento, ou imprevisibilidade [20].

É justamente essa imprevisibilidade que necessitaremos para gerar a chave de cifra utilizada em nosso sistema, como será visto no próximo capítulo. A entropia do autômato celular será utilizada como função objetivo do algoritmo do Resfriamento Simulado.

## 4.4 Algoritmos Genéticos

O princípio básico dos Algoritmos Genéticos (AGs) [21] baseia-se em uma analogia direta com a Genética Natural, manipulando conceitos como população de indivíduos, que representam, por sua vez, possíveis soluções do problema a ser resolvido. A cada indivíduo é associado um valor de adaptação (*fitness*), e para aqueles com maior adaptação é dada a oportunidade de reprodução. São selecionados os melhores indivíduos daquela geração. Esses indivíduos são então submetidos a operações como a *crossing-over* (um novo indivíduo é criado através da combinação de partes de dois ou mais indivíduos) e mutação (um novo indivíduo é criado por pequenas mudanças arbitrárias sobre os indivíduos). Os AGs podem ser aplicados em diversas áreas. A área



de Otimização Combinatória é naturalmente “contemplada” por este algoritmo, já que a tarefa de gerar um indivíduo de melhor adaptação pode ser associada à tarefa de encontrar a melhor solução para um problema.

Apesar de aparentemente o Resfriamento Simulado não guardar nenhuma relação em sua metáfora física com o Algoritmo Genético em sua metáfora biológica, ambos são realmente matematicamente muito semelhantes. A “temperatura” inicial do Resfriamento Simulado e, conseqüentemente, a probabilidade de aceitação de uma configuração nova que provoque diminuição no valor da função objetivo, é análoga aos procedimentos de mutação e “*crossing-over*” responsáveis pela geração de novos cromossomos que podem diminuir o valor da função de adaptação. Os sorteios de parâmetros de controle e variáveis do problema são também utilizados em ambos os procedimentos e fazem com que os dois métodos se classifiquem como métodos da matemática experimental, assim como o Método de Monte Carlo. Talvez, a grande diferença entre estas duas heurísticas genéricas seja a facilidade de implementação computacional do Resfriamento Simulado em relação ao Algoritmo Genético.

# Capítulo 5

## O Método proposto

Em [22, 23, 24], Tomassini fez experimentos para gerar chaves para uma criptografia de fluxo usando autômatos celulares bastante “entrópicos” gerados a partir de Algoritmos Genéticos. Nessa dissertação, propomos um método que, ao invés dos AGs, utiliza o algoritmo do Resfriamento Simulado.

Nesse capítulo apresentamos o método proposto, seguido de seus resultados. Após, são mostrados experimentos, feitos a partir da aplicação do método, comparando os resultados considerados “ótimos” pelo método proposto e os considerados satisfatórios por Tomassini.

### 5.1 O método

O método que propomos foi a geração de chaves de criptografia de fluxo a partir da seleção de autômatos celulares “ótimos” escolhidos através da aplicação da meta-heurística do Resfriamento Simulado.

Os experimentos foram realizados com autômatos celulares unidimensionais de 8 *bits* e vizinhança de von Neumann e tamanho  $\delta = 3$ , ou seja, os vizinhos de uma célula considerados são a célula à sua esquerda, ela própria e a célula à sua direita. Os

autômatos utilizados são não-uniformes, isto é, a regra de cada célula não necessariamente é idêntica à das outras. Com isso, para cada simulação temos 8 regras distintas, uma para cada célula, fazendo, assim, com que a solução possa se tornar ainda mais aleatória.

Como foi mencionado no capítulo anterior, utilizamos o valor da entropia para ser a função objetivo do Resfriamento Simulado, uma vez que entropia é imprevisibilidade, ou seja, uma excelente forma de determinarmos se o autômato gerado segue algum “padrão”, isto é, é fácil de sabermos qual será o comportamento do autômato nas próximas transições de tempo, ou se é aleatório suficiente para ser uma chave de criptografia [25]. A entropia foi calculada gerando-se 1024 passos do autômato celular, de acordo com suas regras de transição.

Seja  $i_j$  a  $j$ -ésima subsequência de tamanho  $i$ ;  $j = 1, 2, 3, \dots, k^i$ . A probabilidade desta sequência é dada por  $P(i_j)$ . Logo a entropia  $H_i$  para o este conjunto de seqüências é dada por:

$$H_i = -\sum_{j=1}^{k^i} P(i_j) \log_2 P(i_j) \quad (5.1)$$

onde  $i_1, i_2, \dots, i_{k^i}$  são todas as possíveis subsequências de tamanho  $i$ .

