UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Marco André Lopes Mendes

Modelo Simplificado do Cifrador RC6

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Ricardo Felipe Custódio, Dr.
Orientador
custodio@inf.ufsc.br

Florianópolis, Dezembro de 2001

Modelo Simplificado do Cifrador RC6

Marco André Lopes Mendes

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Ricardo Felipe Custódio, Dr.

Orientador

custodio@inf.ufsc.br

Prof. Fernando Álvaro Ostuni Gauthier, Dr.

Coordenador do Curso

gauthier@inf.ufsc.br

Banca Examinadora

Prof. Cláudio César de Sá, Dr.

dcc2ccs@udesc.br

Prof. João da Silva Dias, Dr.

jdias@ppgia.pucpr.br

Prof. Marcelo da Silva Hounsel, Dr.

dcc2msh@udesc.br

Prof. Vitório Bruno Mazzola, Dr.

mazzola@inf.ufsc.br

"O temor ao Senhor é o princípio da sabedoria"
Provérbios 9:10

Para Alessandra e Gabriel, meus maiores tesouros.

Agradecimentos

A Deus por ter me criado com um propósito. À minha esposa Alessandra e meu filho Gabriel pelo tempo que era deles e me foi permitido dispensar neste trabalho. Aos meus pais, pelo incentivo em todos os momentos. À SOCIESC, por ter me dado as condições necessárias para realizar este empreendimento. Ao meu orientador, pelo privilégio de trabalharmos juntos. A Eduardo da Silva pela ajuda com a edição em LaTeX e na programação em C. A Gilsiley Darú pelo grande apoio no desenvolvimento do modelo. Aos professores do IST, que ajudaram na revisão deste trabalho e deram valiosas sugestões.

Abstract

The development of a simplified model for the RC6 symmetrical block cipher, denominated S-RC6 is proposed. A simplified model can help one to understand a cipher, for it keeps the primitive operations, while decreseases the parameters, making it easier to use it. Some RC6's simplified variants are analised to verify if any fullfil the requiriments defined in this document. The principles that surround the symmetrical block cipher's design are studied, and some of the most important symmetrical block ciphers are compared. DES and its simplified model, S-DES, are analised, to verify if S-DES contributes to the understanding of DES. A manual decryption using S-DES is performed to ilustrate its use. The RC2, RC4, RC5 and RC6 algorithms are studied to define their commom features and their evolution through the years.

Resumo

Este trabalho consiste na proposta de um modelo simplificado para o cifrador de bloco simétrico RC6, a ser denominado S-RC6. Um modelo simplificado pode auxiliar no entendimento de um cifrador, pois ele mantém as operações primitvas, enquanto diminui os parâmetros, facilitando seu uso. Algumas variantes simplificadas do RC6 são analisadas para verificar se alguma atende os objetivos definidos neste trabalho. Os princípios que envolvem o projeto de cifradores de bloco simétricos são estudados, e são comparados vários dos mais importantes cifradores de bloco simétricos existentes. O DES e seu modelo simplificado, o S-DES são analisados, verificando-se a contribuição que o S-DES traz no entendimento do DES. É feita uma decifração manual utilizando o S-DES para demonstrar seu uso. Os algoritmos RC2, RC4, RC5 e RC6 são estudados, visando definir suas características em comum e sua evolução através do tempo.

Conteúdo

Lista de Figuras

Lista de Tabelas

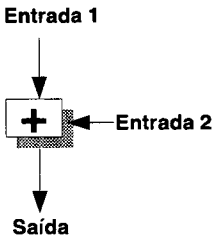
Lista de siglas e abreviaturas

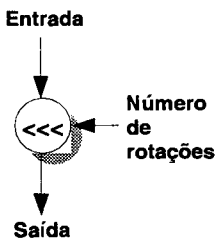
3DES	DES Triplo
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
CBC	Cipher Block Chaining
CTS	Ciphertext Stealing Mode
CFB	Cipher Feedback
DES	Data Encryption Standard
DES-EEE3	DES Triplo que utiliza três cifrações com três chaves diferentes
DES-EDE3	DES Triplo que utiliza três operações DES na seqüência cifra-decifra-cifra utilizando três chaves diferentes
DES-EEE2	Idêntico ao DES-EEE3, exceto que a primeira e terceira operações utilizam a mesma chave
DES-EDE2	Idêntico ao DES-EDE3, exceto que a primeira e terceira operações utilizam a mesma chave
ECB	Electronic Codebook
E/P	Expansão/Permutação
FIPS	Federal Information Processing Standard
F	Função F, reponsável pela confusão de *bits*
f_k	Função de confusão de *bits* dependente de k
FR	Fixed Rotation
IBM	International Bussiness Machines Corporation
IDEA	International Data Encryption Algorithm
IEC	International Electrotechnical Commission

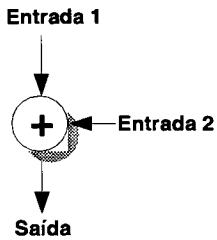
IEEE	Institute of Electrical and Electronic Engineers
IP	Permutação Inicial
$IP^{-}1$	Permutação Inversa da Inicial
ISO	International Organization For Stardadization
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector
IV	Inicialization Vector
K_N	N-ésima chave ou sub-chave
L_N	N-ésima metade esquerda de um bloco
LS-1	Deslocamento circular à esquerda (rotação) de 1 *bit*
LS-2	Deslocamento circular à esquerda (rotação) de 2 *bits*
LUCIFER	Cifrador de Bloco Simétrico desenvolvido pela IBM e que foi o precursor do DES
MARS	Cifrador de Bloco Simétrico desenvolvido pela IBM
MIT	Massachussets Institute of Technology
NESSIE	New European Schemes for Signature, Integrity, and Encryption
NFR	No Fixed Rotation
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OFB	Output Feedback
P10	Permutação de 10 *bits*
P4	Permutação de 4 *bits*
P8	Permutação de 8 *bits*
PGP	Pretty Good Privacy
P_w	Constante P com *w bits* de comprimento
Q_w	Constante Q com *w bits* de comprimento
RAM	Random Access Memory - Memória Principal do Computador
RC2	Ron's Code 2 - Cifrador de bloco simétrico desenvolvido por Ron Rivest
RC4	Ron's Code 4 - Cifrador de fluxo desenvolvido por Ron Rivest

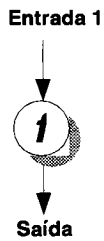
RC5	Ron's Code 5 - Cifrador de bloco simétrico desenvolvido por Ron Rivest
RC6	Ron's Code 6 - Cifrador de bloco simétrico desenvolvido por Ron Rivest
R_N	N-ésima metade direita de um bloco
RSA	Rivest-Shamir-Adelman - Algoritmo de Criptografia por chave pública
Rijndael	Cifrador de Bloco Simétrico desenvolvido por Joan Daemen e Vincent Rijmen
S_0	Primeira caixa S
S_1	Segunda caixa S
Serpent	Cifrador de Bloco Simétrico desenvolvido por Ross Anderson and Eli Biham and Lars Knudsen
S-DES	Simplified Data Encryption Algorithm
S-MIME	Secure MIME - Protocolo para correio eletrônico seguro
S-RC6	Simplified RC6 - RC6 Simplificado
SK	Sub-chave
SW	Swap - Troca entre duas metades de um bloco
SQL	Structured Query Language
SSL	Security Socket Layer - Protocolo para web segura
Troca	Troca entre as duas metades de um bloco
Twofish	Cifrador de Bloco Simétrico desenvolvido por Bruce Schneier e equipe

Lista de Símbolos









Capítulo 1

Introdução

A segurança da informação está se tornando mais importante a medida que mais dados são enviada através das redes. Uma vez que os dados são colocados numa rede, eles podem ser interceptados em vários pontos. Uma forma de proteger a informação, sendo ela transmitida ou permanecendo num computador, é cifrá-la, isto é, torná-la ilegível a todos, exceto àqueles que realmente podem ter acesso a ela. Para o adequado uso e o aperfeiçoamento dos algoritmos de criptografia, faz-se necessário o estudo e a análise dos mesmos, como também o seu perfeito entendimento, para a procura de possíveis vulnerabilidades.

Um dos algoritmos mais utilizados na atualidade para a cifração de informações é o DES (Data Encription Standard - Padrão de Cifração de Dados) [STA 99]. O DES surgiu da necessidade de se ter um único algoritmo que pudesse ser utilizado em todas as trocas de informações do governo e das empresas privadas americanas. Começando como um padrão do governo, o DES acabou tornando-se um padrão de mercado, principalmente porque o governo americano é um grande comprador de tecnologia e isto força seus parceiros a se adequarem a seus padrões. O fato é que atualmente o DES é implementado em circuitos integrados e em software, assim como é utilizado em transações comerciais, comunicação telefônica, troca de mensagens eletrônicas e diversos outros serviços que necessitam de segurança na comunicação entre seus parceiros.

Em função da sua importância, o DES é descrito e estudado na qua-

se totalidade dos materiais de referência sobre Criptografia, Segurança em Redes, Redes de Computadores, Comunicação de Dados e áreas relacionadas, como em [MAS 88, MEN 96, LAB 88, SCH 96b, STA 99, STI 95, TAN 96]. Para facilitar seu entendimento, foi desenvolvido o DES Simplificado [SCH 96a], conhecido como S-DES, que consiste num algoritmo com características semelhantes ao DES, porém com parâmetros reduzidos. Por exemplo, enquanto o DES executa 16 rodadas sobre blocos de 64 bits, o S-DES executa apenas 2 rodadas sobre blocos de 8 bits. O objetivo é que o S-DES seja facilmente estudado, analisado e até mesmo implementado em software ou utilizado manualmente. Após seu estudo, se pode, então, estudar as diferenças entre este e o DES, ajudando no entendimento deste último.

No início da década de 90, começou a surgir uma série de trabalhos de criptoanálise sobre o DES. Estes trabalhos mostraram algumas vulnerabilidades que poderiam, num curto espaço de tempo, tornar o DES inseguro.

No final da década de 90, nos Estados Unidos, o NIST[1] iniciou os trabalhos com o objetivo de desenvolver um novo padrão de criptografia. Este novo padrão, substituto do DES, seria conhecido como AES[2] [FOR 00]. O objetivo era desenvolver um Padrão de Processamento de Informações Federais (FIPS) que especificasse um algoritmo capaz de proteger as informações importantes do governo durante as próximas décadas. Assim como com o DES, se espera que ele seja utilizado pelo Governo Americano e, voluntariamente, no setor privado bem como em outros países [FOR 00].

1.1 O contexto de RC6

Em 1997, o NIST anunciou o esforço para o desenvolvimento do AES e fez uma chamada formal por algoritmos para concorrerem a se tornar este padrão. Então, foi anunciado um grupo de quinze algoritmos que se enquadravam nas características básicas requisitadas pelo NIST. Após uma série de análises e comentários feitos pelo próprio NIST e pela comunidade, cinco finalistas foram escolhidos dentre os quinze can-

[1] National Institute of Standards and Technology - Instituto Nacional de Padrões e Tecnologia

[2] Advanced Encryption Standard - Padrão de Criptografia Avançada

didatos iniciais. Os cinco algoritmos candidatos finalistas foram MARS, RC6, RIJNDAEL, SERPENT e TWOFISH. Estes finalistas receberam análises adicionais durante um segundo e mais longo período, anterior à escolha final do algoritmo AES. Durante este período, o NIST solicitou e analisou comentários e análises públicas bem como patrocinou conferências para a discussão dos algoritmos finalistas. Finalmente, em outubro de 2000, o NIST anunciou que havia escolhido o algoritmo RIJNDAEL como proposta para o AES [FOR 00].

A escolha do algoritmo RIJNDAEL para ser o AES não significou que foram encontrados problemas de segurança nos demais algoritmos. Em relação à segurança dos cinco finalistas, o NIST afirmou em seu relatório que *"todos os cinco algoritmos parecem ter a segurança adequada para o AES"* [FOR 00].

O RC6 foi considerado o mais rápido dos finalistas em plataformas de 32 *bits*. Sua maior desvantagem está na necessidade de se computar todas as sub-chaves antes do início do processo de decifração, criando necessidade de uma grande quantidade de memória. Desta forma, ele não é muito indicado para a implementação em dispositivos com restrição de memória principal disponível, quando a decifração é necessária, como é o caso de *"smartcards"*. O relatório final sobre os candidatos ao AES está disponível em [NEC 00].

Enquanto o projeto AES se aproxima de sua conclusão, outros esforços para definir famílias de algoritmos seguros de criptografia estão emergindo. O RC6 está atualmente sendo submetido a três destes projetos [LAB 01]:

- **Projeto NESSIE:** Este é o projeto para o novo esquema europeu para assinatura, integridade e cifração, algo como o AES Europeu, só que mais amplo. O projeto NESSIE envolve cifradores de bloco simétricos, cifradores de fluxo assimétricos e esquemas de assinatura. Este projeto terá um grande impacto no futuro das aplicações de criptografia na Europa;

- **Projeto ISO/IEC NP 18033:** Este projeto da ISO tem como objetivo encontrar algoritmos simétricos e assimétricos de cifração. Este algoritmos serão incluídos em um padrão da ISO/IEC;

- **Projeto CRYPTREC:** Este projeto japonês possui objetivos similares ao projeto AES. O alvo é definir algoritmos de cifração padrão para o uso dentro do governo japonês. Da mesma forma que o projeto NESSIE, o escopo é mais amplo do que o projeto AES.

Percebe-se por estes esforços no sentido de definir novos padrões que o mercado continuará analisando e adotando vários outros algoritmos de cifração de dados, além do AES definido pelo NIST, dentre estes, o RC6.

O RC6 foi desenvolvido por Ron Rivest, professor do MIT e co-fundador da empresa RSA Data Security, e pelos Laboratórios da RSA. Ele é também o responsável pelo desenvolvimento dos algoritmos RSA, RC2, RC4 e RC5, amplamente utilizados em vários protocolos e produtos de segurança da Internet, como PGP, SSL, S/MIME e outros. A empresa RSA Data Security, empresa fundada pelos inventores do sistema de criptografia por chave-pública RSA, é uma das maiores empresas de segurança do mundo, fornecendo sistemas de segurança eletrônica de dados. As tecnologias da RSA fazem parte de padrões existentes ou propostos para a Internet, ISO, ITU-T, ANSI, IEEE e redes de negócios, financeiras e de comércio eletrônico ao redor do mundo. Ela também desenvolve e comercializa componentes de segurança independentes de plataforma e kits de desenvolvimento relacionados e fornece serviços de consultoria na área de criptografia. Os Laboratórios da RSA são a divisão de pesquisa da empresa RSA Data Security. Seu propósito é o de fornecer conhecimento de alto nível em criptografia e segurança de informação para o benefício da RSA Data Security e seus clientes [RSA 00].

O RC6 é uma variação aperfeiçoada do algoritmo cifrador de bloco RC5, que recebeu atenção considerável nos últimos anos por criptoanalistas em todo o mundo, e tem se mostrado virtualmente imune a todos os tipos de ataques práticos, incluindo os mais avançados ataques de criptoanálise diferencial e linear, bem como os ataques de procura de chaves por força bruta [RSA 00].

1.2 Objetivo do trabalho

Para facilitar o estudo das características do cifrador de bloco RC6, faz-se necessário, da mesma forma que aconteceu com o DES, o desenvolvimento de um modelo simplificado. Este trabalho tem como objetivo propor um modelo simplificado para o cifrador de bloco simétrico RC6, da RSA Data Security. Um modelo simplificado serve para o estudo didático do funcionamento de um cifrador, para a análise do cifrador e de suas características de funcionamento e para a descoberta de falhas e o desenvolvimento de melhorias no mesmo. Além disto, pretende-se que o resultado deste trabalho torne-se uma referência para o desenvolvimento de modelos simplificados de cifradores.

É recomendado que o leitor deste documento possua um conhecimento básico de Criptografia, incluindo o entendimento do que vem a ser: efeito avalanche, critério de independência de bits, caixas S, difusão, confusão, criptoanálise diferencial e linear, ataques por força bruta, ataque *meet-in-the-middle*. A leitura de [STA 99], sobretudo os quatro primeiros capítulos, é recomendada para o melhor entendimento do trabalho.

1.3 Materiais e métodos

Foram utilizados ambientes operacionais e compiladores distintos para as implementações dos cifradores. As implementações do S-DES, que aparece no anexo B e do S-RC6, que aparece no anexo F foram feitas no compilador `gcc`[3] sendo executado em computadores com o sistema operacional Linux. Foram utilizadas as distribuições Conectiva Linux 4.2, 5.0, 6.0 e 7.0[4] e SuSE Linux 7.2[5]. A implementação do SRC-6 em C-Builder[6] , que aparece no anexo G foi feita com a versão 3 do C-Builder sendo executada em computadores executando os sistemas operacionais Windows NT 4.0 e Windows 98[7].

[3] `http://www.gnu.org/`
[4] `http://www.conectiva.com/`
[5] `http://www.suse.com/`
[6] `http://www.inprise.com/`
[7] `http://www.microsoft.com/`

Com exceção da implementação do S-RC6 feita em C-Builder, todas as demais atividades necessárias ao desenvolvimento deste trabalho foram feitas utilizando o sistema operacional Linux e as ferramentas de código aberto disponíveis para esta plataforma. Tanto o trabalho individual quanto esta dissertação foram escritos utilizando LaTeX no editor de textos `vim`[8]. A correção ortográfica foi feita com o `ispell`[9], utilizando o dicionário em português desenvolvido pela equipe da USP[10]. A visualização e impressão dos documentos `Postscript` gerados pelo LaTeX foram feitas com a ferramenta `gv`[11]. A conversão dos documentos LaTeX para o formato `PDF` foi feita através das ferramentas `ps2pdf` (incluída no pacote `ghostscript`[12]), `dvips` e `pdflatex` (incluídos no pacote `tetex`[13]). A apresentação do trabalho individual foi feita utilizando a ferramenta `prestimel`[14]. A apresentação da dissertação foi feita em LaTeX utilizando o pacote `FoilTeX`[15] e fontes True Type importadas do Windows. As ilustrações foram feitas no `Star Draw`, o editor de desenhos do `Star Office`[16] e depois exportadas para os formatos `EPS` ou `JPG`, conforme a necessidade. A manipulação das figuras e conversão de formatos foi feita utilizando o `Gimp`[17]. Todas as pesquisas na Internet foram feitas utilizando-se a ferramenta de busca `Google`[18].

Foram ainda utilizados alguns *scripts* para a manutenção dos arquivos contendo os capítulos em formato LaTeX, como a pesquisa e substituição de termos em vários arquivos simultaneamente. Utilitários padrão do ambiente Linux, como *grep*, *zip*, *diff*, *make*[19] e outros foram também amplamente utilizados.

---

[8] `http://www.vim.org/`
[9] `http://fmg-www.cs.ucla.edu/ficus-members/geoff/ispell.html`
[10] `http://www.ime.usp.br/~ueda/br.ispell`
[11] `http://wwwthep.physik.uni-mainz.de/~plass/gv/`
[12] `http://www.cs.wisc.edu/~ghost/`
[13] `http://www.tug.org/teTeX/`
[14] `http://oeh.tu-graz.ac.at/prestimel/`
[15] `http://www.ensta.fr/internet/tex/style_macros/foiltex.html`
[16] `http://www.sun.com/staroffice/`
[17] `http://www.gimp.org/`
[18] `http://www.google.com/`
[19] `http://www.gnu.org/`

1.4 Organização do trabalho

Este trabalho está organizado da seguinte forma: o capítulo 2 descreve os fundamentos da criptografia e analisa as características importantes a serem levadas em consideração no projeto de um bom cifrador de bloco simétrico. Este capítulo compara também vários cifradores baseando-se nas características analisadas. O capítulo 3 analisa as características e o funcionamento do DES e do DES Simplificado, relacionando suas semelhanças e mostrando a importância do modelo simplificado no estudo de um cifrador. No capítulo 4, são analisados os cifradores de Ron Rivest. São analisadas as características de cada um deles, verificando as semelhanças e a evolução ocorrida entre eles no decorrer dos anos. O capítulo 5 analisa as características e o funcionamento de algumas variantes simplificadas do RC6, tentando verificar o uso destas variantes para o melhor entendimento do próprio RC6. No capítulo 6, é proposto o desenvolvimento de um modelo simplificado do cifrador RC6 e as estratégias adotadas para alcançar este objetivo são explicadas. Os anexos A e B consistem em implementações dos cifradores DES e S-DES respectivamente, em linguagem C. O anexo C reproduz a mensagem original postada anonimamente numa lista de mensagens, e que divulgou o código do RC4 ao mundo. O anexo D descreve os passos que foram utilizados no projeto do RC6, tomando o RC5 como ponto de partida. O anexo E consiste na implementação do cifrador RC6 em linguagem C. Os anexos F e G consistem em implementações do cifrador S-RC6 em linguagem C e C-Builder, respectivamente.

Capítulo 2

Projeto de Cifradores de Bloco Simétricos

2.1 Introdução

O objetivo deste capítulo é definir os parâmetros que devem ser levados em consideração no desenvolvimento de um cifrador. Estes parâmetros serão necessários para o desenvolvimento da proposta do modelo simplificado do cifrador RC6, que é o objetivo deste trabalho. As principais características a serem consideradas no projeto de um bom cifrador de bloco simétrico são analisadas e também são comparados alguns dos mais importantes cifradores atualmente em uso utilizando estas características.

2.2 Cifradores

Cifradores são algoritmos que transformam um texto conhecido, chamado de *texto aberto*, em um texto ilegível, conhecido como *texto cifrado*. Os *cifradores de fluxo* (*"stream ciphers"*) são aqueles que cifram um *bit* ou *byte* de cada vez, enquanto que os *cifradores de bloco* (*"block ciphers"*) tratam um bloco de texto aberto como um todo e produzem um bloco de texto cifrado de igual comprimento [STA 99]. Os *cifradores simétricos*, também conhecidos como cifradores de cifração convencional, utilizam

a mesma chave para o processo de cifração e decifração, enquanto que os *cifradores assimétricos* ou cifradores de cifração por chave pública utilizam uma chave para cifração e outra para decifração. A figura 2.1 ilustra o processo de funcionamento de um cifrador simétrico. Neste caso, a chave secreta K deve ser compartilhada pelo emissor (aqui denominado Alice) e pelo receptor (denominado Bob) da mensagem, já que ela será utilizada tanto no processo de cifração quanto na decifração da mensagem.

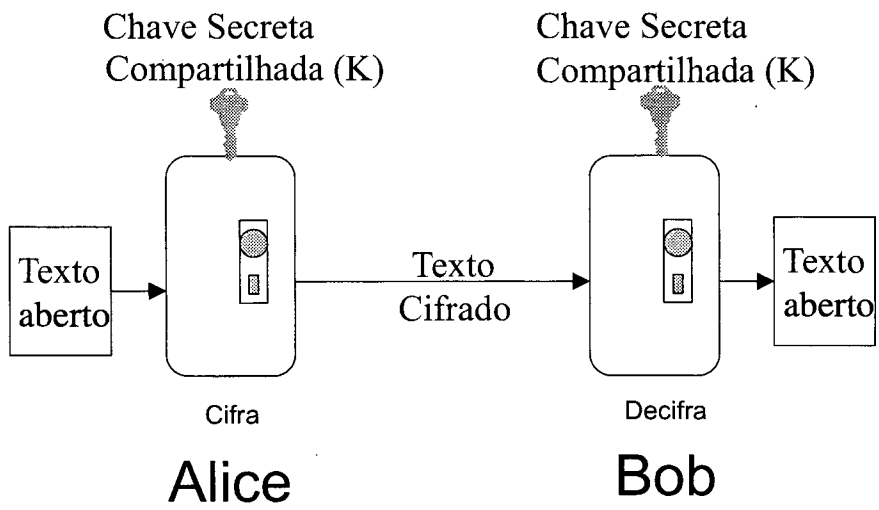


Figura 2.1: Funcionamento de um Cifrador Simétrico

Este trabalho se concentrará no estudo de *Cifradores de Bloco Simétricos*, que são algoritmos que transformam textos abertos em textos cifrados tratando a informação como blocos de *bits* e utilizando a mesma chave secreta tanto para a cifração quanto para a decifração [STA 99]. Os cifradores de bloco simétricos são utilizados devido a velocidade com que eles cifram os dados, sendo o desempenho numa transmissão muito pequeno. Outra vantagem deles é que as chaves podem ser muito menores, quando comparadas àquelas utilizadas nos cifradores de criptografia por chave pública.