O algoritmo utilizado foi o seguinte:

1. Gerar oito bits aleatórios (estado inicial do autômato) e oito regras aleatórias de oito bits (entre 0 e 255) para este autômato;
2. A partir desta configuração inicial, escolher uma “temperatura” inicial alta e determinar um número mínimo de iterações por “temperatura”;
3. Gerar 1024 passos do autômato, de acordo com suas regras de transição, e calcular a sua entropia;

4. *Selecionar, aleatoriamente, uma das oito regras do autômato e um novo valor para ela, também de oito bits (entre 0 e 255), e calcular a entropia desse novo autômato, gerando 1024 passos dele;*
5. *Calcular a diferença  $\Delta$  entre o valor da entropia do autômato com a regra sorteada alterada e o autômato com o valor da regra sorteada original:*
  - a. *Se  $\Delta$  for positivo, aceitar a nova configuração do autômato com a regra escolhida alterada como sendo o seu valor efetivo a partir deste instante;*
  - b. *Se  $\Delta$  for negativo, calcular a probabilidade  $p = e^{-\Delta/KT}$ , onde  $K$  é a constante de Boltzmann, de aceitar a nova configuração do autômato como possível. Isto é feito sorteando-se um número aleatório e verificando se ele cai entre zero e o valor  $p$  recentemente calculado:*
    - i. *Se o número aleatório está dentro do intervalo zero e  $p$ , a nova configuração do autômato é aceita como o efetivo a partir de então;*
    - ii. *Do contrário esta configuração é rejeitada e a regra sorteada retoma seu valor original de antes do sorteio;*
6. *Calcular o valor da diferença das médias da entropia atual e da entropia há  $x$  (mínimo de iterações por temperatura) passos atrás, ou seja, verificar se o sistema atingiu o “equilíbrio térmico”:*
  - a. *Se a diferença for menor que um dado valor (parâmetro  $\epsilon$ ), diminuir a “temperatura” levemente (parâmetro  $\alpha$ ) e retornar ao passo 4;*
  - b. *Senão, somente retornar ao passo 4, com o mesmo valor da “temperatura”;*

7. O término do processo ocorre quando a “temperatura” é próxima de zero (parâmetro temperatura mínima). A configuração do autômato com suas regras neste ponto será a solução do problema, o máximo global da função, ou seja, a entropia máxima do autômato.

Utilizamos como verificação da “cristalização”, para o término do processo, o parâmetro “temperatura” mínima, ou seja, independentemente se a média da entropia de uma “temperatura” foi ou não superior ou igual à média da entropia da “temperatura” anterior, o sistema somente pára quando se atinge essa “temperatura” mínima. Isso para que o simulador gere mais soluções ótimas, e não pare na primeira encontrada.

A determinação da “temperatura” inicial [26] e dos outros parâmetros do problema (decrécimo da temperatura ( $\alpha$ ), “temperatura” mínima, número mínimo de iterações por “temperatura” e a diferença das médias das entropias ( $\epsilon$ )) é de suma importância para a obtenção de resultados ótimos. Exhaustivos testes resultaram nos seguintes valores para esses parâmetros:

- *Temperatura inicial: 0,1;*
- *Decrécimo da temperatura ( $\alpha$ ): 0,95;*
- *Temperatura mínima: 0,001;*
- *Número mínimo de iterações por temperatura: 200;*
- *Diferença das médias ( $\epsilon$ ): 0,01.*

A infra-estrutura utilizada nos experimentos foi composta de um computador pessoal (PC), tendo um processador AMD Sempron™ 2800+ de 2.0GHz com 512 megabytes de memória principal. O simulador foi desenvolvido na plataforma Borland Delphi 5.

## 5.2 Resultados

Com os parâmetros especificados na seção anterior, em aproximadamente 65% dos casos, e num tempo médio de 1 hora, foram obtidas soluções consideradas ótimas (valor da entropia superior a 7,8). A Tabela 5.1 contém 50 exemplos dessas soluções.

A Figura 5.1 exemplifica uma dessas soluções ótimas. O autômato cujas regras são 150, 85, 153, 105, 85, 150, 165 e 90, respectivamente e entropia 7,9937 se enquadra na solução de maior entropia que conseguimos chegar. Estão representados 100 passos da simulação, desprezando-se a configuração inicial aleatória.

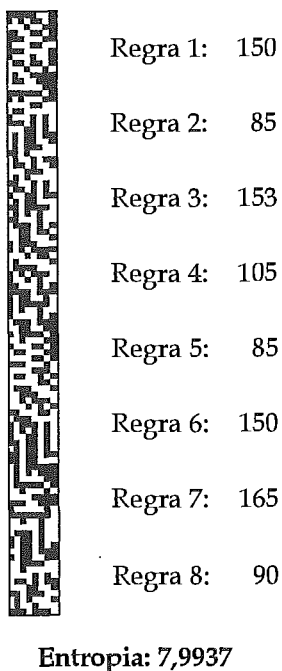


Figura 5.1: Simulação com as regras 150, 85, 153, 105, 85, 150, 165 e 90, respectivamente. Com entropia 7,9937, o autômato pode ser classificado como caótico. Estão representados 100 passos da simulação.

A chave-semente  $S$  será formada, então, pelo vetor com as 8 regras de atualização  $R_i$  e o os 8 *bits* aleatórios do estado inicial do autômato  $C(0)$ , ou seja,  $S = \langle R_i, C(0) \rangle$ ,  $1 \leq i \leq 8$ . Com isso, temos uma chave-semente relativamente pequena que, quando colocada no gerador de chave de fluxo gera a chave  $K$  e esta tem 8192 *bits*, e é pseudo-aleatória.

Regra 1	Regra 2	Regra 3	Regra 4	Regra 5	Regra 6	Regra 7	Regra 8	Entropia
85	86	90	170	101	170	170	106	7,9937
89	85	150	153	165	150	170	170	7,9937
90	102	90	106	90	170	29	166	7,9937
90	149	89	153	166	149	170	106	7,9937
102	85	89	102	90	102	90	105	7,9937
102	101	85	102	90	153	86	150	7,9937
106	169	85	86	90	153	105	170	7,9937
166	170	102	153	154	150	170	85	7,9937
169	149	90	170	86	149	170	106	7,9937
169	165	89	153	154	150	170	85	7,9937
150	150	45	105	106	15	85	105	7,9937
90	102	90	105	90	170	170	90	7,9874
165	51	165	75	105	212	105	106	7,9874
89	165	51	150	90	150	30	57	7,9874
105	105	105	90	105	120	51	153	7,9874
150	150	71	99	150	165	85	105	7,9874
89	240	51	105	90	150	30	57	7,9748
89	240	51	150	165	150	225	105	7,9748
90	105	90	150	170	15	197	105	7,9748
150	105	90	108	170	15	90	150	7,9748
165	150	90	150	170	15	197	105	7,9748
150	150	135	54	106	15	85	105	7,9686
150	150	71	99	86	165	85	105	7,9623
51	85	150	105	90	240	85	210	7,9497
51	108	150	90	86	240	85	180	7,9497
51	198	169	90	149	240	90	210	7,9497
58	195	90	225	147	90	15	51	7,9497
60	150	150	105	90	30	85	120	7,9497
90	102	90	165	90	170	29	89	7,9497
105	54	90	15	85	210	51	170	7,9497
150	150	45	99	106	165	85	165	7,9497
163	195	90	225	57	105	15	51	7,9497
195	102	105	90	86	240	85	225	7,9497
150	150	45	99	106	165	85	105	7,9434
169	165	90	170	101	89	170	106	7,9245
149	102	90	154	150	170	170	89	7,9183
85	86	154	170	101	170	170	106	7,9120
90	102	90	106	90	170	170	90	7,9057
150	105	75	54	170	15	85	105	7,8994
169	86	90	170	101	89	170	106	7,8931
90	102	90	106	90	149	170	90	7,8805
150	105	180	60	170	15	85	105	7,8742
90	102	90	165	150	170	165	149	7,8617
85	86	105	154	150	170	170	149	7,8554
90	102	90	165	150	170	102	149	7,8491
58	60	85	225	147	90	15	51	7,8491
83	195	85	225	147	90	15	51	7,8491
85	102	90	154	150	170	86	149	7,8365
85	102	90	154	150	170	170	149	7,8239
169	86	90	170	101	170	170	106	7,8177

Tabela 5.1: Exemplos de soluções ótimas (entropia superior a 7,8)

Com os resultados obtidos, concluímos que não importa somente descobrirmos regras consideradas “boas”, mas também em quais posições essas regras estarão. Para demonstrar essa necessidade, realizamos alguns testes com as soluções satisfatórias. Sorteamos as posições dos conjuntos de regras dessas soluções e as colocamos na Figura 5.2.