2.3 Principais passos no projeto de cifradores simétricos

Embora muito avanços tenham surgido no sentido de se projetar cifradores mais fortes, os princípios fundamentais não mudaram tanto desde o trabalho de

Feistel [FEI 73] e da equipe de desenvolvimento do DES, no início dos anos 70. Baseados nisto, os três principais aspectos que devem ser levados em consideração no projeto de cifradores de bloco são: o *número de rodadas*, o *projeto da função F* e o *agendamento de chaves* [STA 99].

Primeiro será analisado o projeto da função F. O "coração" de um cifrador de blocos de Feistel e da maioria dos cifradores de bloco simétricos é a função F. Ela fornece o elemento de confusão, tornando difícil desembaralhar a substituição feita pela função F. A função F deve ser não linear, possuir um bom efeito avalanche e um bom critério de independência de *bits*. A não-linearidade é uma característica muito importante em um cifrador, pois dificulta a criptoanálise do mesmo. As caixas S são uma forma de alcançar estes objetivos. A figura 2.2 ilustra uma Estrutura Clássica de Feistel. A entrada do algoritmo de cifração é um bloco de texto aberto de 2 *bits* e uma chave K. O texto aberto é dividido em duas metades, L_0 e R_0. As duas metades do dado passam por n rodadas de processamento para produzir o bloco de texto cifrado. Cada rodada i tem como entradas $L_i - 1$ e $R_i - 1$, derivadas da rodada precedente, bem como uma chave K_i, derivada da chave principal K. De forma geral, as sub-chaves K_i são diferentes de K e também diferentes umas das outras. Todas as rodadas possuem a mesma estrutura, e as operações são sempre executadas sobre a metade esquerda do bloco. É importante notar que numa Estrutura Clássica de Feistel a função F opera apenas sobre a metade do bloco em cada rodada, sendo necessárias pelo menos duas rodadas para a operação sobre todos os *bits* do bloco. O DES utiliza as Estruturas de Feistel desta forma, enquanto outros cifradores, como Blowfish e RC6 operam em ambas as partes do bloco a cada rodada [STA 99, RIV 98b].

Quanto maior o número de rodadas, mais forte será um algoritmo contra ataques, mesmo para uma função F fraca. O número de rodadas escolhido deve ser tal que a melhor forma de se descobrir uma senha seja pelo ataque de força bruta, ou seja, pelo teste de todas as possíveis chaves, até que seja encontrado o texto aberto original. Supondo uma chave de *n bits* de comprimento, existem 2^n possíveis chaves, e um ataque por força bruta deve testar, em média $2^{(n-1)}$ chaves até encontrar a chave correta [STA 99].

Por último, pode-se analisar o algoritmo de agendamento das chaves,

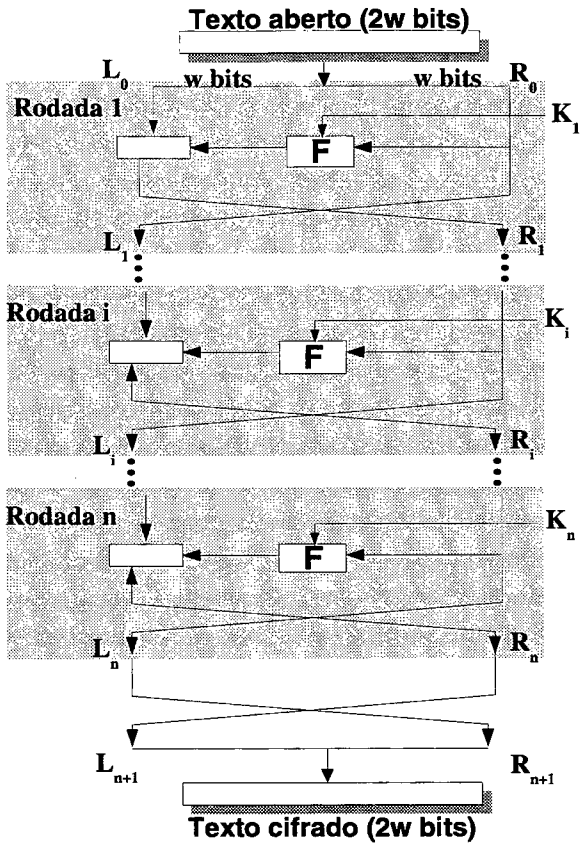


Figura 2.2: Estrutura Clássica de Feistel

que é a geração de sub-chaves K_1, K_2, ..., K_i para as *i* rodadas, a partir da chave inicial. O objetivo aqui é maximizar a dificuldade de deduzir sub-chaves individuais, bem como de dificultar a descoberta da chave principal. Espera-se que o agendamento de chaves possua um bom efeito avalanche e um bom critério de independência de *bits* [STA 99].

Além destes aspectos analisados, outras características que são encontradas no projeto dos mais importantes algoritmos simétricos atualmente em uso podem ser citadas[STA 99]. Estas são:

1. Comprimento da chave;
2. Comprimento do bloco;
3. Número de rodadas;
4. Comprimento variável da chave;

5. Uso de mais do que um operador aritmético ou Booleano;
6. Rotações de *bits* dependentes dos dados;
7. Rotações de *bits* dependentes da chave;
8. Caixas S dependentes da chave;
9. Uso de algoritmos longos de expansão de chave;
10. Função F variável para cada rodada;
11. Comprimento variável do bloco de texto aberto e texto cifrado;
12. Número de rodadas variável;
13. Operação no bloco inteiro em cada rodada de uma Estrutura de Feistel.

Estas características servem de referência para a tabela 2.1, que compara vários cifradores simétricos atualmente em uso. Os cifradores são apresentados por ordem de criação. Pode-se perceber assim a evolução dos cifradores ao longo do tempo e notar a incorporação de novas características ao cifradores mais recentes. Uma característica marcante é o uso de atributos variáveis, sobretudo o comprimento da chave. Isto favorece a escolha por parte do usuário do cifrador da melhor relação entre segurança e desempenho. Evita-se com isto também o erro cometido com o DES, que é considerado um bom cifrador, porém com uma chave de comprimento muito pequeno. Os valores de comprimento de chave e de blocos são medidos em *bits*. "S" indica que a característica existe no algoritmo, "N" que não existe e "NA" que a característica não se aplica ao algoritmo. Devido à grande diferença na estrutura destes algoritmos, muitas das características aqui descritas se aplicam apenas parcialmente ou exigiriam explicações com relação ao seu funcionamento. Para maiores detalhes sobre os algoritmos aqui referenciados, podem ser consultadas as seguintes referências: [STA 99] para detalhes sobre o DES, IDEA, Blowfish e RC5; [RIV 94] e [KAL 95] para detalhes sobre o RC5; [RIV 98b] e [CON 98] para saber mais sobre o RC6; [BUR 99] para o MARS; [AND 98] para o Serpent; [DAE 98] para o Rijndael e [SCH 98] para o Twofish.

2.4 Conclusão

Foram definidos os tipos de cifradores existentes e citadas as características que definem um bom projeto de cifrador simétrico. Algumas destas características são decorrentes do desenvolvimento do DES e da Estrutura de Feistel, enquanto outras têm surgido no desenvolvimento de cifradores mais modernos, como o AES. Consenso entre os pesquisadores é que uma boa estratégia seja encontrar refinamentos no cifrador de Feistel e no DES, ao invés de propor uma estrutura totalmente nova [STA 99]. O motivo é que a estrutura de Feistel tem sido amplamente estudada e não foram encontradas até hoje falhas na sua estrutura. Existem, no entanto, diversos cifradores desenvolvidos recentemente que não utilizam a estrutura de Feistel, sendo o mais notável de todos o Rijndael [DAE 98], que venceu a corrida para o AES. Na tabela 2.1, que compara os diversos cifradores, tem-se uma visão das estratégias utilizadas no desenvolvimento de um cifrador e também é possível verificar a evolução no desenvolvimento destes ao longo do tempo.

Tabela 2.1: Comparação entre os principais cifradores simétricos

Algoritmo	Ano da criação	Comprimento da chave	Comprimento dos blocos de entrada e saída	Número de rodadas	Comprimento variável da chave	Uso de mais do que um operador	Rotação dependente dos dados	Rotação dependente da chave	Caixas S dependentes da chave	Uso de algoritmos longos de expansão da chave	Função F variável para cada rodada	Comprimento variável do bloco	Número de rodadas variável	Operação no bloco inteiro em cada rodada
DES	1974	56	64	16	N	N	N	N	N	N	N	N	N	N
IDEA	1990	128	64	8	N	S	N	N	NA	N	N	N	N	NA
Blowfish	1993	32-2048	64	16	S	S	N	N	S	S	N	N	N	S
RC5	1994	0..2048	32,64,128	0..255	S	S	S	S	NA	S	N	S	S	S
SDES	1996	10	8	2	N	N	N	N	N	N	N	N	N	N
RC6	1998	128,192,256	64,128,256	0..255	S	S	S	S	NA	S	N	S	S	S
Rijndael	1998	128,192,256	128,192,256	¿10	S	S	S	N	S	S	N	S	S	NA
Serpent	1998	128,192,256	128	32	S	N	N	N	N	N	N	N	N	N
Twofish	1998	0..256	128	16	S	N	N	N	S	S	N	N	N	N
MARS	1998	128..448	128	32	S	S	N	N	N	S	N	N	N	S

S: Sim

N: Não

NA: Não aplicável

Capítulo 3

Os cifradores DES e S-DES

3.1 Introdução

Uma das maiores dificuldades que se encontra está na apresentação e conseqüente entendimento de um cifrador. Alguns cifradores são muito complexos, sendo difícil o entendimento do seu funcionamento, implicando também numa dificuldade de se analisá-los, seja para encontrar vulnerabilidades, para propor uma implementação ou para propor melhorias.

Após um extenso levantamento bibliográfico, verificou-se que existem poucas propostas de modelos simplificados disponíveis para estudo. Um dos mais conhecidos é o S-DES, ou DES Simplificado, cujo objetivo é facilitar o entendimento do conhecido cifrador DES. Desta forma, este capítulo apresenta o DES e também o seu modelo simplificado, como forma de demonstrar que o estudo de um modelo simplificado facilita o entendimento do algoritmo real, mesmo no caso de um algoritmo complexo, como se mostra o DES.

3.2 DES Simplificado

Uma dificuldade encontrada no estudo do DES é o seu nível de complexidade e a dificuldade de seu entendimento [SCH 96a]. O DES Simplificado, que

será chamado de S-DES, é um algoritmo muito mais didático do que de cifração segura, tendo propriedades e estrutura similares ao DES, porém com parâmetros muito menores [SCH 96a]. Uma vez tendo entendido o S-DES, se torna muito mais fácil expandir os parâmetros para ver como o DES real funciona [SCH 96a].

Esta seção apresenta as características do S-DES, o seu algoritmo de funcionamento incluindo o processo de geração de sub-chaves. Todas as etapas da cifração são explicadas em detalhes. Depois, a segurança do algoritmo é discutida. No final, é demonstrado o uso do S-DES para a cifração de um texto, e é feita uma comparação entre o DES e o S-DES.

3.2.1 Características do S-DES

A figura 3.1 ilustra o esquema do DES Simplificado. No anexo B se encontra uma implementação do S-DES em linguagem C. O algoritmo de cifração S-DES recebe um texto aberto de 8 *bits* e uma chave de 10 *bits* como entrada e produz um bloco de texto cifrado de 8 *bits* como saída. O algoritmo é simétrico e portanto a mesma chave é usada na decifração [STA 99].

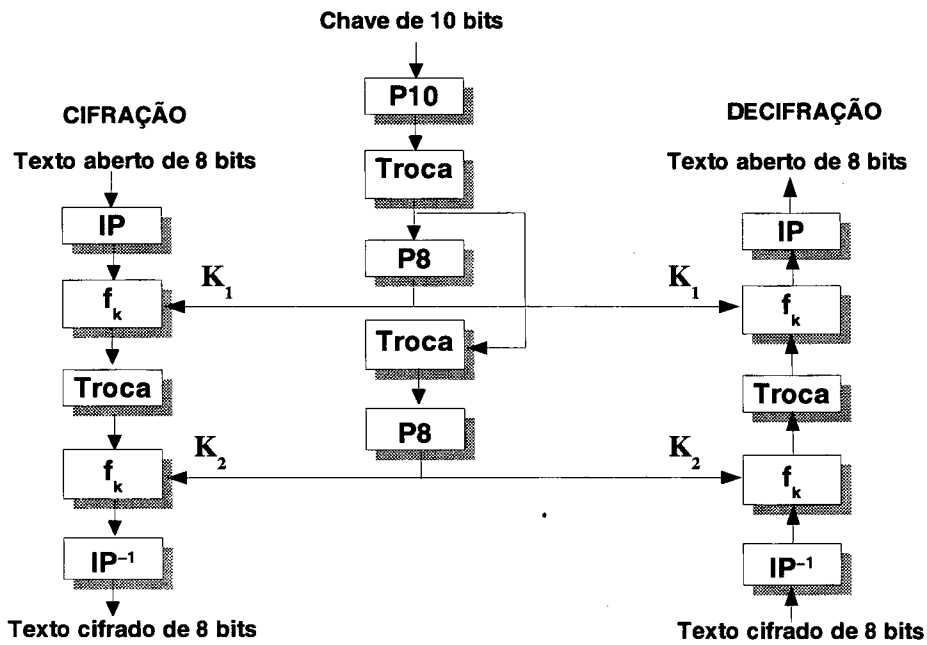


Figura 3.1: Esquema de funcionamento do S-DES

3.2.2 Funcionamento do algoritmo S-DES

O algoritmo de cifração envolve cinco funções [STA 99]:

1. Uma permutação inicial (IP)[1];
2. Uma função complexa f_k, que envolve operações de permutação e substituição e é dependente da chave de entrada;
3. Uma operação de permutação simples que troca as duas metades do dado;
4. A função f_k novamente;
5. Finalmente, uma função de permutação, que é o inverso da permutação inicial (IP^{-1}).

3.2.3 Geração das sub-chaves do S-DES

O S-DES depende do uso de uma chave de 10 *bits* compartilhada entre o emissor e o receptor. A partir desta chave, duas sub-chaves de 8 *bits* são produzidas para o uso no algoritmo de cifração e decifração [STA 99]. A figura 3.2 ilustra os estágios envolvidos na geração das sub-chaves. Primeiro os *bits* da chave são permutados com a permutação P10[2], ilustrada na tabela 3.1.

Tabela 3.1: Permutação P10

3	5	2	7	4	10	1	9	8	6

Depois, se executa um deslocamento circular à esquerda, ou rotação de 1 *bit*, separadamente nos primeiros 5 *bits* e últimos 5 *bits* (LS-1)[3].

Depois se aplica P8, que pega e permuta 8 dos 10 *bits*, conforme ilustra a tabela 3.2. Os dois primeiros *bits* não são utilizados.

[1] "IP" vem do termo em inglês *Initial-Permutation*, ou Permutação Inicial.

[2] "P" vem do termo em inglês *Permutation*, ou Permutação.

[3] "LS" vem do termo em inglês *Left-Shift*, ou deslocamento circular à esquerda.

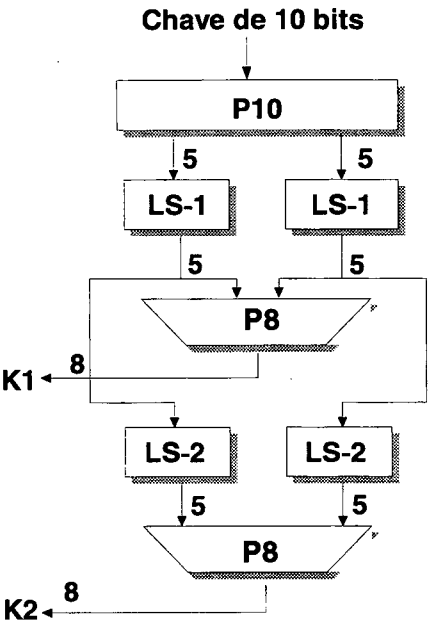


Figura 3.2: Geração das sub-chaves para o S-DES

Tabela 3.2: Permutação P8

6	3	7	4	8	5	10	9

O resultado é a sub-chave 1 (K_1[4]).

Agora retorna-se ao par de 5 *bits* gerados por LS-1 e executa-se uma rotação circular à esquerda de 2 *bits* em cada metade (LS-2)[5].

Finalmente, aplica-se P8 novamente para produzir K_2[6].

3.2.4 Cifração utilizando S-DES

A figura 3.3 ilustra detalhadamente o esquema de cifração do S-DES. Agora, cada uma das cinco funções que compõem o processo de cifração do S-DES será examinada.

[4]"K" é comumente utilizado, e vem do termo em inglês *Key*, ou chave.

[5]Novamente aqui, "LS" vem do termo em inglês *Left-Shift*, ou deslocamento circular à esquerda.

[6]Ou seja, a chave 2.

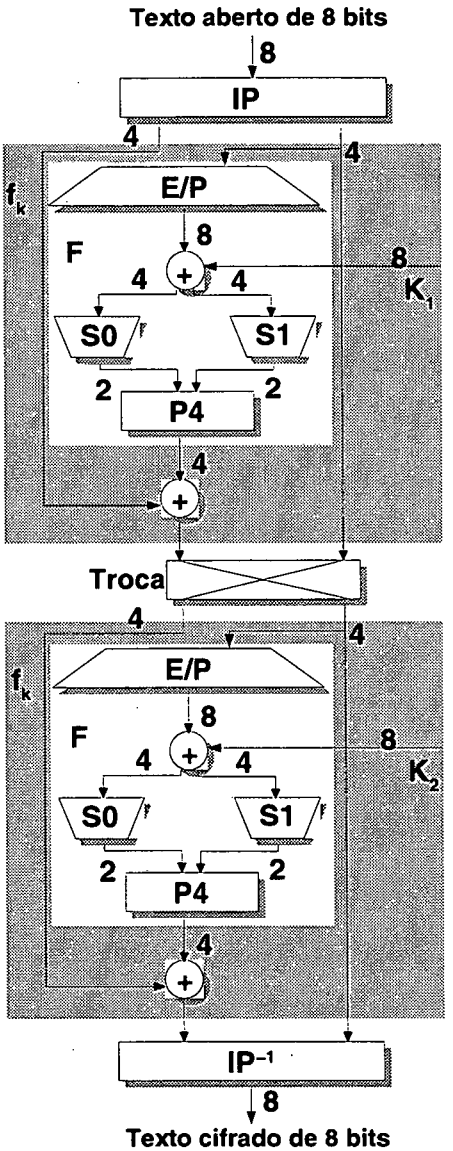


Figura 3.3: Esquema detalhado do processo de cifração do S-DES

Permutação inicial e final A entrada para o algoritmo é um texto aberto de 8 *bits*, sobre o qual é executada uma permutação inicial IP, conforme ilustrado na tabela 3.3.

Tabela 3.3: Permutação Inicial

2	6	3	1	4	8	5	7

Isto mantém todos os *bits* do texto aberto, mas os mistura. No final do

algoritmo, a permutação inversa (IP^{-1}), ilustrada na figura 3.4, é utilizada.

Tabela 3.4: Permutação Inversa

4	1	3	5	7	2	8	6

A função f_k O componente mais complexo do S-DES é a função f_k, que consiste na combinação de permutações e substituições. A função pode ser expressa da seguinte forma: sendo L^7 os quatro *bits* mais à esquerda e R^8 os quatro *bits* mais à direita dos 8 *bits* da entrada de f_k, e sendo F um mapeamento (não necessariamente um para um) de uma cadeia de 4 *bits* para outra cadeia de 4 *bits*. Então tem-se

$$f_k(L,R) = (L \oplus F(R,SK),R)$$

onde *SK* é uma sub-chave e $\oplus$ é uma função de OU Exclusivo *bit*-a-*bit*.

Agora, o mapeamento F será descrito. A entrada é um número de 4 *bits*. A primeira operação é uma expansão/permutação (E/P)[9] , que ocorre conforme a tabela 3.5.

Tabela 3.5: Expansão/Permutação

4	1	2	3	2	3	4	1

A sub-chave K1 é agora adicionada a este valor utilizando um OU Exclusivo.

Os primeiros quatro *bits* são colocados em uma caixa-S, S0, para produzir uma saída de 2 *bits*, e os 4 *bits* restantes são colocados em outra caixa-S, S1, para produzir outra saída de 2 *bits*. As duas caixas S são definidas conforme a tabela 3.6.

[7]"L" vem do termo em inglês *Left*, ou esquerda.

[8]"R" vem do termo em inglês *Right*, ou direita.

[9]"E/P" vem do termo em inglês *Expantion-Permutation*, ou Expansão/Permutation.

Tabela 3.6: Caixas S

S0 =	1	0	3	2	S1 =	0	1	2	3
	3	2	1	0		2	0	1	3
	0	2	1	3		3	0	1	0
	3	1	3	2		2	1	0	3

As caixas S trabalham da seguinte forma: os primeiro e quarto *bits* são tratados com um número de 2 *bits* que determina a linha da caixa-S, e o segundo e terceiro *bits* determinam a coluna da caixa-S. O número na linha e coluna determinadas é a saída da caixa-S.

A seguir, os quatro *bits* produzidos pelas caixas S0 e S1 passam por outra permutação P4, conforme a tabela 3.7.

Tabela 3.7: Permutação P4

2	4	3	1

A saída de P4 é a saída da função F.

A função de troca A função f_k altera somente os quatro *bits* mais à direita da entrada. A função de troca faz a troca entre os 4 *bits* da esquerda com os da direita para que a segunda rodada trabalhe sobre 4 *bits* diferentes. Nesta segunda rodada, as funções E/P, S0, S1 e P4 são as mesmas e a chave utilizada é K2.

3.2.5 Segurança do S-DES

Um ataque de força bruta ao S-DES é perfeitamente possível, uma vez que com uma chave de 10 *bits*, existem somente $2^{10} = 1024$ possibilidades. Dado um texto cifrado, um atacante poderia tentar cada uma das possibilidades e analisar o resultado para determinar se o texto aberto faz sentido.

3.2.6 Exemplo de Cifração utilizando S-DES

Para o melhor entendimento do algoritmo S-DES, o seguinte exemplo pode ser utilizado [SCH 96a]. A decifração da seqüência de *bits* 10100010 utilizando a chave 0111111101 será feita, sendo conhecido que os quatro primeiros *bits* correspondem a uma letra e os quatro últimos a outra, no intervalo de A-P, isto é:

A = 0000, B = 0001, ..., P = 1111

O texto aberto resultante é `OK`. O objetivo é que a decifração seja feita a mão, ou através do desenvolvimento de um pequeno programa.

A geração das sub-chaves é feita conforme a tabela 3.8

Tabela 3.8: Exemplo de geração da sub-chaves do S-DES

Operação		
K	=	01111 11101
P10	=	11111 10011
LS-1	=	11111 00111
P8	=	01011 111
LS-2	=	11111 00111
P8	=	11111 100

E produz as sub-chaves K_1 e K_2 que estão na tabela 3.9.

Tabela 3.9: Sub-chaves geradas

Sub-chaves		
K_1	=	0101 1111
K_2	=	1111 1100

Tendo as chaves geradas, o processo de decifração ocorre conforme ilustrado na tabela 3.10.

Tabela 3.10: Exemplo do processo de decifração do S-DES

Operação		Valor
Texto Cifrado	=	1010 0010
IP	=	0011 0001
E/P	=	1000 0010
K_2	=	1111 1100
E/P $\oplus$ K_2	=	1111 1100
Saída de S1	=	0000
P4	=	0000
Metade Esquerda	=	0011
P4 $\oplus$ Esquerda	=	0011
Saída de f_k	=	0011 0001
Troca	=	0001 0011
E/P	=	1001 0110
K_1	=	0101 1111
E/P $\oplus$ K_1	=	1100 1001
Saída de S0	=	0110
P4	=	1010
Metade Esquerda	=	0001
P4 $\oplus$ Esquerda	=	1011
Saída de f_k	=	1011 0011
IP^{-1}	=	1110 1010
Texto aberto	=	1110 1010

Convertendo os valores binários em letras, seguindo a notação que foi proposta, se chega aos valores representados na tabela 3.11.

Tabela 3.11: Resultado da decifração utilizando o S-DES

Código Binário	Valor Decimal	Letra
1110	14	O
1010	10	K

3.3 DES

3.3.1 Histórico do DES

O DES é o mais conhecido e mais amplamente utilizado cifrador de bloco simétrico [LAB 88]. Ele foi submetido em 1974 em resposta a um convite público feito, nos Estados Unidos, pelo NBS[10], para ser utilizado como um padrão de cifração de dados pelo Governo Americano e entidades privadas [MAS 88]. O projeto DES da IBM foi uma modificação de um antigo cifrador da empresa chamado LUCIFER, que utilizava chaves de 128 *bits* [MAS 88]. A NSA[11], que foi chamada para avaliar o algoritmo pelo NBS, fez duas coisas: modificou as constantes das caixas S e reduziu o comprimento original da chave de 128 *bits* para 56 *bits* [SCH 00].

Apesar das críticas devido à redução da chave e das mudanças sem explicação no projeto das caixas S, o DES foi adotado como um FIPS em 1976, sob o nome FIPS PUB 46, e, posteriormente, também por outras organizações de padrões ao redor do mundo, como ANSI (ANSI X3.92) e ISO [SCH 00]. No desenvolvimento do DES, pela primeira vez um algoritmo avaliado pela NSA foi tornado público, e ele foi um dos dois mais importantes desenvolvimentos para o crescimento da pesquisa pública da criptografia, ao lado da invenção da criptografia por chave pública [SCH 00].

O consenso geral dos pesquisadores de criptologia atualmente é que o DES é um cifrador extremamente bom com uma chave, infelizmente, muito pequena [MAS 88]. O comprimento efetivo da chave pode ser aumentado utilizando múltiplas cifrações do DES com chaves diferentes [MAS 88]. Se o DES não é atualmente inseguro,

[10]National Bureau of Standards - Escritório Nacional de Padronização

[11]National Security Agency - Agência de Segurança Nacional

ele será em breve, pelo simples fato de que chaves de 56 *bits* estão se tornando vulneráveis à pesquisa exaustiva [LAB 88]. O NIST tem recertificado o DES a cada cinco anos, tendo sido recertificado pela última vez em 1993, não sendo mais recertificado a partir de então [LAB 88]. A partir de 1998 o DES não foi mais permitido para uso do Governo Americano. O 3DES ou DES Triplo, que consiste de três execuções seguidas do DES sobre o mesmo bloco de texto aberto, será utilizado até que o AES, que irá substituí-lo, esteja pronto para o uso geral [LAB 88].

3.3.2 Características do DES

O DES é um algoritmo de criptografia simétrica, mais especificamente um cifrador de Feistel com 16 rodadas e foi originalmente projetado para implementação em hardware [LAB 88], devido a sua natureza repetitiva [SCH 96b], não sendo muito eficiente em software, especialmente nas arquiteturas de 32 *bits* comuns hoje [SCH 00]. O DES é um cifrador de blocos, cifrando e decifrando dados em blocos de comprimento fixo [SCH 00]. O DES é um cifrador de produto ou de iteração, significando que ele contém 16 iterações ou rodadas de um cifrador mais simples [SCH 00], contendo operações simples como substituições e permutações [SCH 96b].

A principal força do algoritmo vem das caixas S, uma operação de procura não-linear em uma tabela que substitui grupos de 6 *bits* por grupos de 4 *bits*. O DES possui um comprimento de bloco de 64 *bits* e usa uma chave de 56 *bits* durante a execução, sendo tirados 8 *bits* de paridade da chave de 64 *bits* completa [LAB 88]. Embora não existam provas, alguns autores apontam que a redução da chave de 64 *bits* para 56 *bits* teve o objetivo de, intencionalmente, reduzir o custo da busca exaustiva da chave por um fator de 256 [MEN 96].

3.3.3 Funcionamento do algoritmo DES

A figura 3.4 ilustra o esquema de funcionamento do cifrador DES. No anexo A, pode-se verificar uma implementação do cifrador DES em linguagem C. O DES trabalha sobre blocos de 64 *bits* de texto aberto. Após uma permutação inicial, o

bloco é quebrado em uma metade esquerda e uma direita, com 32 *bits* cada uma. Então ocorrem 16 rodadas de operações idênticas, chamadas de Função F, em que os dados são combinados com a chave. Após a décima sexta rodada, as metades esquerda e direita são unidas, e uma permutação inversa à inicial finaliza o algoritmo [SCH 96b].

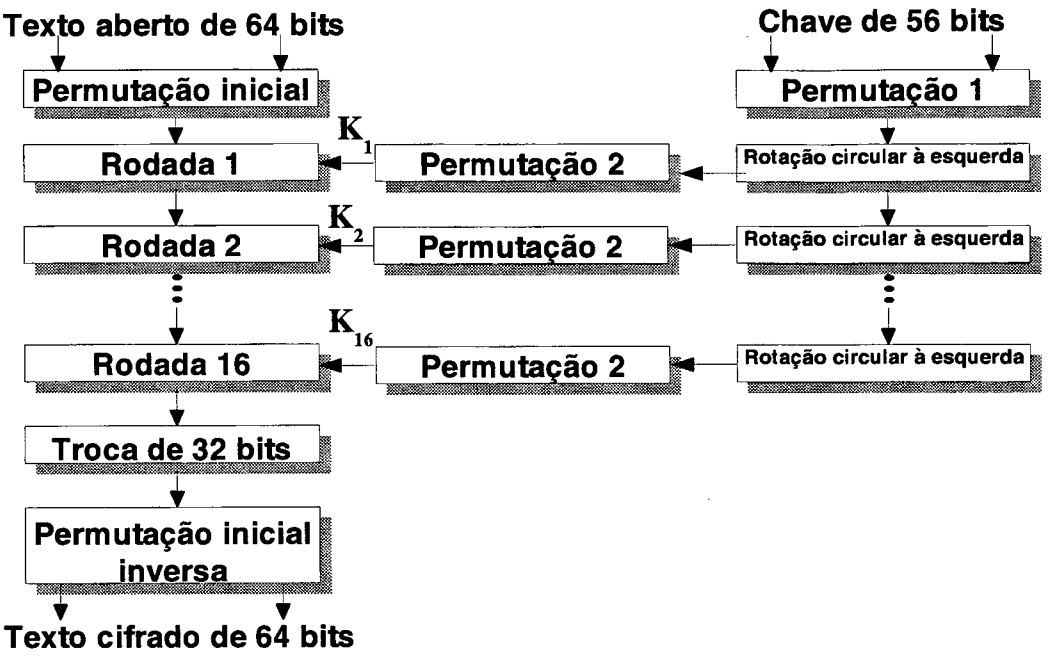


Figura 3.4: Esquema geral do DES

Em cada rodada, os *bits* da chave são deslocados, e então 48 *bits* são selecionados dos 56 *bits* da chave, produzindo as sub-chaves K_i. A metade direita dos dados é expandida para 48 *bits* através de uma permutação de expansão, combinada com os 48 *bits* de uma chave deslocada e permutada através de um OU Exclusivo, enviada através de 8 caixas S produzindo 32 *bits*, e permutada novamente. Estas quatro operações definem a Função F. A saída da Função F é então combinada com a metade esquerda através de outro OU Exclusivo. O resultado destas operações produz a nova metade esquerda. Estas operações são repetidas 16 vezes, fazendo as 16 rodadas do DES [SCH 96b].

Ao final deste processo, são obtidos os 64 *bits* do texto cifrado. O mesmo processo se repete para cada bloco de texto aberto da dado a ser cifrado.

3.3.4 Geração das sub-chaves do DES

A chave de 64 *bits* do DES é inicialmente reduzida para uma chave de 56 *bits*, ignorando cada oitavo *bit*, que são usados como verificação de paridade. Desta chave de 56 *bits* são geradas 16 diferentes sub-chaves de 48 *bits* para cada uma das rodadas do DES [SCH 96b].

As sub-chaves são geradas tomando-se a chave de 56 *bits* e dividindo-a em duas metades de 28 *bits*, que sofrem então uma deslocamento circular à esquerda de 1 ou 2 *bits*, dependendo da rodada, e então selecionando-se 48 *bits* dos 56 *bits* da chave [SCH 96b].

3.3.5 Modos de Operação do DES

O algoritmo DES é um bloco básico de construção cujo objetivo é fornecer segurança de dados. Para aplicar o DES a uma variedade de aplicações, quatro *"modos de operação"* foram definidos (FIPS PUB 74,81). São eles os modos ECB, CBC, CFB e OFB, que serão descritos logo em seguida. A intenção é que com estes quatro modos de operação seja possível cobrir todos os tipos de aplicações onde o DES poderia ser utilizado. Os mesmos modos de operação podem ser aplicados a qualquer cifrador de bloco simétrico [STA 99].

ECB *("Electronic Codebook"):* O modo ECB simplesmente cifra cada 64 *bits* de texto aberto, um após o outro, utilizando a mesma chave DES de 56 *bits* [LAB 88]. Para uma mensagem maior que o tamanho do bloco, o procedimento consiste em quebrar a mensagem em blocos de 64 *bits*, completando o último bloco se necessário. A decifração é feita um bloco de cada vez, utilizando a mesma chave. Na figura 3.5, o texto aberto consiste de uma seqüência de blocos de 64 *bits*, P_1, P_2, ..., P_N; a seqüência correspondente de blocos do texto cifrado é C_1, C_2, ..., C_N.

O modo ECB é ideal para pequenas quantidades de dados, como uma chave de cifração. A principal característica do ECB é que se um mesmo bloco de 64 *bits* de texto aberto aparece mais de uma vez em uma mensagem, ele sempre produzirá o

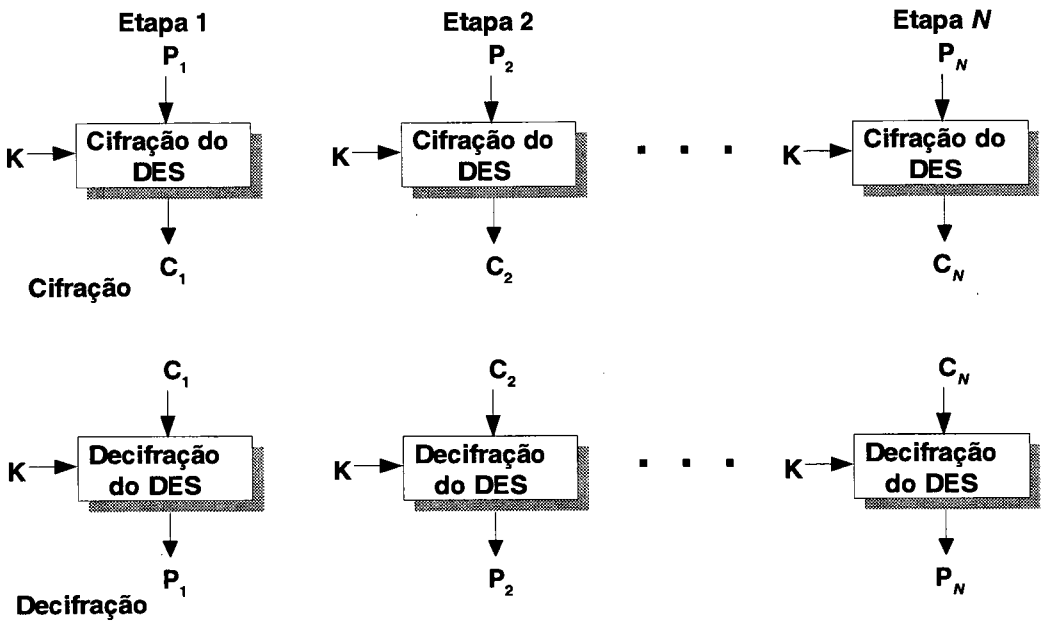


Figura 3.5: Modo de operação ECB

mesmo texto cifrado. Para textos longos, o modo ECB pode não ser seguro. Se a mensagem é bem estruturada, é possível explorar estas regularidades. Um exemplo deste tipo de mensagens são arquivos de imagem *bitmap* ou de texto, que possuem uma estrutura bem conhecida e grandes quantidades de dados redundantes, como espaços em branco.

CBC *("Cipher Block Chaining"):* Para resolver as deficiências de segurança do ECB, faz-se necessária uma técnica em que o mesmo bloco de texto aberto, se repetido, produza diferentes blocos de texto cifrado. Uma forma simples de satisfazer este requisito é o modo CBC, cujo esquema aparece na figura 3.6. No modo CBC é feito um OU Exclusivo em cada bloco de 64 *bits* de texto aberto com o bloco de texto cifrado anterior, antes de ser cifrado com a chave DES [LAB 88]. Desta forma, a cifração de cada bloco depende dos blocos anteriores e o mesmo texto aberto pode ser cifrado em textos cifrados diferentes dependendo de seu contexto na mensagem completa.

Para produzir o primeiro bloco de texto cifrado, é feito um OU Exclusivo entre um vetor de inicialização (IV[12]) e o primeiro bloco de texto aberto. O vetor de

[12]"IV" vem do termo em inglês *Initialization Vector*, ou Vetor de Inicialização.

inicialização precisa ser conhecido tanto pelo emissor quanto pelo receptor, e deveria ser tão protegido quanto a senha. Devido ao mecanismo de encadeamento do CBC, ele é um modo apropriado para a cifração de mensagens de comprimento superior a 64 *bits*.

O modo CBC, que é na prática o mais amplamente utilizado, ajuda a proteger contra certos tipos de ataques, mas não auxilia na proteção contra a busca exaustiva ou a criptoanálise diferencial [LAB 88].

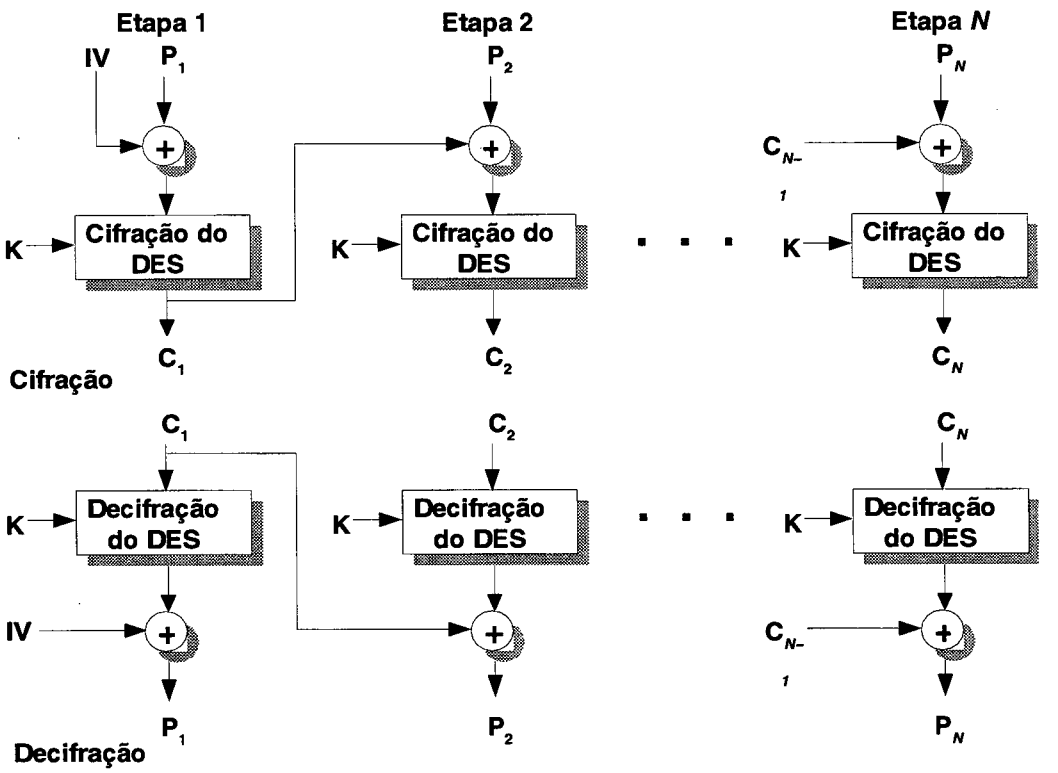


Figura 3.6: Modo de operação CBC

CFB ***("Cipher Feedback"):*** O modo CFB permite que o DES seja utilizado com blocos menores do que 64 *bits*.

OFB ***("Output Feedback"):*** O modo OFB permite que o DES seja utilizado como um cifrador de *fluxo* [LAB 88].

3.3.6 Segurança do DES

Durante muito tempo a segurança do DES tem sido alvo dos mais diversos tipos de questionamentos [SCH 96b], sobretudo o comprimento da chave, o número de rodadas e o projeto das caixas S. As caixas S são particularmente misteriosas - todas aquelas constantes, sem uma aparente razão do porquê ou para quê elas são assim. A discussão se concentra em torno de possíveis *"trap-doors"* colocados pela IBM e retirados pela NSA ou até mesmo adicionados pela própria NSA [SCH 96b].

Assumindo que seja possível obter uma enorme quantidade de pares de textos abertos conhecidos, a criptoanálise linear fornece o mais poderoso ataque ao DES atualmente, não sendo no entanto, considerada uma ameaça em termos práticos [MEN 96].

A criptoanálise diferencial é um método de ataque de texto aberto escolhido, envolvendo a comparação do OU Exclusivo de dois textos planos com o OU Exclusivo dos dois textos cifrados correspondentes [STI 95].

3.3.7 Chaves fracas

O DES possui cerca de 64 chaves consideradas fracas ou semi-fracas, o que é um número muito pequeno quando comparado ao total de possíveis chaves (2^{56}). Estas chaves podem ser simplesmente desprezadas, uma vez que a chance de serem usadas é muito pequena, ou podem ser verificadas e descartadas durante o processo de geração de chaves.

3.3.8 DES Triplo

Devido ao potencial de vulnerabilidade do DES a um ataque de força bruta, tem havido um considerável interesse em se encontrar uma alternativa ao seu uso. Uma abordagem seria o projeto de um algoritmo completamente novo, como é o caso do AES, que irá substituir o DES. Outra alternativa, que poderia preservar o investimento existente em software e equipamento, seria utilizar cifração múltipla com o DES e múltiplas chaves. O DES Triplo é uma destas propostas. Ele consiste na cifração do

mesmo texto aberto pelo DES, 3 vezes seguidas. Alguns modos de cifração tripla foi proposto: [LAB 88]:

- DES-EEE3: 3 cifrações DES com 3 chaves diferentes;
- DES-EDE3: 3 operações DES na seqüência cifra-decifra-cifra, utilizando 3 chaves diferentes;
- DES-EEE2 e DES-EDE2: idênticos aos formatos anteriores, exceto que a primeira e terceira operações utilizam a mesma chave.

De todas as formas de cifração múltipla com o Triplo DES propostas, a mais segura é o Triplo DES com 3 chaves distintas, ou seja, o DES-EEE3 [LAB 88].

3.4 Relação entre S-DES e DES

Um esquema de cifração do DES pode ser definido como

$$IP^{-1} \circ f_{k_{16}} \circ \text{Troca} \circ f_{k_{15}} \circ \ldots \circ f_{k_{16}} \circ \text{Troca} \circ f_{k_1} \circ IP$$

O DES trabalha em blocos de entrada de 64 *bits*, e com uma chave de 56 *bits*, da qual são calculadas 16 sub-chaves de 48 *bits*. Há uma permutação de 56 *bits* seguida de uma seqüência de 16 rodadas de deslocamentos e permutações de 48 *bits*. O mapeamento F trabalha com 32 *bits*, ao invés de 4 *bits*, e existem 8 caixas S, cada uma com 4 linhas e 16 colunas. É possível gerar 2^{56} textos cifrados diferentes para cada texto aberto.

O S-DES, por sua vez, trabalha em blocos de entrada de 8 *bits*, e com uma chave de 10 *bits*, da qual são calculadas 2 sub-chaves de 8 *bits*. Há uma permutação de 8 *bits* seguida de uma seqüência de 2 rodadas de deslocamentos e permutações de 8 *bits*. O mapeamento F trabalha com 8 *bits*, e existem 2 caixas S, cada uma com 4 linhas e 4 colunas. É possível gerar 2^{10} textos cifrados diferentes para cada texto aberto .

Na tabela 3.12 um resumo das características do DES e S-DES é apresentado.

Tabela 3.12: Relação entre DES e S-DES

Característica	S-DES	DES
Comprimento do bloco de entrada	8	64
Comprimento do bloco de saída	8	64
Comprimento da Chave	10	56
Número de sub-chaves geradas	2	16
Comprimento de cada sub-chave	8	48
Número de rodadas	2	16
Número de caixas S	2	8
Comprimento das caixas S	4x4	4x16
Número de textos cifrados possíveis	2^{10}	2^{56}

3.5 Conclusão

Um modelo simplificado de um cifrador pode facilitar em muito o entendimento e a análise de um cifrador real. Neste capítulo, em particular, o modelo simplificado S-DES foi apresentado. Este modelo permite o fácil entendimento do DES. O DES é utilizado em aplicações práticas, enquanto que o uso do S-DES é feito apenas no meio acadêmico. Após ter sido plenamente entendido o S-DES, torna-se mais simples entender o próprio DES, pois pode-se fazer uma relação entre ambos.

Capítulo 4

Cifradores de Ron Rivest

4.1 Introdução

Ron Rivest é professor do MIT, co-fundador da RSA Data Security e um respeitável e competente criptógrafo. Ele é um dos responsáveis pelo desenvolvimento do algoritmo de criptografia por chave pública RSA, dos algoritmos de *hash* MD2, MD4 e MD5 e também da série de algoritmos simétricos conhecidos pela sigla "RC", que para muitos significa "Ron's Code" ou ainda "Rivest Cipher". Ao todo, Ron Rivest trabalhou em 6 algoritmos, denominados RC1, RC2, RC3, RC4, RC5 e RC6. O RC1 nunca saiu das anotações de Ron Rivest, enquanto o RC3 foi "quebrado" na RSA Data Security durante o seu desenvolvimento [SCH 96b, STA 99]. O algoritmo RC4 é um cifrador de fluxo, enquanto que os algoritmos RC2, RC5 e RC6 são todos cifradores de bloco, e apresentam uma perceptível evolução, partindo do RC2 até chegar ao RC6.