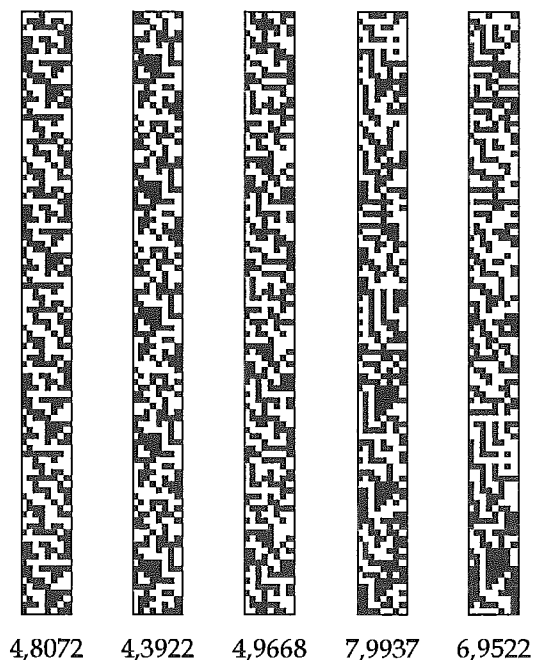


Figura 5.2: Resultados obtidos usando as regras 150, 85, 153, 105, 85, 150, 165 e 90 em posições aleatórias. Todos, também com 100 passos. Abaixo dos passos, o valor da entropia em cada simulação.

Observando essa enorme importância que a posição das regras tem, tentamos criar uma heurística com as posições mais frequentes. Para isso criamos a Tabela 5.2, que mostra as regras obtidas, juntamente com suas posições, de 471 exemplos de soluções ótimas. As regras que não constam na tabela, não aparecerem em nenhuma das soluções.

De posse desta tabela, simulamos algumas combinações de regras e posições, tentando sempre pegar regras que aparecem com frequência na posição presente na tabela. Os resultados podem ser vistos na Figura 5.3.



Nº Regra	Posição							
	Regra 1	Regra 2	Regra 3	Regra 4	Regra 5	Regra 6	Regra 7	Regra 8
15	0	0	0	17	0	37	28	0
29	0	0	0	0	0	0	4	0
30	0	0	0	1	0	10	4	4
45	0	0	5	1	0	7	0	0
51	54	1	9	0	0	0	51	48
53	0	0	0	0	0	0	3	5
54	0	4	0	31	4	0	0	2
57	7	1	0	1	6	0	0	8
58	17	2	0	0	0	0	0	9
60	16	19	0	2	0	0	118	65
71	0	0	5	0	0	0	0	0
75	0	0	2	2	0	0	0	0
83	4	0	0	0	0	0	2	0
85	21	31	38	9	33	6	81	10
86	0	9	5	12	29	0	4	2
89	12	0	16	0	25	4	0	4
90	26	0	105	87	83	60	18	13
92	0	0	0	1	0	0	0	0
99	2	0	0	21	4	0	0	0
101	0	6	0	1	9	0	0	2
102	25	62	6	14	0	2	1	10
105	100	69	44	64	49	56	23	61
106	5	0	0	3	10	0	0	25
108	15	9	0	3	0	0	0	0
120	0	0	0	1	0	38	2	27
135	0	0	30	0	0	12	0	0
147	6	0	0	0	14	0	0	0
149	1	18	0	0	36	8	0	12
150	68	67	113	50	76	68	0	21
153	1	18	0	34	8	17	1	8
154	0	2	1	9	7	0	3	1
156	3	0	0	1	0	0	0	0
163	7	2	0	0	0	0	1	0
165	32	12	39	23	36	23	18	1
166	5	0	1	0	4	1	0	2
169	14	3	6	2	2	0	4	3
170	15	64	39	35	32	23	86	54
180	0	0	4	8	0	2	4	14
195	6	46	3	1	0	0	7	27
197	0	0	0	0	0	0	3	5
198	6	18	0	0	0	0	0	0
201	3	0	0	0	0	0	0	2
210	0	0	0	10	3	32	0	15
212	0	0	0	0	0	1	0	0
225	0	0	0	23	0	0	5	11
240	0	8	0	4	1	64	0	0

Tabela 5.2: As regras e suas posições de 471 exemplos de soluções ótimas.

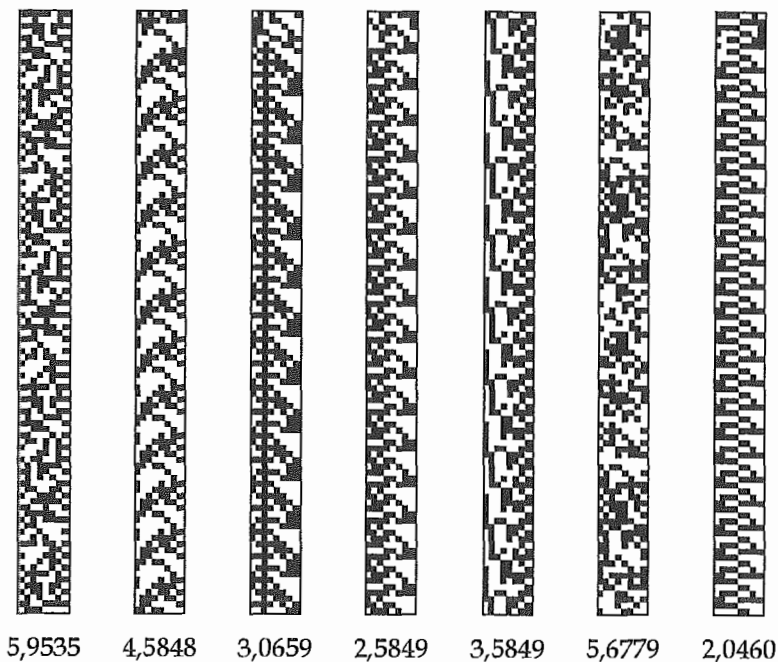


Figura 5.3: Resultados obtidos escolhendo regras a partir da tentativa de criação de uma heurística das posições das soluções ótimas encontradas na Tabela 5.2. Em todas as simulações estão representados 100 passos e abaixo deles, o valor de cada entropia.

Como podemos observar pelos valores das entropias encontrados nas simulações, nem mesmo tentando criar uma lista com as regras mais obtidas nas respectivas posições é suficiente para encontrarmos soluções ótimas, ou seja, para se conseguir soluções aceitáveis, temos que submeter o autômato ao procedimento que utilizamos, isto é, fazer o uso do algoritmo do Resfriamento Simulado.

### 5.2.1 Comparação com Algoritmos Genéticos

Tomassini chegou à conclusão de que apenas combinando as regras 90, 105, 150 e 165 em posições sortidas, se obtém uma solução satisfatória. As quatro regras obtidas pelos testes realizados com os AGs são consideradas boas (estão presentes em grande parte das soluções ótimas), porém não são suficientes para se chegar a um resultado satisfatório. Devemos combiná-las juntamente com outras regras. Como vimos na seção anterior, o resultado usando o *Simulated Annealing* (SA) mostrou que a posição das regras também é de fundamental importância, ou seja, não adianta, como concluiu

erroneamente Tomassini, pegamos as regras consideradas boas e escolhemos, aleatoriamente, posições para elas. A Figura 5.4 mostra alguns resultados obtidos sorteando as regras propostas por Tomassini em posições sortidas.

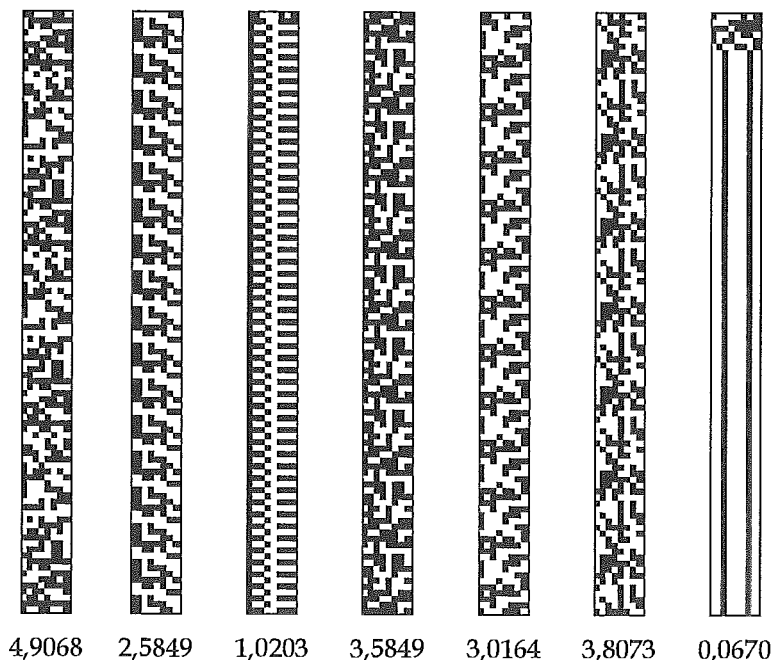


Figura 5.4: Resultados obtidos usando as regras 90, 105, 150 e 165 em posições aleatórias. Todos, também com 100 passos. Abaixo dos passos, o valor da entropia em cada simulação.

### 5.3 Experimentos

Utilizamos o software do NIST (National Institute of Standards and Technology) para verificar a qualidade da aleatoriedade das seqüências geradas pela aplicação.