Este capítulo descreve os cifradores RC2, RC4, RC5 e RC6, definindo os motivos que levaram à escolha do RC6 para o desenvolvimento de um modelo simplificado. As características e o funcionamento do RC6 são também abordadas em detalhes. É mostrado o conjunto de passos utilizados no desenvolvimento do RC6 a partir do RC5, deixando claro que houve uma evolução do RC5 para se chegar ao RC6. Por fim, uma tabela compara as diversas características de cada um dos cifradores de Ron Rivest.

4.2 RC2

O RC2 foi desenvolvido em 1989 como um algoritmo confidencial e proprietário, e depois publicado como um *Internet Draft* em 1997 [KNU 98]. O RC2 possui muitas características de projeto interessantes e únicas, principalmente quando considerado o estilo de cifradores que dominava a literatura e o mercado na época de sua criação. Entre estas características, pode-se citar o deslocamento circular à esquerda, o uso de vários operadores, como adição, e os operadores OU Exclusivo e "E", o tamanho da chave variável e o processo longo na geração das sub-chaves. Ele foi desenvolvido para ser particularmente eficiente e fácil de implementar em processadores de 16 *bits*, que eram muito comuns na época de seu desenvolvimento. O RC2 é um cifrador de bloco simétrico que utiliza comprimentos de bloco de texto aberto e texto cifrado de 64 *bits* e comprimento de chave que varia de 8 a 1024 *bits* de comprimento, sendo 40 *bits* um comprimento de chave comum para a adequação às leis de exportação de produtos de criptografia dos Estado Unidos. A intenção era que ele viesse a ser um substituto para o DES uma vez que a RSA afirma que o RC2 é de 2 a 3 vezes mais rápido do que o DES [SCH 96b]. Além disso, com uma chave maior, o RC2 pode ser significativamente mais seguro que o DES [SCH 96b, STA 99]. A velocidade de cifração do RC2 independe do comprimento da chave. Uma característica significativa do RC2 é a flexibilidade oferecida ao usuário em termos de comprimento de chave efetivo. Esta flexibilidade tornou-se uma característica comum em muitas propostas de cifradores de bloco e é uma propriedade que provou ser importante em aplicações comerciais. O RC2 tem sido bastante divulgado, aparecendo com destaque no padrão de mensagens seguras S/MIME, onde é utilizado com chaves de 40, 64 e 128 *bits* [KNU 98].

O algoritmo do RC2 é dividido em duas partes principais, sendo a primeira a *expansão da chave* e a outra a *cifração* propriamente dita. A expansão da chave gera 128 *bytes* de sub-chaves, que são armazenados no vetor L[0], L[1],...,L[127]. É feita então uma operação sobre o vetor L, utilizando um vetor auxiliar P[0], P[1],...,P[255], que é baseado nos dígitos de π (3,141592653...).

As operações primitivas utilizadas pelo RC2 são [STA 99]:

- **Adição:** Adição de palavras, indicada por +, é executada modulo 2^{16}. A operação inversa, indicada por -, é a subtração módulo 2^{16}.

- **OU Exclusivo *bit*-a-*bit*:** Indicado por $\oplus$.

- **Complemento *bit*-a-*bit*:** Indicado por $\sim$.

- **E *bit*-a-*bit*:** Indicado por &.

- **Rotação circular à esquerda:** A rotação cíclica da palavra x à esquerda em y *bits* é indicada por $x \lll y$.

O RC2 não utiliza a estrutura clássica de Feistel, tornando difícil a comparação com outros algoritmos. Ele pega os 64 *bits* de entrada divididos em 4 elementos de 16 *bits* R[0], R[1], R[2] e R[3] e os coloca novamente nestes elementos após 18 rodadas de 2 tipos: *mixing* e *mashing*. Cada rodada utiliza quatro sub-chaves. Existem 16 rodadas do tipo *mixing*, de forma que todas as sub-chaves são utilizadas uma vez. Além disso, sub-chaves são selecionadas de forma dependente dos dados para as rodadas do tipo *mashing*.

A decifração é executada como a operação inversa da cifração, de forma que as rodadas são executadas e as chaves são utilizadas na ordem inversa.

A RSA afirma que, ao contrário da cifração e da decifração, a procura exaustiva pela chave não é rápida. Uma quantidade significativa de tempo é gasta definindo as sub-chaves. Embora este tempo seja desprezível na cifração e decifração de mensagens, ele é muito significativo quando se está tentando todas as possíveis chaves [SCH 96b].

4.3 RC4

O RC4 é um cifrador simétrico de fluxo não sendo uma evolução do RC2, que é um cifrador de bloco. Ele foi desenvolvido em 1987 e durante sete anos foi proprietário, sendo mantido como um segredo comercial pela RSA Data Security. Em

9 de Setembro de 1994, o algoritmo foi postado anonimamente na Internet na lista de mensagens *Cypherpunks*[1]. A mensagem completa está reproduzida no anexo C, e contém o código fonte do RC4 juntamente com vetores de teste, que podem ser utilizados na verificação do funcionamento do cifrador.

O RC4 está presente em vários produtos comerciais de criptografia, incluindo o Lotus Notes e o Oracle Secure SQL. Ele é também parte da especificação de Pacotes de Dados para Telefonia Celular Digital.

O RC4 também trabalha em duas fases, a *expansão da chave* e a *cifração*. A expansão da chave é a primeira e mais difícil fase do algoritmo. A cifração consiste numa operação de OU Exclusivo entre a chave e o texto aberto. O RC4 é cerca de 10 vezes mais rápido que o DES [SCH 96b].

4.4 RC5

O RC5 é um cifrador de bloco simétrico, que possui as seguintes características [RIV 94]:

- **Ajustável para hardware e software:** utiliza somente operações primitivas computacionais encontradas com freqüência nos microprocessadores;

- **Rapidez:** Para conseguir esta rapidez, o RC5 é um algoritmo simples e é orientado para palavra, isto é, as operações básicas trabalham em palavras completas de dados de cada vez;

- **Adaptável a processadores de diferentes comprimentos de palavras:** O número de *bits* numa palavra é um parâmetro do RC5; diferentes comprimentos de palavras produzem diferentes algoritmos. Isto significa dizer que se um texto aberto é cifrado utilizando o RC5 com determinados parâmetros, ele deve ser decifrado utilizando os mesmos valores para estes parâmetros;

[1] `http://cypherpunks.venona.com/date/1994/09/msg00434.html`

- **Número de rodadas variável:** O número de rodadas é o segundo parâmetro do RC5. Este parâmetro permite uma negociação entre maior velocidade e maior segurança;
- **Comprimento de chave variável:** O comprimento da chave é o terceiro parâmetro do RC5. Novamente, ele permite uma negociação entre velocidade e segurança;
- **Simplicidade:** A estrutura simples do RC5 é de fácil implementação e facilita a tarefa de determinar a força do algoritmo;
- **Baixos requisitos de memória:** Um baixo requisito de memória torna o RC5 ajustável para *smart cards* e outros dispositivos com restrição de memória;
- **Alta segurança:** O RC5 foi projetado para fornecer alta segurança com parâmetros ajustáveis;
- **Rotações dependentes dos dados:** O RC5 incorpora rotações (deslocamento circular de *bits*) cuja quantidade é dependente dos dados.

O RC5 possui muitas características semelhantes ao RC2, podendo ser considerado uma evolução deste. As semelhanças estão nas primitivas utilizadas no uso de rotação de dados, no uso de chaves de comprimento variável e na expansão da chave.

4.4.1 Parâmetros do RC5

O RC5 é na verdade uma família de algoritmos de cifração determinados por três parâmetros, conforme descrito na tabela 4.1.

Desta forma, o RC5 cifra blocos de texto aberto de comprimento 32, 64 ou 128 *bits* em blocos de texto cifrado do mesmo comprimento. O comprimento da chave varia de 0 até 2040 *bits*. Uma versão específica do RC5 é designada como RC5-*w/r/b*. Por exemplo, o RC5-32/12/16 possui palavras de 32 *bits* (texto aberto e texto cifrado de 64 *bits*), 12 rodadas nos algoritmos de cifração e decifração e uma chave de 16 *bytes* (128 *bits*) de comprimento. Ron Rivest sugere o uso de RC5-32/12/16 como a versão "nominal". É importante que o comprimento de *w* seja o mesmo do comprimento da

Tabela 4.1: Parâmetros do RC5

Parâmetro	Definição	Valores Permitidos
w	Comprimento da palavra em *bits*. O RC5 cifra blocos de 2 palavras	16, 32, 64
r	Número de rodadas.	0, 1, ..., 255
b	Número de *bytes* na chave secreta K.	0, 1, ..., 255

palavra utilizada pela CPU onde o algoritmo está sendo executado. Desta forma todas as operações são feitas utilizando o tamanho "natural" para o processador, aumentando o desempenho do algoritmo. A possibilidade de modificação do comprimento da palavra permite adaptar o algoritmo para CPUs com diferentes comprimentos de palavra.

4.4.2 Operações primitivas do RC5

As operações primitivas utilizadas pelo RC5 são [STA 99]:

- **Adição:** Adição de palavras, indicada por +, é executada modulo 2^w, isto é, o resultado é um valor de *w bits*. A operação inversa, indicada por -, é a subtração módulo 2^w.
- **OU Exclusivo *bit*-a-*bit*:** Indicado por $\oplus$.
- **Rotação circular à esquerda:** A rotação cíclica da palavra x à esquerda em y *bits* é indicada por $x \lll y$. O inverso é a rotação circular à direita da palavra x em y *bits*, indicada por $x \ggg y$.

4.4.3 Expansão da chave do RC5

O RC5 executa um conjunto complexo de operações na chave secreta para produzir um total de t sub-chaves. Duas sub-chaves são utilizadas em cada rodada e duas sub-chaves são utilizadas em uma operação adicional que não faz parte de nenhuma rodada, então $t = 2r + 2$. Cada sub-chave possui uma palavra (*w bits*) de comprimento.

A figura 4.1 ilustra a técnica utilizada para gerar as sub-chaves do RC5. A figura 4.2 apresenta o algoritmo de expansão da chave do RC5. As sub-chaves são armazenadas num vetor de t palavras definido como S[0], S[1], ..., S[t - 1]. Usando os parâmetros r e w como entrada, este vetor é inicializado com um padrão de *bits* pseudo-randômico fixo. Esta inicialização faz uso de duas constantes denominadas P_w e Q_w que são definidas como:

$$P_w = \text{Ímpar}[(e - 2^w)]$$
$$Q_w = \text{Ímpar}[(\phi - 2^w)]$$

onde

e = 2,718281828459 ... (base dos logaritmos naturais[2])

ϕ = 1,618033988749 ... (razão dourada) = $\left(\frac{1+\sqrt{5}}{2}\right)$

e Ímpar[x] é o inteiro ímpar mais próximo de x (arredondado para cima se x é par).

As constantes P_w e Q_w variam de comprimento de acordo com o comprimento de w. Dados os valores permitidos de w, as constantes são as que aparecem na tabela 4.2 (em hexadecimal). Depois de inicializado, o vetor S é misturado com o vetor da chave L para produzir um vetor S final de sub-chaves, conforme o algoritmo da figura 4.2.

Tabela 4.2: Valores das constantes P_w e Q_w em hexadecimal

w	16 *bits*	32 *bits*	64 *bits*
P_w	B7E1	B7E15163	B7E151628AED2A6B
Q_w	9E37	9E3779B9	9E3779B97F4A7C15

[2]O uso deste valores é arbitrário. A principal razão de sua utilização é que são números fáceis de gerar e que possuem um certo grau de imprevisibilidade na seqüência dos algarismos

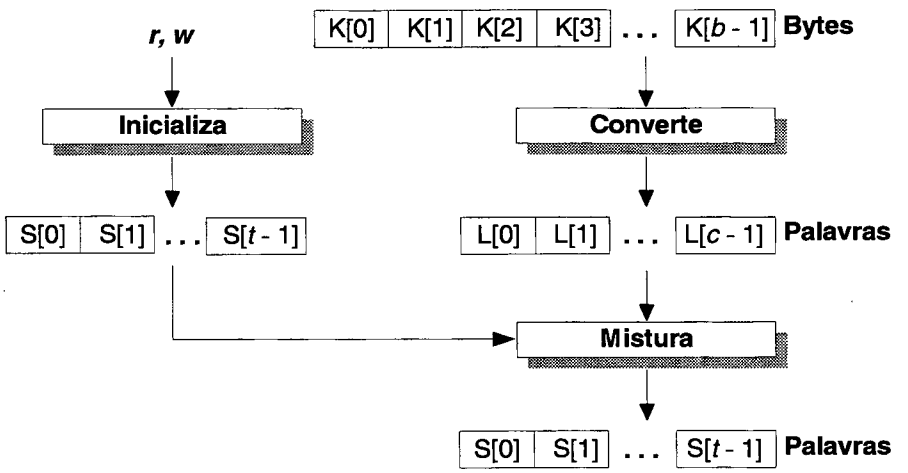


Figura 4.1: Esquema da expansão de chave do RC5

Entrada: r, w, b, chave K

Saída: 2r + 2 sub-chaves de w *bits* armazenadas em S[0], ..., S[2r+1]

Processo:

S[0] = P_w

for i = 1 to 2r + 2 do

  S[i] = S[i-1] + Q_w

A = B = i = j = 0

v = 3 x max{c,2r + 2}

for s = 1 to v do {

  A = S[i] = (S[i] + A + B) $\lll$ 3

  B = L[j] = (L[j] + A + B) $\lll$ (A + B)

  i = (i + 1) mod (2r + 2)

  j = (j + 1) mod c

}

Figura 4.2: Algoritmo de expansão da chave do RC5

4.4.4 Algoritmo de cifração do RC5

Tanto na cifração quanto na decifração este algoritmo supõe que o computador armazena os *bytes* em modo *little-endian*[3], ou seja, tanto o texto aberto como o texto cifrado são armazenados da seguinte forma nas variáveis A e B, para w=32: os primeiros quatro *bytes* do texto (x_0,x_1, x_2, x_3) são armazenados em A na seqüência (x_3,x_2, x_1, x_0), e os quatro *bytes* seguintes do texto (x_4,x_5, x_6, x_7) são armazenados em B na seqüência (x_7,x_6, x_5, x_4).

A figura 4.3 ilustra o esquema de cifração do RC5. Como pode ser visto, esta não é uma estrutura clássica de Feistel. Inicialmente, o texto aberto está em dois registradores A e B de $2w$ *bits* de comprimento.

O texto cifrado resultante estará contido nas duas variáveis A e B. Cada uma das r rodadas consiste de substituições usando ambas as palavras de dados, uma permutação usando ambas as palavras de dados e uma substituição que depende da chave. A operação é excepcionalmente simples, sendo definida em 5 linhas de código, mostradas na figura 4.4 [RIV 94]. Ambas as metades do dado são atualizadas em cada rodada, tornando o RC5 de certa forma equivalente a duas rodadas do DES.

```
A = A + S[0]
B = B + S[1]
for i = 1 to r do {
   A = ((A ⊕ B) ≪ B )+ S[2i]
   B = ((B ⊕ A) ≪ A )+ S[2i + 1]
}
```

Figura 4.4: Algoritmo da cifração do RC5

4.4.5 Algoritmo de decifração do RC5

O algoritmo de decifração do RC5 é facilmente derivado do algoritmo de cifração e está demonstrado na figura 4.5 [RIV 94].

[3]pequeno endiano

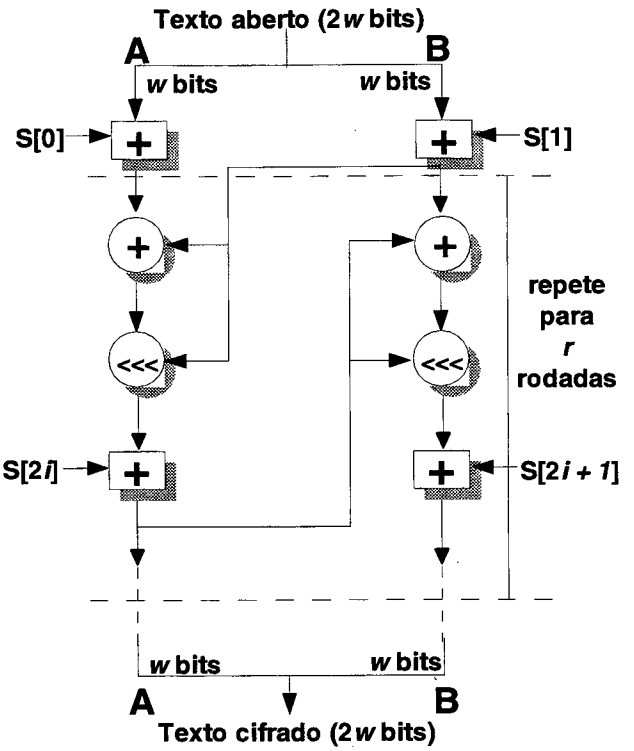


Figura 4.3: Esquema geral do RC5

```
for i = r downto 1 do {
   B = ((B - S[2i + 1]) ⋙ A) ⊕ A
   A = ((A - S[2i]) ⋙ B) ⊕ B
}
B = B - S[1]
A = A - S[0]
```

Figura 4.5: Algoritmo da decifração do RC5

4.4.6 Modos de operação do RC5

A RFC 2040 [BAL 96] define quatro modos de operação para o RC5:

- **Cifrador de Bloco:** Este é o algoritmo de cifração puro, que pega um bloco de entrada de comprimento fixo (2*w bits*) e produz um bloco de texto cifrado de mesmo

comprimento utilizando uma transformação que depende da chave. Este modo é também conhecido como modo ECB, conforme explicado no capítulo 3.

- **CBC:** O Modo CBC também foi explicado no capítulo 3. O modo CBC processa mensagens de comprimento múltiplo do comprimento do bloco do RC5 (*2w bits*). O modo CBC fornece uma segurança maior comparado ao modo ECB porque blocos repetidos de texto aberto produzem blocos diferentes de texto cifrado.

- **CBC-Pad:** Este é um modo ao estilo do CBC que manipula texto aberto de qualquer comprimento. O texto cifrado será maior que o texto aberto em, no máximo, o tamanho de um bloco do RC5 (*2w bits*).

- **CBC-CTS:** É também um modo ao estilo do CBC que manipula texto aberto de qualquer comprimento e produz texto cifrado de igual comprimento.

4.4.7 Considerações a respeito do RC5

É consenso entre os especialistas que após 8 iterações todo *bit* de texto aberto afeta pelo menos uma rotação. Para saber mais sobre a segurança do RC5, consulte [KAL 95].

As duas características marcantes do RC5 são a simplicidade do algoritmo e o uso das rotações dependentes dos dados. As rotações são a única parte não-linear do algoritmo; não existem tabelas de substituição ou outros operadores não-lineares, como o uso de caixas S, feito pelo DES e outros cifradores. A força do RC5 depende em grande parte das propriedades criptográficas das rotações dependentes dos dados. Ron Rivest acredita que devido à quantidade de rotações variar dependendo do valor do dado que se move através do algoritmo, as criptoanálises linear e diferencial deveriam ser muito difíceis. Em [YIN 97] é afirmado que vários estudos confirmam esta suposição.

4.5 RC6

O RC6 é um cifrador de bloco simétrico que foi desenvolvido para ser submetido ao NIST, para consideração como o novo AES [RIV 98b]. Ele é um melhoramento do RC5 e é resultado dos estudos desenvolvidos por Ron Rivest, Matthew J. B. Robshaw, Ray Sidney e Yiqun Lisa Yin, pesquisadores do MIT e dos Laboratórios RSA, em 1998. A implementação do RC6 em linguagem C está disponível no anexo E.

A figura 4.6 apresenta o esquema geral do RC6. O RC6 é baseado na estrutura de Feistel, mas de forma diferente de outros cifradores, como por exemplo, o DES. Nestes cifradores, a estrutura de Feistel é utilizada sobre metade do bloco a ser cifrado a cada rodada. Já no caso do RC6, o bloco é dividido em quatro partes, chamadas A, B, C e D, e a estrutura atua sobre 2 pares destas partes em cada rodada. Estas partes são então ligadas pela troca de alguns dados, na operação de troca dos registradores.

O projeto do RC6 começou com a consideração do RC5 sendo um candidato em potencial à submissão ao AES, sendo feitas modificações para satisfazer aos requisitos do AES, tais como aumentar a segurança e melhorar o desempenho. As principais melhorias foram a utilização do número de rotações dependente de todos os dados e o uso da multiplicação de inteiros de 32 *bits*, que é uma primitiva dos processadores modernos executada rapidamente e que aumenta a difusão em cada fase processada.

4.5.1 Parâmetros do RC6

Como o RC5, o RC6 é um algoritmo parametrizado, ou uma família de algoritmos em função destes algoritmos. Ele recebe 3 parâmetros na sua execução. Uma versão de RC6 é definida sempre como RC6-*w*/*r*/*b* onde a palavra possui *w bits*, executando *r* rodadas e utilizando chaves de *b bytes*. Por exemplo, para satisfazer os requisitos do AES, utiliza-se palavras de 32 *bits*, 20 rodadas e chaves de 16, 24 ou 32 *bits*. A operação ocorre sempre utilizando-se 4 registradores de *w bits* [RIV 98b]. Os valores permitidos para os parâmetros do RC6 são descritos na tabela 4.3.

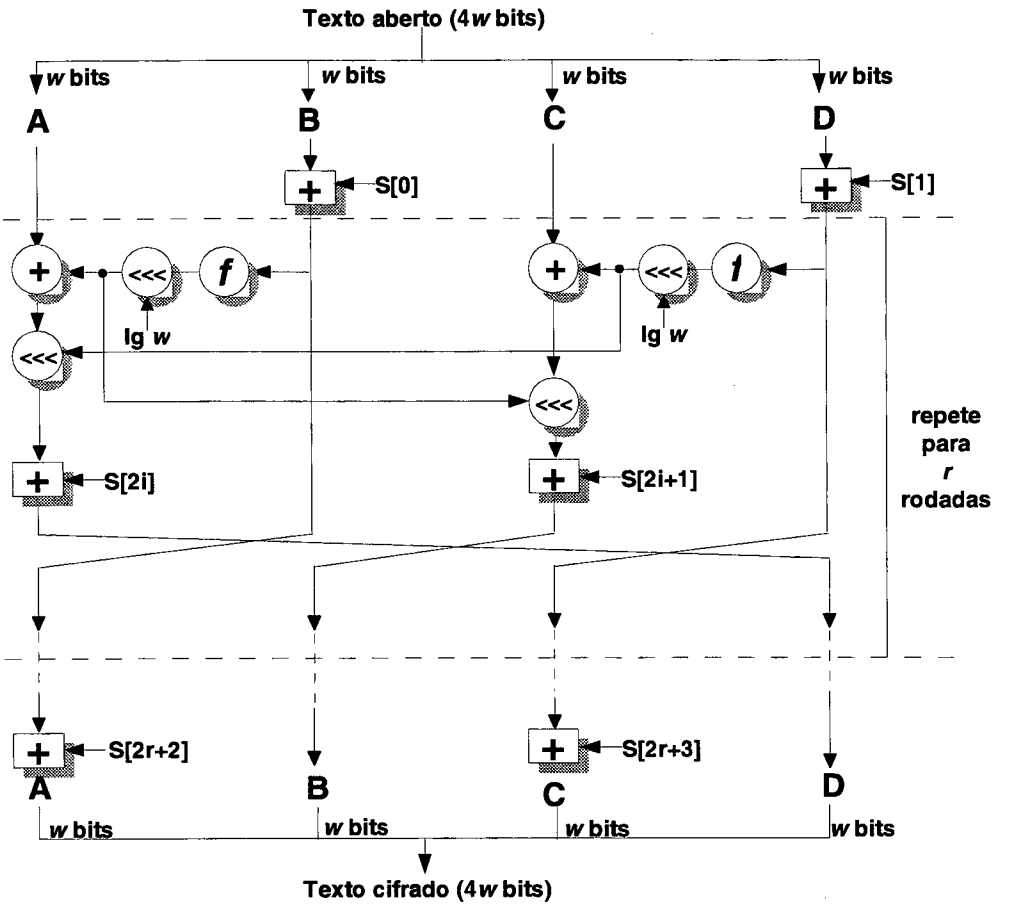


Figura 4.6: Esquema geral do RC6. Aqui, $f(x) = x \times (2x + 1)$

Tabela 4.3: Parâmetros do RC6

Parâmetro	Definição	Valores Permitidos
w	Comprimento da palavra em *bits* (O bloco possui sempre $4w$ *bits*).	16, 32, 64
r	Número de rodadas.	0, 1, ..., 255
b	Número de *bytes*. na chave secreta K.	0, 1, ..., 255

Desta forma, o RC6 cifra blocos de texto aberto de comprimento 64, 128 ou 256 *bits* em blocos de texto cifrado do mesmo comprimento. O comprimento da chave varia de 0 até 2048 *bits*.