O NIST Statistical Test Suite [27] é um pacote contendo 16 testes que foram desenvolvidos para testar a aleatoriedade de seqüências binárias produzidas por softwares baseados em geradores de números aleatórios ou pseudo-aleatórios para criptografia. Esses testes focam numa variedade de diversos tipos de não-aleatoriedade que possam existir em uma seqüência.

Como nossas seqüências (chaves geradas pela simulação) são extremamente pequenas (8192 *bits*) só conseguimos realizar 3 dos 16 testes existentes. São eles:

- Teste de frequência bit a bit (*Frequency (Monobits) Test*): o foco desse teste é a proporção de zeros e uns na seqüência. O propósito desse teste é determinar quando o número de zeros e uns na seqüência são aproximadamente o mesmo que se esperaria de uma verdadeira seqüência aleatória. O teste baseia-se na fração  $\frac{1}{2}$ , isto é, o número de uns e zeros na seqüência deve ser o mesmo.
- Teste de frequência com bloco de bits (*Test for Frequency within a Block*): o foco desse teste é a proporção de zeros e uns com blocos de  $M$  bits. A finalidade desse teste é determinar quando a frequência de um bloco de  $M$  bits é aproximadamente  $M/2$ . Nos testes realizados, utilizamos  $M = 8$ , ou seja, bloco de 8 bits.
- Subseqüência de bits iguais (*Runs Test*): o foco desse teste é o número total de zeros e uns corridos em uma seqüência, aonde uma corrida é uma seqüência de bits idênticos ininterrupta. Uma corrida de tamanho  $k$  significa que ela tem exatamente  $k$  bits idênticos e é rodeada antes e depois com o bit de valor oposto. O propósito do teste de corridas é determinar quando o número de corridas de uns e zeros de vários tamanhos é o que se espera de uma seqüência aleatória. Em particular, esse teste determina quando a oscilação entre essas subseqüências é muito rápida ou muito devagar.

Utilizamos diversas seqüências em nossos experimentos. Vamos exemplificá-los utilizando uma amostragem de três seqüências. A seqüência 1, formada pelas regras 106, 169, 85, 86, 90, 153, 105 e 105, respectivamente, e cuja entropia é 7,9937, a seqüência 2, com as regras 90, 165, 150, 85, 105, 153, 85 e 150, respectivamente, e com entropia 3,8148, e a seqüência 3, que tinha as regras 105, 90, 90, 165, 150, 165, 150 e 90, respectivamente, e a entropia calculada em 3,5849. Como pode ser visto, a seqüência 1 foi considerada em nossos resultados uma solução ótima. Realizamos testes com a segunda seqüência para provar a importância das posições das regras. Já a seqüência de número 3 foi considerada aceitável pelos resultados obtidos através da simulação com Algoritmos Genéticos.

Realizamos os três testes (frequência, frequência de blocos e subsequências iguais) de três formas diferentes. Na primeira, consideramos a sequência inteira, ou seja, 8192 bits. No segundo modo, dividimos a sequência em 8 subsequências de 1024 bits. E na terceira, e última, forma, consideramos 64 subconjuntos de 128 bits. Isso para assegurar que as subsequências de cada resultado também são aleatórias.

A única sequência que obteve sucesso em todos os testes foi a sequência 1. A segunda sequência não passou nos testes de frequência da sequência inteira e subsequências iguais na sequência inteira, sequências de 1024 bits e sequências de 128 bits. Já a sequência de número 3 não obteve resultado positivo nos testes de frequência da sequência inteira e sequências de 1024 bits e subsequências iguais na sequência inteira e sequências de 1024 bits.

## 5.4 VoIP

A comunicação de voz em redes IP, chamada de VoIP, consiste no uso das redes de dados que utilizam o conjunto de protocolos das redes IP (TCP/UDP/IP) para a transmissão de sinais de voz em tempo real na forma de pacotes de dados [28]. Nos sistemas tradicionais, o sinal de voz utiliza uma banda de 4 kHz, e digitalizado com uma taxa de amostragem de 8 kHz para ser recuperado adequadamente (Princípio de Nyquist). A banda de comunicação de dados utilizada pela aplicação de voz sobre IP depende do tipo de codificador usado, o que leva a um consumo maior ou menor de banda por cada canal de voz ativo. No caso de menor consumo, que atende à recomendação ITU-T G.729 (CS-CELP), cada canal de voz necessita de uma banda de 8 kbits/s.

Assim sendo, necessitaríamos de, no mínimo, uma chave de criptografia de 8 kbits para criptografar um segundo de VoIP, ou seja, as chaves obtidas pela simulação

proposta seriam suficientes para criptografar pouco mais de um segundo de comunicação.

Para se tentar manter um fluxo de comunicação constante, teríamos que trocar de chave a, aproximadamente, cada segundo. Não poderíamos ficar utilizando a mesma chave por diversas vezes, senão a seqüência pseudo-aleatória não passaria no teste de frequência com blocos de 8192 *bits*. Com isso, uma solução seria trocar de chave a cada segundo, seguindo a tabela de soluções ótimas. Mas isso também não é muito interessante, já que ficar trocando de chave uma vez por segundo gera atrasos na comunicação, promovendo o que chamamos de eco.

Com isso, cria-se a necessidade de simularmos autômatos celulares maiores, de 16 *bits* ou mais. Para soluções com ACs de 16 *bits* teríamos, aproximadamente, 524 segundos de transmissão por chave, sem compressão. Para ACs de 32 *bits*, conseguiríamos transmissões de mais de 18h para uma única chave. Discutiremos a viabilidade da implementação dessas simulações no próximo capítulo.

# Capítulo 6

## Considerações finais

Nesse trabalho, nós abordamos a utilização de autômatos celulares como sendo chave de cifras de fluxo. Para gerar autômatos que sejam boas soluções como chave, ou seja, a chave seja aleatória suficiente, utilizamos o algoritmo de Resfriamento Simulado.

Fizemos o uso de autômatos celulares não-uniformes, usando 8 regras distintas, uma para cada célula, para tornar a solução ainda mais aleatória.

Realizamos exaustivos testes, apresentados no Capítulo 5, os quais nos mostram exemplos de soluções, com as regras e suas posições. Com isso, podemos facilmente observar que não é necessário somente acharmos regras “boas” e colocá-las sortidamente em qualquer posição. Para gerarmos boas seqüências aleatórias para serem chaves de uma criptografia de fluxo, devemos descobrir uma combinação de regras e as posições corretas para cada uma delas.

Dessa forma, encontrar boas soluções não é uma tarefa simples, como propôs Tomassini na sua solução utilizando Algoritmos Genéticos, que concluiu que apenas sorteando uma regra dentro do conjunto  $\{90, 105, 150, 165\}$  para qualquer posição, se obtém uma solução aceitável.

Vale lembrar que realizamos testes utilizando autômatos celulares unidimensionais de 8 *bits*, que nos permitiu “rodar” 1024 transições de tempo sem que a seqüência começasse a se repetir, gerando uma chave de 8192 *bits*, desprezando a configuração inicial. É uma solução extremamente pequena, mas que já nos prova que obtivemos um resultado correto, provando, assim, que, agora, somente necessitaríamos gerar autômatos maiores (16 *bits*, 32 *bits* e casos bidimensionais) que, na prática, poderiam ser utilizados em casos reais.

Cabe a observação de que os algoritmos de fluxo requerem operações em nível de *bit*, o que costuma ser computacionalmente caro.

Desenvolvemos, também, a simulação com autômatos de 16 *bits*. Essa simulação geraria, teoricamente, uma seqüência de 4194304 *bits* sem repetição se a solução fosse “ótima”, ou seja, 512 vezes maior do que a solução de 8 *bits*. Porém, não conseguimos atingir resultados satisfatórios, uma vez que a aplicação ficou muito “pesada”. Uma alternativa seria tentarmos desenvolver o projeto em uma outra linguagem de programação, utilizando menos recursos visuais, ou rodarmos em um sistema de *cluster*. Talvez conseguíssemos chegar a resultados satisfatórios.

Para uma simulação de 32 *bits*, que geraria chaves de 536870912 *bits*, 128 vezes maior que a de 16 *bits* e, conseqüentemente, 65536 vezes maior que a de 8 *bits*, acreditamos que, talvez, nem mesmo a troca de ambiente seria suficiente para conseguirmos obter os resultados. A verificação da aleatoriedade da seqüência deveria ser feita de outra forma e não mais utilizando a entropia como forma de cálculo. Alguns, ou uma combinação de vários, dos testes do Instituto NIST seriam ótimos candidatos para isto.

Em casos de autômatos bidimensionais poderíamos ter chaves ainda maiores. Para o caso de autômatos 8 x 8, teríamos 64 regras distintas e chegaríamos a chaves bastante



grandes. Alguns estudos [29] mostram que para casos de duas dimensões a qualidade das seqüências é maior que para os de apenas uma dimensão.