4.5.2 Operações primitivas do RC6

As operações primitivas utilizadas no cifrador RC6 são descritas na tabela 4.4 [RIV 98b]. As operações de adição módulo 2^w, subtração módulo 2^w, OU Exclusivo e o deslocamento circular (rotação) dependente dos dados já eram utilizadas no RC5. Foi acrescentada a rotação em *lg(w) bits*, que significa a quantidade representada pelo logaritmo na base 2 de *w*. Foram também incluídas as operações de multiplicação módulo 2^w e a troca da ordem dos registradores.

Tabela 4.4: Operações primitivas utilizadas no RC6

Operação	Descrição
A + B	Adição de registradores módulo 2^w
A - B	Subtração de registradores módulo 2^w
A $\oplus$ B	OU Exclusivo entre dois registradores
A $\lll$ B	Rotação dos *bits* do registrador A à esquerda em *lg(w) bits* de B
A $\ggg$ B	Rotação dos *bits* do registrador A à direita em *lg(w) bits* de B
(A,B,C,D) = (B,C,D,A)	Troca da ordem dos registradores
A x B	Multiplicação de registradores módulo 2^w

4.5.3 Expansão da chave do RC6

O algoritmo de geração das sub-chaves do RC6 é praticamente idêntico ao do RC5[4], exceto que mais sub-chaves são geradas a partir da chave fornecida durante o processo de cifração e decifração. O usuário fornece uma chave *K* de *b bytes* que será

[4]Conforme pode ser visto na seção 4.4.3

expandida para $2r + 4$ sub-chaves de *w bits*: S[0], ..., S[2*r*+3]. Inicialmente, a chave é carregada numa matriz de c palavras de *w bits*, onde $c = w/b$, L[0], ..., L[c-1]. As constantes P_w e Q_w são as mesmas utilizadas pelo RC5 e aparecem na tabela 4.2. Estas constantes são arbitrárias e poderiam ser trocadas em versões adaptadas ou proprietárias do RC6.

A figura 4.7 apresenta o código para o processo de geração das sub-chaves do RC6. Considerando como exemplo os requisitos do AES, de $r = 20$ e $w = 32$, o RC6-32/20 utiliza 44 sub-chaves, numeradas S0 até S43, cada uma com 32 *bits* de comprimento[5].

```
Entrada: r, w, b, chave K
Saída: 2r + 4 sub-chaves de w bits armazenadas em S[0], ..., S[2r+3]
Processo:
S[0] = Pw
for i = 1 to 2r + 3 do
   S[i] = S[i-1] + Qw
A = B = i = j = 0
v = 3 x max{c,2r + 4}
for s = 1 to v do {
   A = S[i] = (S[i] + A + B) <<< 3
   B = L[j] = (L[j] + A + B) <<< (A + B)
   i = (i + 1) mod (2r + 4)
   j = (j + 1) mod c
}
```

Figura 4.7: Algoritmo de expansão da chave do RC6

[5]as valores de b que são relevantes para o AES são 16, 24 e 32

4.5.4 Algoritmo de cifração do RC6

Os quatro registradores A, B, C e D conterão a palavra a ser cifrada ou decifrada, sendo os *bytes* armazenados no formato *little-endian* [6].

O esquema geral do RC6 já foi apresentado na figura 4.6. O algoritmo de cifração do RC6 genérico está descrito na figura 4.8 [RIV 98b]. O RC6 inicia com um OU Exclusivo entre B e S[0] e entre D e S[1]. Após esta etapa, cada rodada utiliza duas sub-chaves; a primeira utiliza S[2] e S[3], a cada rodada seguinte utiliza as sub-chaves sucessivas. Em cada uma das r rodadas as mesmas operações serão repetidas: rotação, soma, multiplicação de *bits* e operação de OU Exclusivo, finalizadas pela troca de ordem dos registradores, já com seus valores pós-calculados. Após as r rodadas, uma etapa adicional é executada: é feito um OU Exclusivo entre A e S[$2r$ + 2] e entre C e S[$2r$ + 3], que é a última sub-chave gerada.

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do {
   t = (B x (2B + 1)) ⋘ lg(w)
   u = (D x (2D + 1)) ⋘ lg(w)
   A = ((A ⊕ t) ⋘ u) + S[2i]
   C = ((C ⊕ u) ⋘ t) + S[2i+1]
   (A,B,C,D) = (B,C,D,A)
}
A = A + S[2r+2]
C = C + S[2r+3]
```

Figura 4.8: Algoritmo da cifração do RC6

[6] conforme explicado na seção 4.4.4

4.5.5 Algoritmo de decifração do RC6

A figura 4.9 apresenta o algoritmo de decifração do RC6 [RIV 98b]. O algoritmo de decifração é derivado do algoritmo de cifração e as sub-chaves utilizadas são as mesmas, porém utilizadas na ordem inversa.

```
C = C - S[2r + 3]
A = A - S[2r + 2]
for i = r downto 1 do {
   (A,B,C,D) = (D,A,B,C)
   u = (D x (2D + 1)) ⋘ lg(w)
   t = (B x (2B + 1)) ⋘ lg(w)
   C = (C - S[2i+1]) ⋙ t) ⊕ u
   A = (A - S[2i]) ⋙ u)  ⊕ t
}
D = D - S[1]
B = B - S[0]
```

Figura 4.9: Algoritmo da decifração do RC6

4.5.6 O RC6 utilizando os requisitos do AES

Na figura 4.10 é apresentado o algoritmo do RC6 utilizando os requisitos para o AES [RIV 98a]. Nesta figura, a parte hachurada indica as modificações neste algoritmo em relação ao RC6 genérico, para atender aos requisitos do AES. Para atender a estes requisitos utiliza-se w=32, r=20 e b=16, 24 ou 32. Neste caso, o RC6 trabalha com blocos de texto aberto e texto cifrado de 128 *bits*, divididos em 4 registradores A, B, C e D de 32 *bits* cada, chave de 128, 192 ou 256 *bits* e gera 44 sub-chaves também de 32 *bits*.

```
B = B + S[0]
D = D + S[1]
for i = 1 to 20 do {
   t = (B x (2B + 1)) ⋘ 5
   u = (D x (2D + 1)) ⋘ 5
   A = ((A ⊕ t) ⋘ u) + S[2i]
   C = ((C ⊕ u) ⋘ t) + S[2i+1]
   (A,B,C,D) = (B,C,D,A)
}
A = A + S[ 42 ]
C = C + S[ 43 ]
```

Figura 4.10: Algoritmo da cifração do RC6 utilizando os requisitos do AES

A figura 4.11 apresenta o algoritmo de decifração do cifrador RC6 utilizando os requisitos do AES [RIV 98a].

```
C = C - S[ 43 ]
A = A - S[ 42 ]
for i = 20 downto 1 do {
   (A,B,C,D) = (D,A,B,C)
   u = (D x (2D + 1)) ⋘ 5
   t = (B x (2B + 1)) ⋘ 5
   C = (C - S[2i+1]) ⋙ t) ⊕ u
   A = (A - S[2i]) ⋙ u) ⊕ t
}
D = D - S[1]
B = B - S[0]
```

Figura 4.11: Algoritmo de decifração do RC6 utilizando os requisitos do AES

4.5.7 Do RC5 ao RC6

Em [CON 99], os criadores do RC6 enfatizam a importância da simplicidade no desenvolvimento de um cifrador. Uma complexidade desnecessária tornaria mais difícil o exame da verdadeira segurança oferecida. Da mesma forma, a excepcional simplicidade do RC5 convida à avaliação de sua segurança. Esta tradição continua com o RC6, cujo projeto encoraja os pesquisadores e facilita o entendimento do cifrador. O projeto do RC6 poderia ser visto de forma progressiva a partir do RC5, através dos passos descritos no apêndice D [RIV 98a].

É importante ressaltar a informação que aparece no apêndice D de que a decisão de expandir para quatro registradores de 32 *bits* foi feita em primeiro lugar, por questões de desempenho, e que, uma vez que o suporte em C para aritmética de 64 *bits* melhore, uma variante do RC6 possuindo 2 registradores de 64 *bits* pode vir a ser adotada.

4.5.8 O desempenho do RC6

Em [RIV 98b] são apresentadas algumas medidas de tempos de cifração, decifração e agendamento de chaves do RC6. Serão apresentados aqui alguns destes tempos, relativos à implementação em C e em plataforma de 32 *bits*. Para maiores detalhes sobre o desempenho do RC6 e sobre seu desempenho em Java, *assembly*, e também seu desempenho em plataformas de 8 *bits*, consulte [RIV 98b].

Os valores mostrados aqui para uma implementação otimizada em ANSI C do RC6 foram obtidas utilizando o compilador Borland C++ Development Suite 5.0 conforme especificado no requisitos de submissão do AES. O desempenho foi medido utilizando um Pentium II 266 MHz com 32 MB de RAM executando o Windows 95. Cada conjunto de testes foi executado 10 vezes, sendo relatada na tabela 4.5 a média obtida.

Tabela 4.5: Desempenho da cifração e decifração do RC6 em ANSI C

Esquema	Ciclos/bloco	Blocos/segundo a 200 MHz	Mbytes/segundo a 200 MHz
cifração(chave de qualquer comprimento)	616	325.000	5,19
decifração(chave de qualquer comprimento)	566	353.000	5,65
cifração do RC6-32/16/16 (para comparação)	328	610.000	4,9

A figura da tabela 4.5 não inclui o agendamento das chaves, e é independente do comprimento da chave fornecida pelo usuário. Os tempos em ANSI C foram obtidos pela cifração e decifração de um único pedaço de dado com 3.000 blocos em modo ECB.

O tempo necessário para o agendamento das chaves com rc6-32/20/*b* quando utilizando chaves de comprimento $b = 128$, $b = 192$ e $b = 256$ *bits* é mostrado na tabela 4.6.

Tabela 4.6: Desempenho do agendamento de chaves do RC6 em ANSI C

Esquema	Ciclos	μsegundos a 200 MHz	agendamentos de chave/segundo a 200 MHz
RC6-32/20/16	2.350	11.8	84.800
RC6-32/20/24	2.350	11.8	84.800
RC6-32/20/32	2.360	11.8	84.800

4.5.9 A segurança do RC6

Os acréscimos feitos no RC6 em relação ao RC5, como a introdução de uma função quadrática e a rotação à esquerda de 5 *bits*, executam um importante papel em complicar as tentativas de ataques contra o cifrador, segundo [RIV 98b]. Os valores da transformação quadrática de B e D são utilizados no lugar de B e D para modificar os registradores A e C, aumentando a não linearidade do esquema, sem perder nenhuma funcionalidade (visto que a transformação é uma permutação). A rotação de 5 *bits* tem o papel de complicar a criptoanálise diferencial e linear.

Em análise feita ao RC6 em [RIV 98a], realizando ataques utilizando

força bruta, criptoanálises diferencial e linear, chegou-se as conclusões descritas na tabela 4.7, com relação ao esforço necessário para se quebrar o cifrador.

Tabela 4.7: Ataques ao RC6

Tipo de ataque	r=8	r=12	r=16	r=20	r=24
Criptoanálise diferencial	2^{56}	2^{117}	2^{190}	2^{238}	2^{299}
Criptoanálise linear	2^{47}	2^{83}	2^{119}	2^{155}	2^{191}

Em relação ao método de força bruta, o esforço necessário para uma busca on-line de uma chave de *b bytes* ou para a matriz de expansão da chave S[0,...,43] é de no mínimo $\{2^{8b}, 2^{1408}\}$ operações. Esta operação necessitaria de uma memória considerável (mais de 2^{700} *bytes*) e por volta de 2^{704} pré-cálculos off-line para uma tentativa de ataque *meet-in-the-middle* para recuperar a matriz de chave expandida [CON 98].

A melhor forma de ataque ao RC6 seria a procura exaustiva pela chave de cifração fornecida pelo usuário; os recursos necessários aos ataques mais sofisticados tanto pelo método de criptoanálise diferencial como linear excedem aos recursos hoje disponíveis; não existem exemplos conhecidos de chaves *fracas* [RIV 98a].

Desta forma, atualmente, o método de força bruta, e até mesmo a criptoanálise diferencial e linear parecem impraticáveis, o que torna o RC6 seguro contra estes tipos de ataques, considerando as exigências do AES de um cifrador de 20 rodadas.

4.6 Comparação entre os cifradores de Ron Rivest

A tabela 4.8 compara os cifradores de Ron Rivest, analisando o ano de criação, o tipo (cifrador de bloco ou de fluxo), o comprimento do bloco, o comprimento da chave e o número de rodadas. Pode-se perceber que algumas características são encontradas em todos estes cifradores, como a parametrização do comprimento de bloco e chave e do número de rodadas. A rotação de *bits* é também uma característica marcante em todos eles, assim como as rotinas longas de agendamento de chave, utilizando sempre dois vetores e valores pré-definidos para a inicialização do vetor de sub-chaves. Percebe-

se também um grande salto entre o RC2 e o RC5, com a eliminação do uso de caixas S, confiando cada vez mais na operação de rotação de *bits* para fornecer a não-linearidade. O RC6 é claramente uma evolução do RC5, que se aproveitou dos anos de estudos sobre o RC5, para criar um novo cifrador que atende aos requisitos do AES.

Tabela 4.8: Tabela comparativa dos parâmetros dos cifradores de Ron Rivest

	RC2	RC4	RC5	RC6
Ano de criação	1989	1987	1994	1998
Tipo de cifrador	Bloco	Fluxo	Bloco	Bloco
Comprimento do bloco (em *bits*)	64	-	32, 64, 128	64, 128, 256
Comprimento da chave (em *bits*)	56	1...2048	0...2048	0...2048
Número de rodadas	18	-	0..255	0..255

4.7 Conclusão

Neste capítulo, foram apresentados os cifradores de Ron Rivest mostrando a evolução destes algoritmos até o advento do RC6. Estes cifradores desempenham um papel de grande relevância no que concerne ao uso atual da criptografia nos sistemas comerciais. O RC2 é utilizado pelo S/MIME, enquanto que o RC4 é utilizado por servidores de Colaboração Pessoal, de Banco de Dados por protocolos de comunicação. O RC5 é utilizado nos principais produtos da RSA Data Security Inc., como os produtos BSAFE, JSAFE e S/MAIL. O RC6 foi um dos 5 finalistas ao Padrão de Criptografia Avançada do governo americano, tendo sido arduamente analisado durante os últimos 3 anos por pesquisadores do mundo inteiro. As conclusões a respeito deste esforço mundial podem ser encontradas em [NEC 00].

Uma das características que fez os algoritmos RC2, RC4 e RC5 serem amplamente utilizados foi a possibilidade de trabalhar com tamanho de chave variável. Esta possibilidade fez com que o governo americano impusesse restrições à exportação de produtos que continham estes algoritmos, limitando o tamanho da chave a 40 *bits*, enquanto que o mesmo produto, quando em uso dentro dos Estados Unidos, podia manter

níveis de segurança mais altos, por exemplo com chaves de 128 *bits*[SCH 96b].

Consenso geral entre os estudiosos de criptografia é que a força de um algoritmo não reside no segredo de sua forma de trabalho. Muito mais importante é a simplicidade do algoritmo, utilizando poucas linhas de código e fazendo uso de primitivas simples. O fato do RC4 ter sido inicialmente um algoritmo de código não-divulgado mostra bem esta verdade, pois ele foi facilmente descoberto e divulgado no mundo inteiro através da Internet. O RC5, por outro lado, teve seu código e suas características amplamente divulgados, e os estudiosos foram incentivados a analisar seu funcionamento a procura de possíveis vulnerabilidades. O RC6 segue a mesma tendência, tendo os autores, deixado disponíveis todo o tipo de informações a respeito do seu funcionamento, além de implementações para diversas plataformas[LAB 01, oS 01]. A escolha do AES também foi um exemplo desta tendência, pois vimos uma seleção aberta, onde toda a comunidade pode opinar, sugerir, testar e analisar os algoritmos e propor mudanças e melhorias. O relatório final, [FOR 00], é claro e detalhado, com mais de 100 páginas de informações, comparativos, tabelas e gráficos, apagando a imagem negativa que o NIST deixou quando da seleção do DES, na década de 70.

O RC6 não foi o escolhido para ser o AES, embora o RC6 tenha sido considerado o mais rápido dos finalistas em plataformas de 32 *bits*. Ele apresentou uma grande desvantagem, pois necessita de uma grande quantidade de memória, não sendo indicado para a implementação em dispositivos com restrição de memória principal. Apesar de não ter vencido a corrida pelo AES, o RC6 está, atualmente, sendo submetido a três outros projetos que definirão padrões de criptografia para os próximos anos. Maiores detalhes são descritos na introdução deste trabalho.

O fato de não se utilizar das tradicionais caixas S, mas confiar a não-linearidade à operação de rotação de *bits* dependente dos dados, também é uma característica marcante nesta família de cifradores. A partir desta experiência, pode-se ver uma série de novos cifradores que não se baseiam na estrutura clássica de Feistel, mas procuram outras alternativas.

Por todos estes motivos entende-se que estes cifradores são um marco na história dos cifradores simétricos e que qualquer estudo da criptografia precisa passar

pela compreensão desta família de cifradores e do que eles representam no contexto atual.

Capítulo 5

Modelos Simplificados do RC6

5.1 Introdução

Este capítulo apresenta as variantes simplificadas do cifrador RC6 disponíveis na literatura. Serão verificadas as características destas variantes, com o intuito de verificar se elas atendem o objetivo proposto neste trabalho, ou se contribuem no desenvolvimento do modelo simplificado aqui proposto.

Se, a partir do RC6, forem retirados a rotação fixa por lg *w bits* e a função quadrática $f(x) = x(2x+1)$, isto é, substituí-la pela função identidade $f(x) = x$, então o cifrador resultante seria muito similar a um RC5 com 4 blocos sendo usados ao mesmo tempo em cada rodada. Este é o ponto de partida para a análise, e a medida que se introduz complexidades adicionais, pode-se ir analisando variantes cada vez mais próximas ao RC6. Estas variantes simplificadas do RC6 foram apresentadas em [CON 98] e também são alvo de estudo em outras publicações, como [CON 99] e [SHI 98]. As variantes simplificadas conhecidas do cifrador RC6 são: RC6-I-NFR, RC6-I e RC6-NFR. A seguir cada uma destas variantes do RC6 será apresentada em mais detalhes.

5.1.1 RC6-I-NFR

Esta é uma versão do RC6 em que a função $f(x) = x(2x+1)$ é substituída pela função identidade $f(x) = x$ e não existem rotações fixas. A figura 5.1 apre-

seria o esquema de cifração do algoritmo RC6-I-NFR [1]. A figura 5.2 ilustra o algoritmo de cifração [CON 98], destacando as alterações em relação ao RC6 nas linhas hachuradas. O RC6-I-NFR é uma versão muito simplificada do cifrador RC6 (figura 4.6). Em sua estrutura, ele pode ser comparado a duas versões paralelas do RC5 (figura 4.3). A quantidade de rotações para uma das partes é tomada da outra parte e vice-versa. Observando o RC6-I-NFR é possível ter uma noção de como é a estrutura interna do RC6 e também identificar seus componentes criptográficos (adição, OU Exclusivo, rotação e multiplicação) que fornecem as maiores contribuições a sua segurança, tanto isolados quanto combinados.

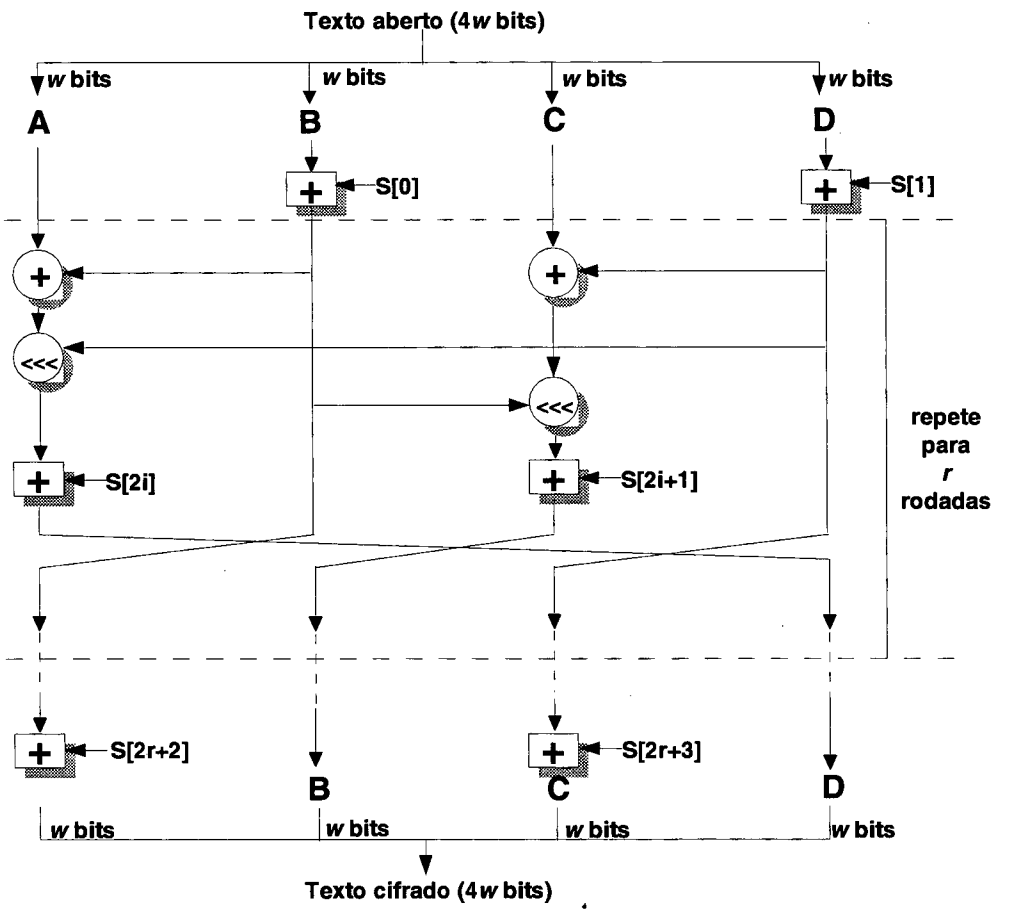


Figura 5.1: Esquema geral do RC6-I-NFR

[1] "I" de função **I**dentidade e "NFR" de **N**o **F**ixed **R**otation (sem rotação fixa).

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do {
    t = B
    u = D
    A = ((A ⊕ t) ⋘ u) + S[2i]
    C = ((C ⊕ u) ⋘ t) + S[2i+1]
    (A,B,C,D) = (B,C,D,A)
}
A = A + S[2r+2]
C = C + S[2r+3]
```

Figura 5.2: Algoritmo de cifração do RC6-I-NFR

5.1.2 RC6-NFR

O RC6-NFR diferencia-se do RC6-I-NFR por manter a função quadrática. Nesta versão, as rotações fixas foram retiradas. A figura 5.3 apresenta o esquema de cifração do RC6-NFR. A figura 5.4 ilustra o algoritmo de cifração do RC6-NFR [CON 98]. Nesta figura, as linhas hachuradas destacam as alterações em relação ao RC6.