O que vale lembrar, também, é que, com chaves maiores conseguiríamos realizar todos, ou quase todos, os testes do NIST. Existem outros institutos que possuem excelentes testes de verificação da aleatoriedade de uma seqüência aleatória ou pseudo-aleatória. O Diehard é um exemplo.

O que queremos enfatizar, principalmente, com o desenvolvimento do método proposto é a grande dificuldade de encontrar um modo de obter seqüências que sejam aleatoriamente suficientes para serem boas chaves para uma cifra de fluxo. É nesse ponto da literatura que o presente trabalho se encaixa.

# Referências Bibliográficas

- [1] A. Kirkby, Cryptography And E-Commerce: A Wiley Tech Brief, *Network Security*, 4: 9-9, 2001.
- [2] R. Brisson, and F. Théberge, An Overview of the History of Cryptology, *Communications Security Establishment, Canada*, 2000.
- [3] D. Davies, A Brief History of Cryptography, *Information Security Technical Report*, 2:14-17, 1997.
- [4] B. Schneier, *Applied Cryptography Second Edition: Protocols, Algorithms and Source Code in C*, 2nd ed., John Wiley & Sons, 1996.
- [5] S. Wolfram, Cryptography with Cellular Automata, *Advances in Cryptology: Crypto '85 Proceedings, Lecture Notes in Computer Science*, 218: 429-432, 1986.
- [6] S. Nandi, B. K. Kar and P. Pal Chaudhuri, Theory and applications of cellular automata in cryptography, *IEEE Trans. Computers*, 43:1346-1357, 1994.
- [7] P. Sarkar, A brief history of cellular automata, *ACM Computing Surveys*, 32:80-107, 2000.
- [8] M. Mitchell, Computation in Cellular Automata: A Selected Review, *Santa Fe Intitute*, 1996.

- [9] N. H. Packard and S. Wolfram, Two-dimensional cellular automata. *Journal of Statistical Physics*, 38:901–946, 1985.
- [10] J. von Neumann, *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, 1966.
- [11] S. Wolfram, *A New Kind of Science*. Wolfram Media, Champaign, IL, 2002.
- [12] S. Wolfram, Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:601–644, 1983.
- [13] V. C. Barbosa, F. M. N. Miranda and M. C. M. Agostini, Cell-centric heuristics for the classification of cellular automata, *Parallel Computing*, 32: 44-66, 2006.
- [14] R. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communications of the ACM*, 21:120-126, 1978.
- [15] N. Ferguson, B. Schneier, *Practical Cryptography*, Wiley, 1<sup>st</sup>. ed., 2003.
- [16] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.
- [17] L. A. V. de Carvalho, *Síntese de Redes Neurais com Aplicações a Representação do Conhecimento e à Otimização*. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1989.
- [18] C. P. Robert, G. Casella, *Monte Carlo Statistical Methods*, Springer, 2nd. ed., 2005.
- [19] M. W. Trosset, What is Simulated Annealing? *Optimization and Engineering*, 2:201-213, 2001.

- [20] C. E. Shannon, "A Mathematical Theory of Communication", *Bell Syst. Tech. J.*, 27: 379-423, 623-656, 1948.
- [21] S. Russell and P. Norvig., *Artificial Intelligence: A Modern Approach*, 2nd. ed., Prentice Hall, 2003.
- [22] M. Tomassini and M. Perrenou, Cryptography with cellular automata. *Applied Soft Computing*, 1:151–160, 2001.
- [23] M. Tomassini, M. Sipper and M. Perrenoud, On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata, *IEEE Trans. Computers*, 49:1146:1151, 2000.
- [24] M. Tomassini, M. Sipper, M. Zolla and M. Perrenoud, Generating high-quality random numbers in parallel by cellular automata, *Future Generation Computer Systems*, 16:291-305, 1999.
- [25] D. E. Eastlake, S. D. Crocker, and J. I. Schiller, RFC 1750: Randomness Recommendations for Security, *Network Working Group*, 1994.
- [26] W. Bem-Ameur, Computing the Initial Temperature of Simulated Annealing, *Computational Optimization and Applications*, 29: 369-385, 2004.
- [27] A. Rukhin, J. Soto, J. Nechvatal and others, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication 800-22, May 15, 2001. <http://csrc.nist.gov/rng/>, weblink.

- [28] V. J. B. de Paula, *Impactos e Compromissos do uso da Cifra de Vernam Envolvendo o Desempenho de Aplicações em Rede*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2005.
- [29] D. R. Chowdhury, I. S. Gupta and P. P. Chaudhuri, A class of two-dimensional cellular automata and applications in random pattern testing. *J. Electron. Test.: Theory Appl.*, 5: 65-80, 1994.