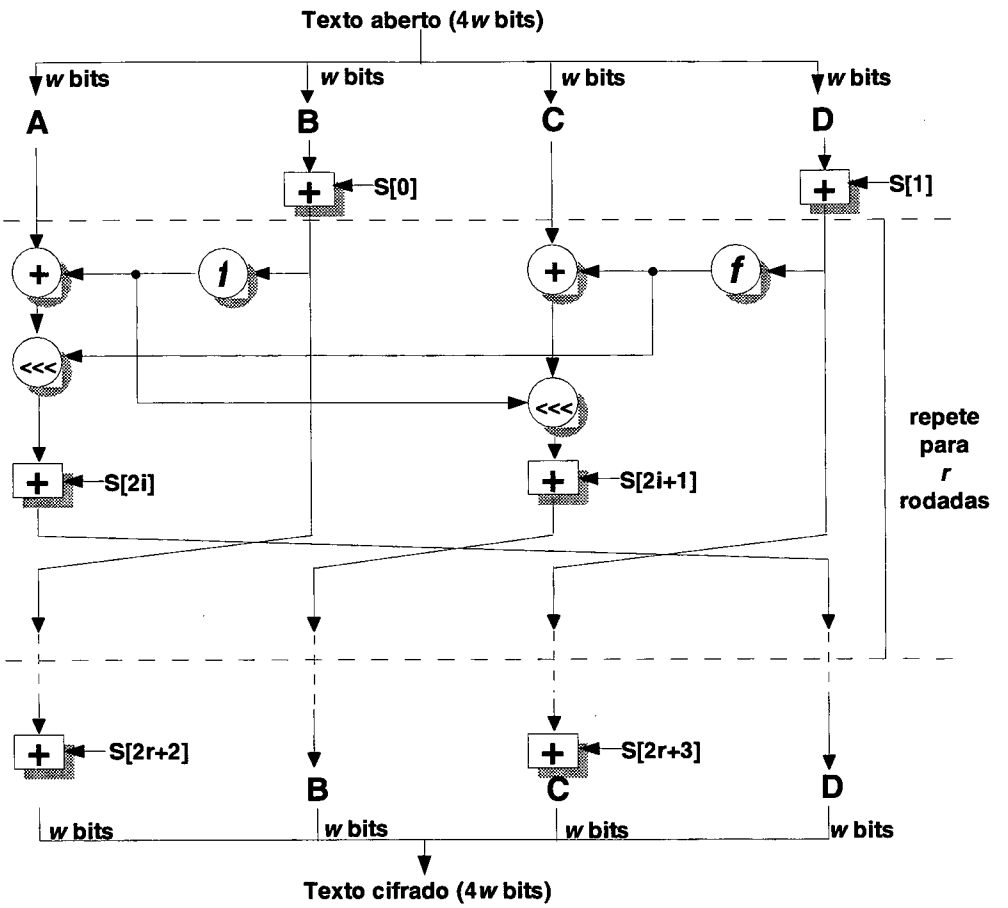


Figura 5.3: Esquema geral do RC6-NFR

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do {
    t = B x (2B + 1)
    u = D x (2D + 1)
    A = ((A ⊕ t) ⋘ u) + S[2i]
    C = ((C ⊕ u) ⋘ t) + S[2i+1]
    (A,B,C,D) = (B,C,D,A)
}
A = A + S[2r+2]
C = C + S[2r+3]
```

Figura 5.4: Algoritmo de cifração do RC6-NFR

5.1.3 RC6-I

Esta é uma versão do RC6 em que a função $f(x) = x(2x+1)$ quadrática é substituída pela função identidade $f(x) = x$. A principal diferença entre as variantes RC6-I e RC6-I-NFR é a rotação fixa por cinco *bits*. Desta forma, pode-se observar como as rotação fixas afetam a evolução das características e diferenças no RC6-I. A figura 5.5 apresenta o esquema de cifração do RC6-I. A figura 5.6 ilustra o algoritmo de cifração do RC6-I [CON 98]. Nesta figura as linhas hachuradas destacam as alterações em relação ao RC6.

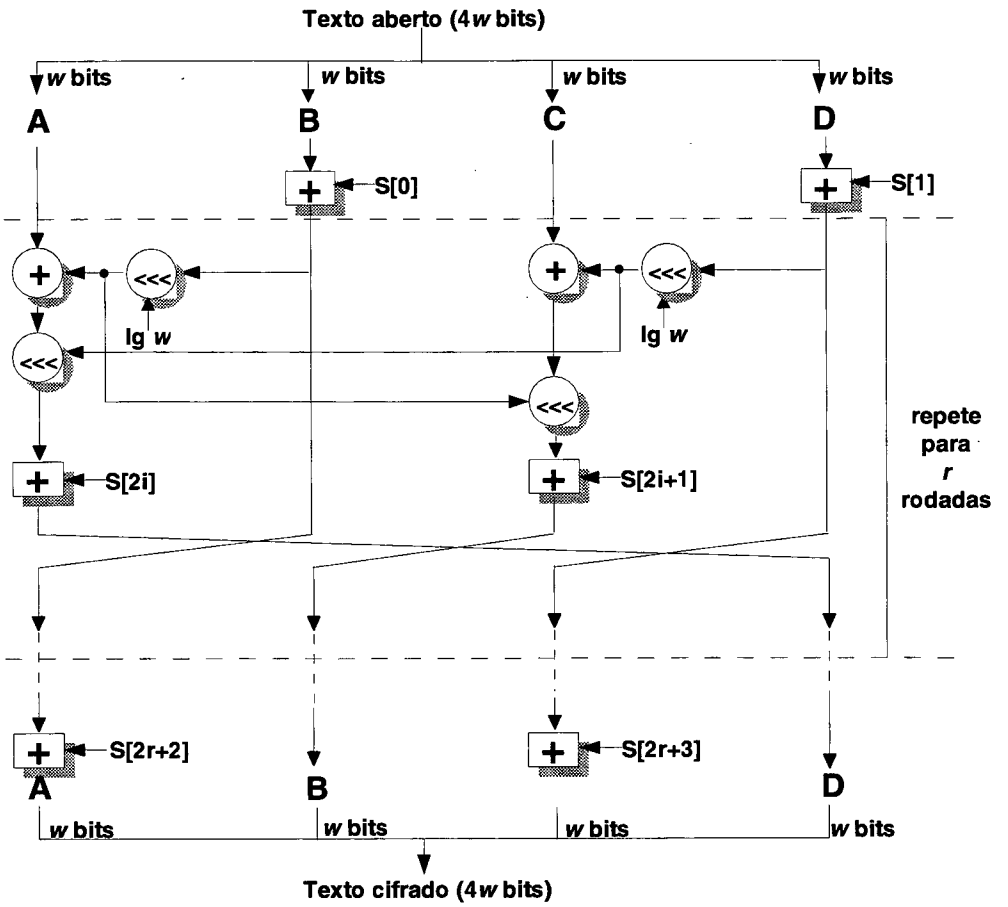


Figura 5.5: Esquema geral do RC6-I

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do {
   t = B ⋘ lg w
   u = D ⋘ lg w
   A = ((A ⊕ t) ⋘ u) + S[2i]
   C = ((C ⊕ u) ⋘ t) + S[2i+1]
   (A,B,C,D) = (B,C,D,A)
}
A = A + S[2r+2]
C = C + S[2r+3]
```

Figura 5.6: Algoritmo de cifração do RC6-I

5.2 Relação entre o cifrador RC6 e as variantes simplificadas

A figura 5.7 mostra o RC6 e as suas variantes numa certa hierarquia, mostrando o próprio RC6 como o algoritmo mais sofisticado entre eles.

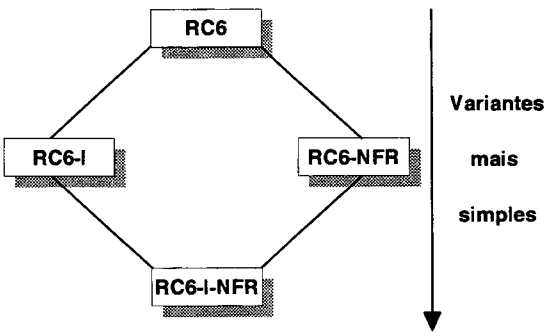


Figura 5.7: Comparação entre as variantes simplificadas do RC6

A diferença entre o cifrador RC6 e as variantes se dá na forma como os valores de t e u são atribuídos. Estas diferenças são resumidas na tabela 5.1. Os valores

de t e u são utilizados para a transformação dos registradores A e C.

Tabela 5.1: Diferenças entre o RC6 e suas variantes simplificadas

	RC6-I-NFR	RC6-I	RC6-NFR	RC6
t=	B	B $\lll$ lg w	B x (2B + 1)	(B x (2B + 1) $\lll$ lg w)
u=	D	D $\lll$ lg w	D x (2D + 1)	(D x (2D + 1) $\lll$ lg w)

De acordo com [CON 98], o objetivo do estudo destas variantes é o de observar o quão seguras as variantes do RC6 vão se tornando na medida em que elas se tornam mais parecidas com ele. A análise destas variantes é importante porque ela aponta para formas de desenvolver ataques ao cifrador RC6 e também destaca como as diferenças o tornam mais efetivo contra estes ataques.

Em [CON 99], o papel das rotações fixas no RC6 é considerado e é demonstrado que efeitos vistos no RC5 e nas variantes simplificadas do RC6 não parecem existir no próprio RC6.

Em [SHI 98], é comentado sobre o objetivo do desenvolvimento das variantes simplificadas aqui apresentadas. Estas variantes simplificadas não são cifradores de bloco reais, mas protótipos de cifradores de blocos utilizados para a comparação da segurança com o cifrador RC6.

5.3 Conclusão

As variantes simplificadas do RC6 apresentadas neste capítulo foram propostas com o objetivo de verificar o grau de segurança que determinadas operações trazem ao RC6, bem como para verificar sua condição diante de determinados ataques.

Estas variantes não atendem ao objetivo proposto neste trabalho, que é o de facilitar o estudo pela redução dos parâmetros, sem a retirada das características do cifrador, necessárias para seu real entendimento. Também não permitem o uso no processo de cifração e decifração manual, uma vez que o número de rodadas e o comprimento da chave e dos blocos é o mesmo do cifrador RC6 original.

Capítulo 6

Proposta de um Modelo Simplificado para o RC6

6.1 Introdução

Neste capítulo, é detalhada a proposta do modelo simplificado para o cifrador de bloco simétrico RC6. Este capítulo contém a descrição dos objetivos do trabalho, as estratégias que foram utilizadas no desenvolvimento do modelo simplificado, os modelos desenvolvidos até se chegar ao modelo final, e um exemplo de uso do modelo simplificado.

6.2 Objetivos

Como foi apresentado, o algoritmo RC6, embora parametrizável, é complexo e seu processo de funcionamento é de difícil entendimento. Isto poderia ser resolvido com a utilização de um modelo simplificado, semelhante ao que se fez com o DES, e seu modelo simplificado o S-DES. Propõe-se o desenvolvimento de um modelo simplificado para o cifrador RC6, a ser chamado S-RC6 (*Simplified RC6*[1]) que:

1. Utilize as mesmas operações primitivas que o RC6, mantendo todas as principais

[1]RC6 Simplificado

características originais[2];

2. Torne natural o entendimento do RC6, uma vez que se definam as diferenças entre ambos;

3. Seja facilmente utilizado tanto para a cifração quanto decifração de forma manual, ou através de um programa simples;

4. Possa ser utilizado em ambiente acadêmico, para o entendimento de conceitos de criptografia como cifradores de bloco, cifradores simétricos, expansão de sub-chaves, funções de não-linearidade e outros;

5. Forneça ao estudioso do algoritmo um bom entendimento do funcionamento e das características do RC6;

6. Facilite a apresentação didática do RC6;

7. Facilite a análise do cifrador;

8. Sirva como referência para a proposição de outros modelos simplificados de cifradores simétricos.

As variantes simplificadas propostas na literatura não atendem aos objetivos propostos neste trabalho, conforme discutido no capítulo 5. Também não se alcança um modelo simplificado ideal simplesmente escolhendo os valores mínimos para w, r e b.

6.3 Estratégia Utilizada

Ao longo do desenvolvimento da proposta de dissertação (trabalho individual), foram definidas as seguintes estratégias, através das quais tinha-se como objetivo chegar a um modelo simplificado adequado:

[2]As operações primitivas do RC6 estão descritas na seção 4.5.2.

1. Executar um número mínimo de rodadas, possivelmente apenas uma ou duas; isto reduziria também o comprimento da matriz de expansão da chave S, já que seu comprimento é de $2r + 4$. No caso de 1 rodada, existiriam 6 elementos e de 2 rodadas, 8 elementos.

2. Reduzir o comprimento dos registradores ao mínimo possível, reduzindo assim o comprimento do bloco. Um comprimento ideal seria 2 para o registrador, trabalhando então com blocos de 8 *bits*;

3. Voltar a trabalhar como no RC5, com 2 registradores em cada rodada. Isto simplificaria o entendimento e a complexidade do algoritmo. O RC6 de 64 *bits* possivelmente trabalhará assim [RIV 98a]; a vantagem seria a redução do comprimento do algoritmo, da matriz de expansão da chave e do número de operações em cada rodada, já que ele é parecido com duas rodadas do RC5 juntas;

4. Trabalhar com uma chave de comprimento reduzido, possivelmente 8 ou 16 *bits*;

Na conclusão deste capítulo, estas estratégias serão novamente analisadas, verificando se elas se mostraram acertadas e em que elas contribuíram para o desenvolvimento do modelo.

6.4 Desenvolvimento do Modelo S-RC6

Nesta seção, mostra-se a estratégia utilizada para o desenvolvimento do modelo simplificado desejado. São apresentados e discutidos vários modelos, até se chegar ao modelo final do S-RC6. Para esta seção será adotada a mesma notação utilizada para descrever versões do RC6, ou seja, RC6-w/r/b, onde a palavra possui *w bits*, executando *r* rodadas e utilizando chaves de *b bytes*. Vale observar que o RC6 trabalha apenas com comprimentos de bloco iguais a 16, 32 ou 64, enquanto que nestes modelo serão adotados outros valores. Para maiores informações sobre os parâmetros do RC6 e os valores utilizados, consulte a seção 4.5.1.

6.4.1 RC6-1/0/1

Partindo das estratégias definidas na seção 6.3, definiu-se uma versão do RC6 com as seguintes características: $w = 1$ *bit*, $r = 0$, e $b = 1$ *byte*, que será chamado de RC6-1/0/1. O esquema geral para esta versão está ilustrado na figura 6.1. Esta versão possui bloco de texto aberto e texto cifrado de 4 *bits* de comprimento, chave de 8 *bits* de comprimento e nenhuma rodada. No entanto, sem pelo menos uma rodada, perde-se muitas das operações primitivas do RC6. Apenas a adição módulo 2^w é executada, excetuando-se operações primitivas importantes, como a função quadrática e a rotação de *bits* dependente dos dados. Na seção 4.5.2, as primitivas do RC6 são explicadas em mais detalhes.

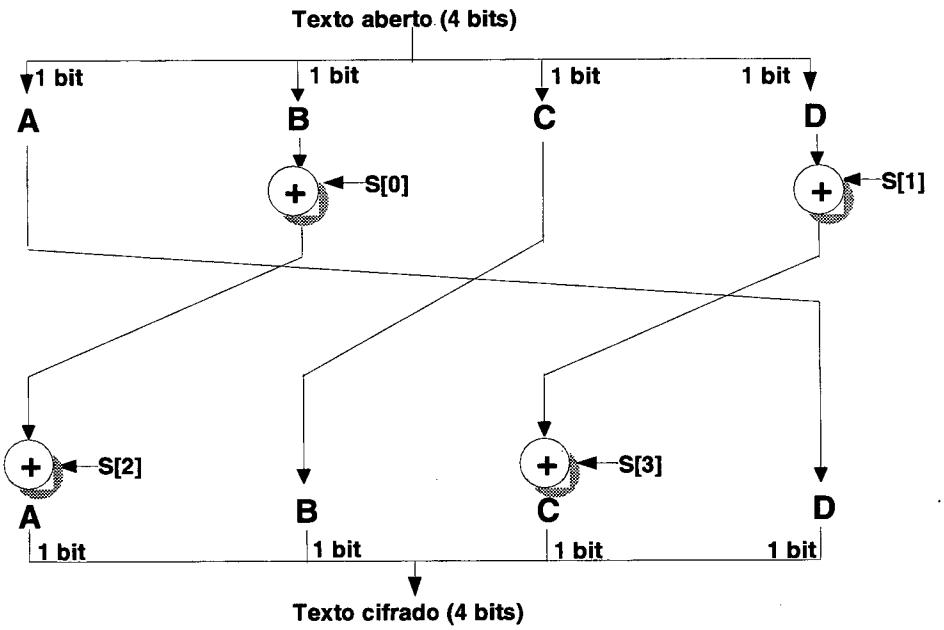


Figura 6.1: Esquema geral do RC6-1/0/1

6.4.2 RC6-1/1/1

Outra versão do RC6 é então criada, denominada RC6-1/1/1. Esta versão é semelhante à anterior, porém executa uma rodada na cifração dos dados, isto é, $r = 1$. As tabelas-verdade das operações primitivas de adição, OU Exclusivo e função

quadrática são as ilustradas nas tabelas 6.1, 6.2, e 6.3, respectivamente. Como a rotação dos *bits* é determinada pelo logaritmo na base 2 de w e neste caso $w = 1$, tem-se $lg1 = 0$, pois $2^0 = 1$. Desta forma, o RC6-1/1/1 não possui a função de rotação de *bits* dependente dos dados. Vale lembrar que todas as operações são executadas módulo 2^w.

Tabela 6.1: Operação de OU Exclusivo para $w = 1$

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 6.2: Operação de adição para $w = 1$

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 6.3: Função quadrática para $w = 1$

x	$f(x) = x(2x + 1)$
0	0
1	1

Pode-se perceber que para o RC6-1/1/1, as seguintes características são:

- A rotação dependente dos dados em $lg\ w$ não acontece, pois $lg1 = 0$;
- A função de OU Exclusivo possui a mesma tabela verdade da função de adição;
- A função quadrática passou a ser substituída pela função identidade $f(x) = x$, ou seja, ela não tem efeito neste caso.

A versão final do RC6-1/1/1 pode ser vista na figura 6.2.

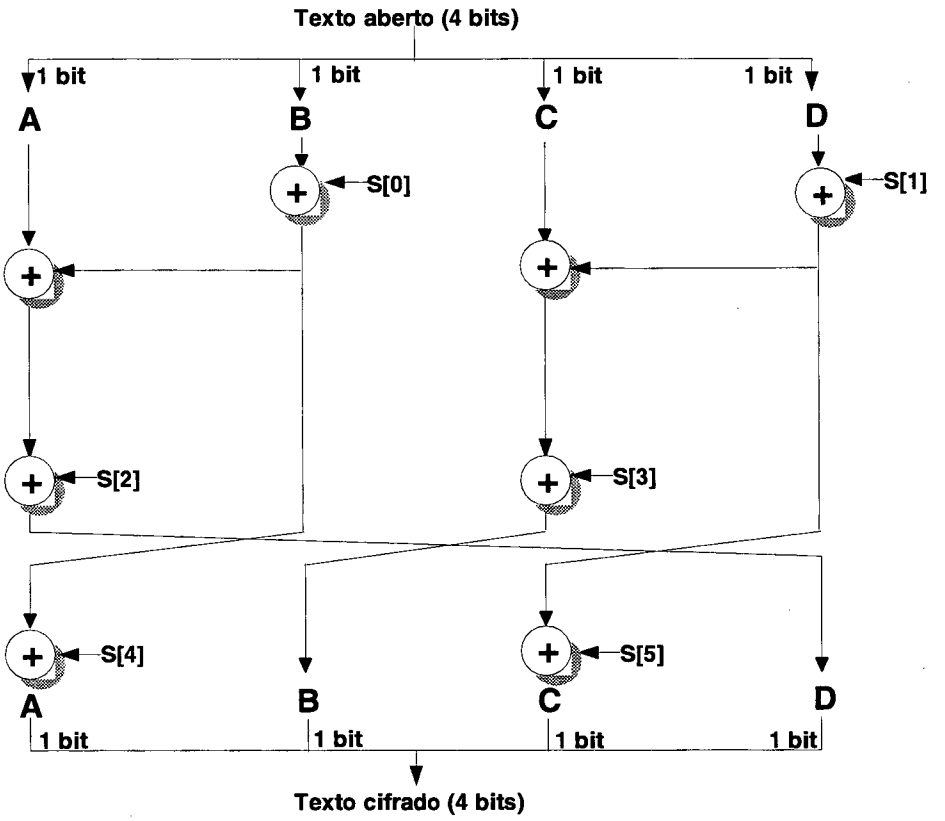


Figura 6.2: Esquema geral do RC6-1/1/1

Uma vez que operações primitivas importantes do RC6 acabam sendo anuladas nesta versão, a conclusão é que ela não serve como um modelo simplificado, pois não cumpre o objetivo de manter as mesmas operações primitivas do RC6.

6.4.3 RC6-2/1/1

A próxima versão a ser verificada é a RC6-2/1/1, onde $w = 2$. Nesta versão, os blocos de texto aberto e texto cifrado possuem 8 *bits* de comprimento, a chave também possui 8 *bits* e a cifração executa uma rodada. São geradas 6 sub-chaves, cada uma com 2 *bits* de comprimento. As tabelas-verdade das operações primitivas de adição, OU Exclusivo e função quadrática são as ilustradas nas tabelas 6.4, 6.5, e 6.6, respectivamente. Nesta versão, a rotação será de 1 *bit*. Todas as operações são executadas módulo

2^2.

Tabela 6.4: Operação de OU Exclusivo para $w = 2$

x	y	$x \oplus y$
0	0	0
0	1	1
0	2	2
0	3	3
1	1	0
1	2	3
1	3	2
2	2	0
2	3	1
3	3	0

Tabela 6.5: Operação de adição para $w = 2$

x	y	$x + y$
0	0	0
0	1	1
0	2	2
0	3	3
1	1	2
1	2	3
1	3	0
2	2	0
2	3	1
3	3	2

Tabela 6.6: Função quadrática para $w = 2$

x	$f(x) = x(2x+1)$	$\lll 1$
0	0	0
1	3	3
2	2	1
3	1	2

O RC6-2/1/1 pode ser visto na figura 6.3. Nesta figura, pode-se verificar que no RC6-2/1/1 todas as operações primitivas estão presentes. Conseqüentemente, o algoritmo não fica descaracterizado. Desta forma, o comprimento de w de 2 *bits* será adotado para o desenvolvimento do modelo simplificado.

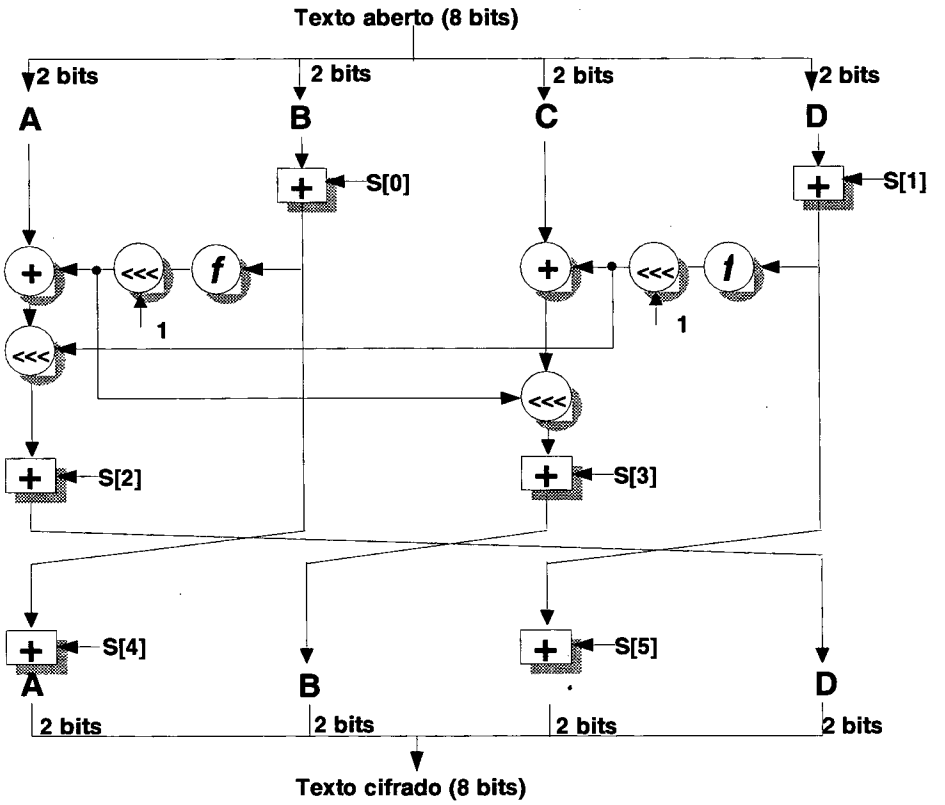


Figura 6.3: Esquema geral do RC6-2/1/1

6.4.4 S-RC6

Analisando o algoritmo da cifração da versão RC6-2/1/1, ilustrada na figura 6.3, pode-se verificar que o texto aberto é dividido em 4 registradores de *w bits*. O uso de 4 registradores foi feito por questões de desempenho, e uma versão com apenas 2 registradores não perderia em termos de segurança[3]. Como esta abordagem torna o algoritmo mais simples, sem no entanto tirar características importantes do algoritmo, será adotado o uso de apenas dois registradores, chamados A e B.

Quando se utiliza apenas dois registradores, volta-se ao mesmo tipo de rodada simples utilizada pelo RC5, onde apenas um registrador é atualizado em cada rodada. Como serão utilizados 2 registradores, A e B, serão executadas duas rodadas, para que os dois registradores sejam atualizados.

Fazendo esta modificações, chega-se à versão ilustrada na figura 6.4, que será chamada a partir de agora de S-RC6[4], que será analisada em mais detalhes a seguir. O S-RC6 possui $w = 2$ *bits*, $r = 2$, e $b = 1$ *byte*. Neste modelo, o bloco de texto aberto e texto cifrado é de 4 *bits* de comprimento ($2w$ *bits*), dividido em dois registradores A e B. A chave possui 8 *bits* e a cifração executa duas rodadas. São geradas 4 sub-chaves, cada uma com 2 *bits* de comprimento. Todas as operações são executadas módulo 2^2. A tabela 6.7 ilustra a tabela verdade da rotação de *bits* para $w = 2$, sendo útil para o processo de cifração manual do S-RC6.

[3]Conforme discutido na seção 4.5.7.

[4](*Simplified* RC6 ou RC6 Simplificado)

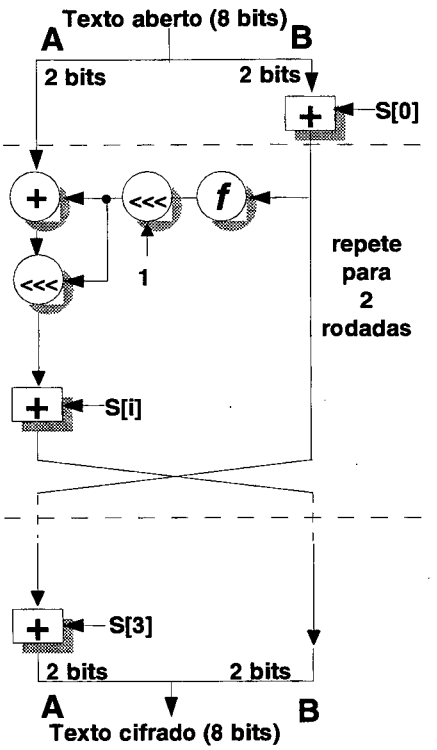


Figura 6.4: Esquema geral do S-RC6

Tabela 6.7: Operação de rotação de *bits* para $w = 2$

x		$y = x \bmod 2$		$z = y \lll 1$	
0	0000	0	00	0	00
1	0001	1	01	2	10
2	0010	2	10	1	01
3	0011	3	11	3	11
4	0100	0	00	0	00
5	0101	1	01	2	10
6	0110	2	10	1	01
7	0111	3	11	3	11
8	1000	0	00	0	00
9	1001	1	01	2	10

Para $w = 2$, embora durante a execução do algoritmo apareçam rotações

de várias quantidades diferentes de *bits*, todas elas podem ser reduzidas a apenas dois tipos: rotação de 1 *bit* ou nenhuma rotação. A tabela 6.8 ilustra estas rotações. Nesta tabela é utilizado um valor arbitrário para x, e verifica-se que valores se obtém para $x \lll y$. Pode-se perceber que se o número de rotações é par, então não há rotação; se o número de rotações é ímpar, então há uma rotação de 1 *bit*. Também pode-se verificar que para este modelo, em particular, a rotação à esquerda é idêntica a rotação à direita. O resultado é o mesmo para qualquer valor de x. O conhecimento desta característica será útil durante a utilização do S-RC6 para a cifração e decifração manual.

Tabela 6.8: Exemplo de rotação de *bits* para $w = 2$

x	y	$x \lll y$	$x \ggg y$
01	0	01	01
01	1	10	10
01	2	01	01
01	3	10	10

6.4.5 Expansão da chave utilizando o S-RC6

O algoritmo de expansão da chave utilizará um vetor S de 4 elementos $(r + 2)$ de 2 *bits* (w) cada um, inicializado com as constantes P_2 e Q_2, e um vetor L de 4 elementos de 2 *bits* cada, contendo a chave. Utilizando as constantes definidas na tabela 4.2, tem-se $P_2 = 10_2 = 2$ e $Q_2 = 10_2 = 2$. O algoritmo de mistura dos vetores S e L para a geração das sub-chaves do S-RC6 está ilustrado na figura 6.5.

6.4.6 Cifração utilizando o S-RC6

A figura 6.6 apresenta o algoritmo de cifração do S-RC6. A rotação em $lg\ w$ *bits* fica sendo nesta versão de 1 *bit*. Os dois registradores A e B recebem cada um 2 *bits* do texto aberto e depois passam por duas rodadas do cifrador, produzindo o texto cifrado.

```
A = B = i = j = 0
for j = 1 to 12 do {
   S[i] = (S[i] + A + B) ⋘ 1
   A = S[i]
   L[i] = (L[i] + A + B) ⋘ (A + B)
   B = L[i]
   i = (i + 1) mod 2
}
```

Figura 6.5: Algoritmo de expansão da chave do S-RC6

```
B = B + S[0]
for i = 1 to 2 do {
   t = (B x (2B + 1)) ⋘ 1
   A = ((A ⊕ t) ⋘ t) + S[i]
   (A,B) = (B,A)
}
A = A + S[3]
```

Figura 6.6: Algoritmo de cifração do S-RC6

6.4.7 Decifração utilizando o S-RC6

A figura 6.7 apresenta o algoritmo de decifração do S-RC6. Os dois registradores A e B recebem o texto cifrado e depois passam por duas rodadas do cifrador, produzindo o texto aberto.

```
A = A - S[3]
for i = 2 downto 1 do {
   (A,B) = (B,A)
   t = (B x (2B + 1)) ⋘ 1
   A = (A - S[i]) ⋘ t) ⊕ t
}
B = B - S[0]
```

Figura 6.7: Esquema de decifração do S-RC6

6.5 Exemplo de cifração utilizando o S-RC6

Da mesma forma que foi feito para o algoritmo S-DES[5], pode-se utilizar um exemplo de cifração utilizando o S-RC6, para demonstrar seu funcionamento. Para isto serão utilizados os valores de texto aberto e chave conforme a tabela 6.9.

Tabela 6.9: Texto aberto e chave para a cifração com o S-RC6

Texto aberto	=	10 10
Chave	=	10 10 10 10

A chave é copiada para o vetor L e o vetor S é inicializado com as constantes P_2 e Q_2, conforme descrito na seção 4.4.3. Os valores dos vetores S e L, após inicializados, são os definidos na tabela 6.10.

Tabela 6.10: Valores iniciais dos vetores S e L para o S-RC6 em binário

S	=	10 00 10 00
L	=	10 10 10 10

A execução do algoritmo de expansão da chave do S-RC6, ilustrado na figura 6.5, produz os valores definidos na tabela 6.11. Os valores hachurados são as quatro sub-chaves que serão utilizadas nos processos de cifração e decifração.

[5]Consulte a seção 3.2.6

Tabela 6.11: Valores da mistura dos vetores S e L para o S-RC6

j	i	S0	S1	S2	S3	L0	L1	L2	L3	A	B
0	0	2	0	2	0	2	2	2	2	0	0
1	1	1	0	2	0	3	2	2	2	1	3
2	2	1	0	2	0	3	2	2	2	0	2
3	3	1	0	0	0	3	2	0	2	0	0
4	0	1	0	0	0	3	2	0	2	0	2
5	1	3	0	0	0	0	2	0	2	3	0
6	2	3	3	0	0	0	2	0	2	3	2
7	3	3	3	2	0	0	2	0	2	2	0
8	0	3	3	2	1	0	2	0	3	1	3
9	1	3	3	2	1	2	2	0	3	3	2
10	2	3	0	2	1	2	0	0	3	0	0
11	3	3	0	1	1	2	0	2	3	1	2
12	0	3	0	1	0	2	0	2	1	0	1

A execução do algoritmo de cifração, ilustrado na figura 6.6, produz os valores definidos na tabela 6.12. Os valores hachurados são os registradores A e B, que compõem o texto cifrado resultante.

Tabela 6.12: Valores do processo de cifração do S-RC6

i	A	B	t
0	2	2	0
0	2	1	0
1	2	1	3
1	1	2	3
2	1	2	1
2	2	1	1
3	2	1	1

6.5.1 Modos de operação do S-RC6

Os modos de operação mais comuns para a utilização em cifradores de bloco simétricos foram explicados na seção 3.3.5. Para o S-RC6, serão propostos dois modos de operação, a título de exemplo:

ECB *("Electronic Codebook"):* Neste modo de operação cada bloco de 4 *bits* de texto aberto é cifrado individualmente, utilizando a mesma chave 8 *bits*. Para uma mensagem com um comprimento maior que 4 *bits*, o procedimento consiste em quebrar a mensagem em blocos de 4 *bits*, completando o último bloco com zeros, se necessário. A decifração é feita um bloco de cada vez, utilizando a mesma chave. O modo ECB é ideal para pequenas quantidades de dados, como uma chave de cifração, por ser simples de implementar e rápido na execução. A principal característica do ECB é que se um mesmo bloco de 64 *bits* de texto aberto aparece mais de uma vez em uma mensagem, ele sempre produzirá o mesmo texto cifrado. Para textos longos, o modo ECB pode não ser seguro. Se a mensagem é bem estruturada, é possível explorar estas regularidades. Um exemplo deste tipo de mensagens são arquivos de imagem *bitmap* ou de texto, que possuem uma estrutura bem conhecida e grandes quantidades de dados redundantes, como espaços em branco.

CBC *("Cipher Block Chaining"):* O modo CBC resolve as deficiências de segurança do ECB, fazendo com que a cifração de cada bloco dependa dos blocos anteriores e o mesmo texto aberto pode produzir textos cifrados diferentes dependendo de seu contexto na mensagem completa. Ele consegue isto executando um OU Exclusivo em cada bloco de 4 *bits* de texto aberto com o bloco de texto cifrado anterior antes de ser cifrado. Para a cifração do primeiro bloco de texto aberto, é feito um OU Exclusivo entre um vetor de inicialização (IV) de 4 *bits* e o primeiro bloco de texto aberto. O vetor de inicialização precisa ser fornecido também na decifração, para ser ter novamente o texto aberto original.

Com estes dois modos de operação, o funcionamento do S-RC6 pode ser verificado em várias situações, cifrando textos de diferentes tamanhos e com diferentes tipos de chaves, verificando como o modo influencia na confusão dos *bits* do texto cifrado.

6.5.2 Segurança do S-RC6

Um ataque de força bruta ao S-RC6 é perfeitamente possível, uma vez que com uma chave de 8 *bits*, existem somente $2^8 = 256$ possíveis chaves. Dado um texto cifrado, um atacante poderia tentar cada uma das possibilidades e analisar o resultado para determinar se o texto aberto faz sentido.

6.6 Conclusão

Neste capítulo foi definido o modelo simplificado do cifrador de bloco simétrico RC6, denominado S-RC6. O desenvolvimento deste modelo partiu do RC6 reduzido a seus parâmetros mínimos. Partindo dos requisitos iniciais de que não fossem perdidas as características do RC6, nem que fossem alterados o funcionamento das primitivas básicas, o modelo foi evoluindo até que se chegou a um modelo adequado, com blocos de 4 *bits*, divididos em dois registradores, chave de 8 *bits* e executando duas rodadas.

Capítulo 7

Considerações Finais

Este trabalho apresentou, como proposta de dissertação, um modelo simplificado para o cifrador RC6, a ser denominado S-RC6, e que poderá vir a ser utilizado na apresentação didática do RC6, bem como no estudo dos conceitos referentes à criptografia, principalmente àqueles relacionados aos cifradores de bloco simétricos.

Inicialmente, foi analisado o projeto de cifradores de bloco simétricos e as características dos modernos cifradores de bloco. O algoritmo DES foi logo a seguir apresentado, assim como seu modelo simplificado, o S-DES, para que pudesse ser observado como o estudo de um modelo simplificado pode auxiliar no estudo de um cifrador. Este é o maior objetivo deste trabalho. Também foram estudados os cifradores de Ron Rivest, ressaltando a importância dos mesmos na criptografia moderna e verificando a evolução ocorrida nestes cifradores no decorrer do tempo. Após o estudo destes cifradores, passou-se ao estudo de algumas das variantes simplificadas conhecidas do cifrador RC6, à procura de alguma que atendesse ao requisitos propostos neste trabalho. Foi verificado que as variantes simplificadas do RC6 encontradas na literatura não atendem aos objetivos aqui propostos. Enquanto o S-RC6 busca a apresentação para o ensino do cifrador, as variantes simplificadas estudadas têm por objetivo unicamente o estudo da segurança advinda de operações que foram incluídas no RC6 e que não estavam presentes no RC5.

7.1 Discussão dos resultados

Analisando os objetivos traçados, observa-se que os mesmos foram atingidos, pois o S-RC6:

1. Utiliza as mesmas operações primitivas que o RC6, isto é, adição, multiplicação, OU Exclusivo, rotação de bits, troca de registradores;

2. Torna natural o entendimento do RC6, uma vez que ele é essencialmente uma versão reduzida do RC6, tendo apenas algumas pequenas modificações (por exemplo, o uso de 2 registradores);

3. Pode ser facilmente utilizado tanto para a cifração quanto decifração de forma manual, ou através de um programa bastante simples. O uso manual, incluindo a expansão da chave, a cifração e a decifração não ocupam mais do que duas páginas e consomem pouco tempo. Um programa pode ser facilmente desenvolvido e executado passo-a-passo através de um depurador;

4. Pode ser utilizado em ambiente acadêmico, como por exemplo cursos de criptografia ou mesmo numa disciplina de graduação ou pós-graduação. O algoritmo é simples de entender e utilizar, embora tenha todas as características principais do RC6;

5. Fornece ao estudante do algoritmo um bom entendimento do funcionamento e das características do RC6, pois ele utiliza as mesmas primitivas e os algoritmos de expansão de chaves, cifração e decifração são muito semelhantes aos do RC6.

6. Facilita a apresentação didática do RC6. É possível apresentar inicialmente o modelo simplificado e, após o entendimento deste, apresentar o cifrador RC6. Uma abordagem semelhante foi utilizada neste trabalho para apresentar o DES, apresentando inicialmente o S-DES;

7. Facilita a análise do cifrador. Rotinas de busca exaustiva de chaves, ou outras técnicas de criptoanálise tornam-se mais simples num cifrador com parâmetros reduzidos;

8. Serve como referência para a proposição de outros modelos simplificados de cifradores simétricos. Partindo das estratégias utilizadas, pode-se desenvolver modelos simplificados para outros cifradores simétricos, alcançando objetivos semelhantes aos obtidos neste trabalho.

Agora, serão analisadas cada uma das estratégias definidas inicialmente:

1. O S-RC6 executa apenas duas rodadas. Este é o número mínimo de rodadas para que os registradores A e B sejam atualizados;

2. O S-RC6 utiliza registradores com 2 *bits* de comprimento, que é o comprimento mínimo para que as primitivas básicas sejam utilizadas;

3. O S-RC6 utiliza apenas 2 registradores, como o RC5;

4. O SRC-6 faz uso do comprimento mínimo da chave, que é de 8 *bits*.

Revendo as estratégias definidas inicialmente, verifica-se que elas foram seguidas e que realmente se mostraram válidas. Pequenos ajustes foram realizados, mas a idéia principal foi mantida.

O S-RC6 é um modelo simplificado de um cifrador moderno. Ele é simples, fácil de implementar e de entender, e permite o estudo de características presentes nos principais cifradores modernos, como a rotação dependente dos dados e as funções de não-linearidades que não se baseiam em caixas S. Desta forma, afirma-se novamente a importância deste modelo simplificado para o estudo e entendimento dos princípios existentes no projeto de cifradores simétricos.

Sendo assim, foi desenvolvida a proposta de um modelo simplificado para o cifrador RC6, que atendeu aos objetivos inicialmente propostos e que fez uso das estratégias previamente definidas. Foi desenvolvida uma versão implementada em linguagem de programação C, que se mostrou bastante didática e facilita ainda mais o entendimento do modelo.

7.2 Trabalhos futuros

A verificação da eficácia deste modelo junto a estudantes de criptografia, medindo o nível de entendimento conseguido por eles através do uso do modelo simplificado e de suas implementações mostrar-se-ia de muita valia para a avaliação do modelo para fins didáticos e poderia ser uma proposta de trabalho futuro. Poderiam ser desenvolvidos também trabalhos visando verificar a segurança deste modelo, bem como possíveis formas de ataque utilizando criptoanálise linear e diferencial e outros. Um trabalho sobre a sistematização do ataque ao S-RC6 seria interessante. O desenvolvimento de trabalhos comparando o desempenho do RC5, RC6 e S-RC6 também é uma área para possíveis trabalhos futuros, pois nos daria mais parâmetros de comparação entre os cifradores reais e seus modelos simplificados.

Referências Bibliográficas

[AND 98] ANDERSON, R.; BIHAM, E.; KNUDSEN, L. **Serpent: A Flexible Block Cipher with Maximum Assurance**. NIST AES Proposal.

[BAL 96] BALDWIN, R.; RIVEST, R. L. **The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms**. Request For Comments 2040.

[BUR 99] BURWICK, C. et al. **The MARS Encryption Algorithm**. NIST AES Proposal.

[CON 98] CONTINI, S. et al. **The Security of the RC6 Block Cipher**. Disponível em [LAB 01].

[CON 99] CONTINI, S. et al. Improved analysis of some variants of rc6. 1999. **Proceedings...** Germany: Springer-Verlag, 1999. v.1636, p.1–16.

[DAE 98] DAEMEN, J.; RIJMEN, V. **The Rijndael Block Cipher - AES Proposal**. NIST AES Proposal.

[FEI 73] FEISTEL, H. **Cryptography and Computer Privacy**.

[FOR 00] FORD, S. **Advanced Encryption Standard (AES) Development Effort**. Disponível em [oS 01].

[KAL 95] KALINSKI, B. S.; YIN, Y. L. **On the Security of the RC5 Encryption Algorithm**. Relatório Técnico TR-602 dos Laboratórios RSA disponível em [oS 01].

[KNU 98] KNUDSEN, L. R. et al. On the design and security of rc2. **Lecture Notes In Computer Science**, [S.l.], v.1372, p.206–221, 1998.

[LAB 88] LABORATORIES, R. **Frequently Asked Questions about Today's Cryptography**. RSA Laboratories, 1988. Disponível em [LAB 01].

[LAB 01] LABORATORIES, R. **RSA Laboratories home page**. `http://www.rsasecurity.com/rsalabs/index.html`.

[MAS 88] MASSEY, J. L. An introduction to contemporary cryptography. 1988. **Proceedings...** Zurich: [s.n.], 1988. v.76, p.533–548.

[MEN 96] MENEZES, A.; VAN OORSCHOT, P.; VANSTONE, S. **Handbook of Applied Cryptography**. California: CRC Press, 1996.

[NEC 00] NECHVATAL, J. et al. **Report on the Development of the Advanced Encryption Standard (AES)**. Disponível em [oS 01].

[oS 01] OF STANDARDS, N. I.; TECHNOLOGY. Advanced encryption standard home page. `http://www.nist.gov/aes`, 04, 2001.

[RIV 94] RIVEST, R. L. The rc5 encryption algorithm. **Lecture Notes In Computer Science**, [S.l.], v.1008, p.86–96, 12, 1994.

[RIV 98a] RIVEST, R. L. **The RC6 Block Cipher: A simple fast secure AES proposal (Presentation)**. Disponível em [LAB 01].

[RIV 98b] RIVEST, R. L. et al. **The RC6 Block Cipher**.

[RSA 00] RSA Laboratories. **RSA's RC6 Algorithm Named As Finalist For New Federal Crypto Standard**, 2000. Disponível em [LAB 01].

[SCH 96a] SCHAEFER, E. F. **A simplified Encryption Standard algorithm**.

[SCH 96b] SCHNEIER, B. **Applied Cryptography**. 2. ed. New York, NY: John Wiley and Sons, Inc., 1996.

[SCH 98] SCHNEIER, B. et al. **Twofish: A 128-Bit Block Cipher**. NIST AES Proposal.

[SCH 00] SCHNEIER, B. **The Data Encryption Standard (DES)**. CRYPTO-GRAM email newsletter. Disponível em `http://www.counterpane.com/crypto-gram.html`.

[SHI 98] SHIMOYAMA, T.; TAKEUCHI, K.; HAYAKAWA, J. **Correlation Attack to the Block Cipher RC5 and the Simplified Variants of RC6**.

[STA 99] STALLINGS, W. **Cryptography and Network Security, Principles and Practice**. 2. ed. Upper Saddle River, NJ: Prentice Hall, 1999.

[STI 95] STINSON, D. R. **Cryptography: Theory and Practice**. New York: CRC Press, 1995.

[TAN 96] TANENBAUM, A. S. **Computer Networks**, chapter7: The Application Layer, p.577–766. Prentice Hall, New Jersey, 3. ed., 1996.

[YIN 97] YIN, Y. L. **The RC5 Encryption Algorithm: Two Years On**.

Apêndice A

Implementação do cifrador DES em Linguagem C

Este anexo consiste num exemplo de implementação do cifrador DES em linguagem de programação C.

des.c

```
//  IBM PC Implementation of the DES Cryptographic Algorithm by
//  Dr B. R. Gladman (gladman@seven77.demon.co.uk)
//
//  Some of the techniques in this DES source code are derived
//  from ideas developed by Richard Outerbridge and Eric Young.
//  I gratefully acknowledge their contribution.
//
//  DES Timings:
//  Key Setup:     1024 cycles
//  Encrypt:      456.4 cycles =    28.05 mbits/sec
//  Decrypt:      453.1 cycles =    28.25 mbits/sec
//  Mean:         454.8 cycles =    28.15 mbits/sec
#define BIG_TABLES
#include "des.h"
#ifdef  BIG_TABLES
#  define   TAB_SIZE    256
#else
#  define   TAB_SIZE     64
#endif
unsigned long sx_tab[8][TAB_SIZE] =
{
  { 0x00820200L, 0x00020000L, 0x80800000L, 0x80820200L,
    0x00800000L, 0x80020200L, 0x80020000L, 0x80800000L,
    0x80020200L, 0x00820200L, 0x00820000L, 0x80000200L,
    0x80800200L, 0x00800000L, 0x00000000L, 0x80020000L,
    0x00020000L, 0x80000000L, 0x00800200L, 0x00020200L,
    0x80820200L, 0x00820000L, 0x80000200L, 0x00800200L,
    0x80000000L, 0x00000200L, 0x00020200L, 0x80820000L,
```

```
    0x00000200L, 0x80800200L, 0x80820000L, 0x00000000L,
    0x00000000L, 0x80820200L, 0x00800200L, 0x80020000L,
    0x00820200L, 0x00020000L, 0x80000200L, 0x00800200L,
    0x80820000L, 0x00000200L, 0x00020200L, 0x80800000L,
    0x80020200L, 0x80000000L, 0x80800000L, 0x00820000L,
    0x80820200L, 0x00020200L, 0x00820000L, 0x80800200L,
    0x00800000L, 0x80000200L, 0x80020000L, 0x00000000L,
    0x00020000L, 0x00800000L, 0x80800200L, 0x00820200L,
    0x80000000L, 0x80820000L, 0x00000200L, 0x80020200L,
#ifdef BIG_TABLES
    0x00820200L, 0x00020000L, 0x80800000L, 0x80820200L,
    0x00800000L, 0x80020200L, 0x80020000L, 0x80800000L,
    0x80020200L, 0x00820200L, 0x00820000L, 0x80000200L,
    0x80800200L, 0x00800000L, 0x00000000L, 0x80020000L,
    0x00020000L, 0x80000000L, 0x00800200L, 0x00020200L,
    0x80820200L, 0x00820000L, 0x80000200L, 0x00800200L,
    0x80000000L, 0x00000200L, 0x00020200L, 0x80820000L,
    0x00000200L, 0x80800200L, 0x80820000L, 0x00000000L,
    0x00000000L, 0x80820200L, 0x00800200L, 0x80020000L,
    0x00820200L, 0x00020000L, 0x80000200L, 0x00800200L,
    0x80820000L, 0x00000200L, 0x00020200L, 0x80800000L,
    0x80020200L, 0x80000000L, 0x80800000L, 0x00820000L,
    0x80820200L, 0x00020200L, 0x00820000L, 0x80800200L,
    0x00800000L, 0x80000200L, 0x80020000L, 0x00000000L,
    0x00020000L, 0x00800000L, 0x80800200L, 0x00820200L,
    0x80000000L, 0x80820000L, 0x00000200L, 0x80020200L,

    0x00820200L, 0x00020000L, 0x80800000L, 0x80820200L,
    0x00800000L, 0x80020200L, 0x80020000L, 0x80800000L,
    0x80020200L, 0x00820200L, 0x00820000L, 0x80000200L,
    0x80800200L, 0x00800000L, 0x00000000L, 0x80020000L,
    0x00020000L, 0x80000000L, 0x00800200L, 0x00020200L,
    0x80820200L, 0x00820000L, 0x80000200L, 0x00800200L,
    0x80000000L, 0x00000200L, 0x00020200L, 0x80820000L,
    0x00000200L, 0x80800200L, 0x80820000L, 0x00000000L,
    0x00000000L, 0x80820200L, 0x00800200L, 0x80020000L,
    0x00820200L, 0x00020000L, 0x80000200L, 0x00800200L,
    0x80820000L, 0x00000200L, 0x00020200L, 0x80800000L,
    0x80020200L, 0x80000000L, 0x80800000L, 0x00820000L,
    0x80820200L, 0x00020200L, 0x00820000L, 0x80800200L,
    0x00800000L, 0x80000200L, 0x80020000L, 0x00000000L,
    0x00020000L, 0x00800000L, 0x80800200L, 0x00820200L,
    0x80000000L, 0x80820000L, 0x00000200L, 0x80020200L,

    0x00820200L, 0x00020000L, 0x80800000L, 0x80820200L,
    0x00800000L, 0x80020200L, 0x80020000L, 0x80800000L,
    0x80020200L, 0x00820200L, 0x00820000L, 0x80000200L,
    0x80800200L, 0x00800000L, 0x00000000L, 0x80020000L,
    0x00020000L, 0x80000000L, 0x00800200L, 0x00020200L,
    0x80820200L, 0x00820000L, 0x80000200L, 0x00800200L,
    0x80000000L, 0x00000200L, 0x00020200L, 0x80820000L,
    0x00000200L, 0x80800200L, 0x80820000L, 0x00000000L,
    0x00000000L, 0x80820200L, 0x00800200L, 0x80020000L,
    0x00820200L, 0x00020000L, 0x80000200L, 0x00800200L,
    0x80820000L, 0x00000200L, 0x00020200L, 0x80800000L,
    0x80020200L, 0x80000000L, 0x80800000L, 0x00820000L,
    0x80820200L, 0x00020200L, 0x00820000L, 0x80800200L,
    0x00800000L, 0x80000200L, 0x80020000L, 0x00000000L,
    0x00020000L, 0x00800000L, 0x80800200L, 0x00820200L,
    0x80000000L, 0x80820000L, 0x00000200L, 0x80020200L,
#endif
  },
  { 0x10042004L, 0x00000000L, 0x00042000L, 0x10040000L,
    0x10000004L, 0x00002004L, 0x10002000L, 0x00042000L,
    0x00002000L, 0x10040004L, 0x00000004L, 0x10002000L,
    0x00040004L, 0x10042000L, 0x10040000L, 0x00000004L,
    0x00040000L, 0x10002004L, 0x10040004L, 0x00002000L,
    0x00042004L, 0x10000000L, 0x00000000L, 0x00040004L,
    0x10002004L, 0x00042004L, 0x10042000L, 0x10000004L,
    0x10000000L, 0x00040000L, 0x00002004L, 0x10042004L,
    0x00040004L, 0x10042000L, 0x10002000L, 0x00042004L,
    0x10042004L, 0x00040004L, 0x10000004L, 0x00000000L,
    0x10000000L, 0x00002004L, 0x00040000L, 0x10040004L,
    0x00002000L, 0x10000000L, 0x00042004L, 0x10002004L,
    0x10042000L, 0x00002000L, 0x00000000L, 0x10000004L,
    0x00000004L, 0x10042004L, 0x00042000L, 0x10040000L,
    0x10040004L, 0x00040000L, 0x00002004L, 0x10002000L,
    0x10002004L, 0x00000004L, 0x10040000L, 0x00042000L,
```

```
#ifdef BIG_TABLES
    0x10042004L, 0x00000000L, 0x00042000L, 0x10040000L,
    0x10000004L, 0x00002004L, 0x10002000L, 0x00042000L,
    0x00002000L, 0x10040004L, 0x00000004L, 0x10002000L,
    0x00040004L, 0x10042000L, 0x10040000L, 0x00000004L,
    0x00040000L, 0x10002004L, 0x10040004L, 0x00002000L,
    0x00042004L, 0x10000000L, 0x00000000L, 0x00040004L,
    0x10002004L, 0x00042004L, 0x10042000L, 0x10000004L,
    0x10000000L, 0x00040000L, 0x00002004L, 0x10042004L,
    0x00040004L, 0x10042000L, 0x10002000L, 0x00042004L,
    0x10042004L, 0x00040004L, 0x10000004L, 0x00000000L,
    0x10000000L, 0x00002004L, 0x00040000L, 0x10040004L,
    0x00002000L, 0x10000000L, 0x00042004L, 0x10002004L,
    0x10042000L, 0x00002000L, 0x00000000L, 0x10000004L,
    0x00000004L, 0x10042004L, 0x00042000L, 0x10040000L,
    0x10040004L, 0x00040000L, 0x00002004L, 0x10002000L,
    0x10002004L, 0x00000004L, 0x10040000L, 0x00042000L,

    0x10042004L, 0x00000000L, 0x00042000L, 0x10040000L,
    0x10000004L, 0x00002004L, 0x10002000L, 0x00042000L,
    0x00002000L, 0x10040004L, 0x00000004L, 0x10002000L,
    0x00040004L, 0x10042000L, 0x10040000L, 0x00000004L,
    0x00040000L, 0x10002004L, 0x10040004L, 0x00002000L,
    0x00042004L, 0x10000000L, 0x00000000L, 0x00040004L,
    0x10002004L, 0x00042004L, 0x10042000L, 0x10000004L,
    0x10000000L, 0x00040000L, 0x00002004L, 0x10042004L,
    0x00040004L, 0x10042000L, 0x10002000L, 0x00042004L,
    0x10042004L, 0x00040004L, 0x10000004L, 0x00000000L,
    0x10000000L, 0x00002004L, 0x00040000L, 0x10040004L,
    0x00002000L, 0x10000000L, 0x00042004L, 0x10002004L,
    0x10042000L, 0x00002000L, 0x00000000L, 0x10000004L,
    0x00000004L, 0x10042004L, 0x00042000L, 0x10040000L,
    0x10040004L, 0x00040000L, 0x00002004L, 0x10002000L,
    0x10002004L, 0x00000004L, 0x10040000L, 0x00042000L,

    0x10042004L, 0x00000000L, 0x00042000L, 0x10040000L,
    0x10000004L, 0x00002004L, 0x10002000L, 0x00042000L,
    0x00002000L, 0x10040004L, 0x00000004L, 0x10002000L,
    0x00040004L, 0x10042000L, 0x10040000L, 0x00000004L,
    0x00040000L, 0x10002004L, 0x10040004L, 0x00002000L,
    0x00042004L, 0x10000000L, 0x00000000L, 0x00040004L,
    0x10002004L, 0x00042004L, 0x10042000L, 0x10000004L,
    0x10000000L, 0x00040000L, 0x00002004L, 0x10042004L,
    0x00040004L, 0x10042000L, 0x10002000L, 0x00042004L,
    0x10042004L, 0x00040004L, 0x10000004L, 0x00000000L,
    0x10000000L, 0x00002004L, 0x00040000L, 0x10040004L,
    0x00002000L, 0x10000000L, 0x00042004L, 0x10002004L,
    0x10042000L, 0x00002000L, 0x00000000L, 0x10000004L,
    0x00000004L, 0x10042004L, 0x00042000L, 0x10040000L,
    0x10040004L, 0x00040000L, 0x00002004L, 0x10002000L,
    0x10002004L, 0x00000004L, 0x10040000L, 0x00042000L,
#endif
  },
  { 0x41000000L, 0x01010040L, 0x00000040L, 0x41000040L,
    0x40010000L, 0x01000000L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00010000L, 0x01010000L, 0x40000000L,
    0x41010040L, 0x40000040L, 0x40000000L, 0x41010000L,
    0x00000000L, 0x40010000L, 0x01010040L, 0x00000040L,
    0x40000040L, 0x41010040L, 0x00010000L, 0x41000000L,
    0x41010000L, 0x01000040L, 0x40010040L, 0x01010000L,
    0x00010040L, 0x00000000L, 0x01000000L, 0x40010040L,
    0x01010040L, 0x00000040L, 0x40000000L, 0x00010000L,
    0x40000040L, 0x40010000L, 0x01010000L, 0x41000040L,
    0x00000000L, 0x01010040L, 0x00010040L, 0x41010000L,
    0x40010000L, 0x01000000L, 0x41010040L, 0x40000000L,
    0x40010040L, 0x41000000L, 0x01000000L, 0x41010040L,
    0x00010000L, 0x01000040L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00000000L, 0x41010000L, 0x40000040L,
    0x41000000L, 0x40010040L, 0x00000040L, 0x01010000L,
#ifdef BIG_TABLES
    0x41000000L, 0x01010040L, 0x00000040L, 0x41000040L,
    0x40010000L, 0x01000000L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00010000L, 0x01010000L, 0x40000000L,
    0x41010040L, 0x40000040L, 0x40000000L, 0x41010000L,
    0x00000000L, 0x40010000L, 0x01010040L, 0x00000040L,
    0x40000040L, 0x41010040L, 0x00010000L, 0x41000000L,
    0x41010000L, 0x01000040L, 0x40010040L, 0x01010000L,
    0x00010040L, 0x00000000L, 0x01000000L, 0x40010040L,
```

```
    0x01010040L, 0x00000040L, 0x40000000L, 0x00010000L,
    0x40000040L, 0x40010000L, 0x01010000L, 0x41000040L,
    0x00000000L, 0x01010040L, 0x00010040L, 0x41010000L,
    0x40010000L, 0x01000000L, 0x41010040L, 0x40000000L,
    0x40010040L, 0x41000000L, 0x01000000L, 0x41010040L,
    0x00010000L, 0x01000040L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00000000L, 0x41010000L, 0x40000040L,
    0x41000000L, 0x40010040L, 0x00000040L, 0x01010000L,

    0x41000000L, 0x01010040L, 0x00000040L, 0x41000040L,
    0x40010000L, 0x01000000L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00010000L, 0x01010000L, 0x40000000L,
    0x41010040L, 0x40000040L, 0x40000000L, 0x41010000L,
    0x00000000L, 0x40010000L, 0x01010040L, 0x00000040L,
    0x40000040L, 0x41010040L, 0x00010000L, 0x41000000L,
    0x41010000L, 0x01000040L, 0x40010040L, 0x01010000L,
    0x00010040L, 0x00000000L, 0x01000000L, 0x40010040L,
    0x01010040L, 0x00000040L, 0x40000000L, 0x00010000L,
    0x40000040L, 0x40010000L, 0x01010000L, 0x41000040L,
    0x00000000L, 0x01010040L, 0x00010040L, 0x41010000L,
    0x40010000L, 0x01000000L, 0x41010040L, 0x40000000L,
    0x40010040L, 0x41000000L, 0x01000000L, 0x41010040L,
    0x00010000L, 0x01000040L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00000000L, 0x41010000L, 0x40000040L,
    0x41000000L, 0x40010040L, 0x00000040L, 0x01010000L,

    0x41000000L, 0x01010040L, 0x00000040L, 0x41000040L,
    0x40010000L, 0x01000000L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00010000L, 0x01010000L, 0x40000000L,
    0x41010040L, 0x40000040L, 0x40000000L, 0x41010000L,
    0x00000000L, 0x40010000L, 0x01010040L, 0x00000040L,
    0x40000040L, 0x41010040L, 0x00010000L, 0x41000000L,
    0x41010000L, 0x01000040L, 0x40010040L, 0x01010000L,
    0x00010040L, 0x00000000L, 0x01000000L, 0x40010040L,
    0x01010040L, 0x00000040L, 0x40000000L, 0x00010000L,
    0x40000040L, 0x40010000L, 0x01010000L, 0x41000040L,
    0x00000000L, 0x01010040L, 0x00010040L, 0x41010000L,
    0x40010000L, 0x01000000L, 0x41010040L, 0x40000000L,
    0x40010040L, 0x41000000L, 0x01000000L, 0x41010040L,
    0x00010000L, 0x01000040L, 0x41000040L, 0x00010040L,
    0x01000040L, 0x00000000L, 0x41010000L, 0x40000040L,
    0x41000000L, 0x40010040L, 0x00000040L, 0x01010000L,
#endif
  },
  { 0x00100402L, 0x04000400L, 0x00000002L, 0x04100402L,
    0x00000000L, 0x04100000L, 0x04000402L, 0x00100002L,
    0x04100400L, 0x04000002L, 0x04000000L, 0x00000402L,
    0x04000002L, 0x00100402L, 0x00100000L, 0x04000000L,
    0x04100002L, 0x00100400L, 0x00000400L, 0x00000002L,
    0x00100400L, 0x04000402L, 0x04100000L, 0x00000400L,
    0x00000402L, 0x00000000L, 0x00100002L, 0x04100400L,
    0x04000400L, 0x04100002L, 0x04100402L, 0x00100000L,
    0x04100002L, 0x00000402L, 0x00100000L, 0x04000002L,
    0x00100400L, 0x04000400L, 0x00000002L, 0x04100000L,
    0x04000402L, 0x00000000L, 0x00000400L, 0x00100002L,
    0x00000000L, 0x04100002L, 0x04100400L, 0x00000400L,
    0x04000000L, 0x04100402L, 0x00100402L, 0x00100000L,
    0x04100402L, 0x00000002L, 0x04000400L, 0x00100402L,
    0x00100002L, 0x00100400L, 0x04100000L, 0x04000402L,
    0x00000402L, 0x04000000L, 0x04000002L, 0x04100400L,
#ifdef BIG_TABLES
    0x00100402L, 0x04000400L, 0x00000002L, 0x04100402L,
    0x00000000L, 0x04100000L, 0x04000402L, 0x00100002L,
    0x04100400L, 0x04000002L, 0x04000000L, 0x00000402L,
    0x04000002L, 0x00100402L, 0x00100000L, 0x04000000L,
    0x04100002L, 0x00100400L, 0x00000400L, 0x00000002L,
    0x00100400L, 0x04000402L, 0x04100000L, 0x00000400L,
    0x00000402L, 0x00000000L, 0x00100002L, 0x04100400L,
    0x04000400L, 0x04100002L, 0x04100402L, 0x00100000L,
    0x04100002L, 0x00000402L, 0x00100000L, 0x04000002L,
    0x00100400L, 0x04000400L, 0x00000002L, 0x04100000L,
    0x04000402L, 0x00000000L, 0x00000400L, 0x00100002L,
    0x00000000L, 0x04100002L, 0x04100400L, 0x00000400L,
    0x04000000L, 0x04100402L, 0x00100402L, 0x00100000L,
    0x04100402L, 0x00000002L, 0x04000400L, 0x00100402L,
    0x00100002L, 0x00100400L, 0x04100000L, 0x04000402L,
    0x00000402L, 0x04000000L, 0x04000002L, 0x04100400L,
```

```
    0x00100402L, 0x04000400L, 0x00000002L, 0x04100402L,
    0x00000000L, 0x04100000L, 0x04000402L, 0x00100002L,
    0x04100400L, 0x04000002L, 0x04000000L, 0x00000402L,
    0x04000002L, 0x00100402L, 0x00100000L, 0x04000000L,
    0x04100002L, 0x00100400L, 0x00000400L, 0x00000002L,
    0x00100400L, 0x04000402L, 0x04100000L, 0x00000400L,
    0x00000402L, 0x00000000L, 0x00100002L, 0x04100400L,
    0x04000400L, 0x04100002L, 0x04100402L, 0x00100000L,
    0x04100002L, 0x00000402L, 0x00100000L, 0x04000002L,
    0x00100400L, 0x04000400L, 0x00000002L, 0x04100000L,
    0x04000402L, 0x00000000L, 0x00000400L, 0x00100002L,
    0x00000000L, 0x04100002L, 0x04100400L, 0x00000400L,
    0x04000000L, 0x04100402L, 0x00100402L, 0x00100000L,
    0x04100402L, 0x00000002L, 0x04000400L, 0x00100402L,
    0x00100002L, 0x00100400L, 0x04100000L, 0x04000402L,
    0x00000402L, 0x04000000L, 0x04000002L, 0x04100400L,

    0x00100402L, 0x04000400L, 0x00000002L, 0x04100402L,
    0x00000000L, 0x04100000L, 0x04000402L, 0x00100002L,
    0x04100400L, 0x04000002L, 0x04000000L, 0x00000402L,
    0x04000002L, 0x00100402L, 0x00100000L, 0x04000000L,
    0x04100002L, 0x00100400L, 0x00000400L, 0x00000002L,
    0x00100400L, 0x04000402L, 0x04100000L, 0x00000400L,
    0x00000402L, 0x00000000L, 0x00100002L, 0x04100400L,
    0x04000400L, 0x04100002L, 0x04100402L, 0x00100000L,
    0x04100002L, 0x00000402L, 0x00100000L, 0x04000002L,
    0x00100400L, 0x04000400L, 0x00000002L, 0x04100000L,
    0x04000402L, 0x00000000L, 0x00000400L, 0x00100002L,
    0x00000000L, 0x04100002L, 0x04100400L, 0x00000400L,
    0x04000000L, 0x04100402L, 0x00100402L, 0x00100000L,
    0x04100402L, 0x00000002L, 0x04000400L, 0x00100402L,
    0x00100002L, 0x00100400L, 0x04100000L, 0x04000402L,
    0x00000402L, 0x04000000L, 0x04000002L, 0x04100400L,
#endif
  },
  { 0x02000000L, 0x00004000L, 0x00000100L, 0x02004108L,
    0x02004008L, 0x02000100L, 0x00004108L, 0x02004000L,
    0x00004000L, 0x00000008L, 0x02000008L, 0x00004100L,
    0x02000108L, 0x02004008L, 0x02004100L, 0x00000000L,
    0x00004100L, 0x02000000L, 0x00004008L, 0x00000108L,
    0x02000100L, 0x00004108L, 0x00000000L, 0x02000008L,
    0x00000008L, 0x02000108L, 0x02004108L, 0x00004008L,
    0x02004000L, 0x00000100L, 0x00000108L, 0x02004100L,
    0x02004100L, 0x02000108L, 0x00004008L, 0x02004000L,
    0x00004000L, 0x00000008L, 0x02000008L, 0x02000100L,
    0x02000000L, 0x00004100L, 0x02004108L, 0x00000000L,
    0x00004108L, 0x02000000L, 0x00000100L, 0x00004008L,
    0x02000108L, 0x00000100L, 0x00000000L, 0x02004108L,
    0x02004008L, 0x02004100L, 0x00000108L, 0x00004000L,
    0x00004100L, 0x02004008L, 0x02000100L, 0x00000108L,
    0x00000008L, 0x00004108L, 0x02004000L, 0x02000008L,
#ifdef BIG_TABLES
    0x02000000L, 0x00004000L, 0x00000100L, 0x02004108L,
    0x02004008L, 0x02000100L, 0x00004108L, 0x02004000L,
    0x00004000L, 0x00000008L, 0x02000008L, 0x00004100L,
    0x02000108L, 0x02004008L, 0x02004100L, 0x00000000L,
    0x00004100L, 0x02000000L, 0x00004008L, 0x00000108L,
    0x02000100L, 0x00004108L, 0x00000000L, 0x02000008L,
    0x00000008L, 0x02000108L, 0x02004108L, 0x00004008L,
    0x02004000L, 0x00000100L, 0x00000108L, 0x02004100L,
    0x02004100L, 0x02000108L, 0x00004008L, 0x02004000L,
    0x00004000L, 0x00000008L, 0x02000008L, 0x02000100L,
    0x02000000L, 0x00004100L, 0x02004108L, 0x00000000L,
    0x00004108L, 0x02000000L, 0x00000100L, 0x00004008L,
    0x02000108L, 0x00000100L, 0x00000000L, 0x02004108L,
    0x02004008L, 0x02004100L, 0x00000108L, 0x00004000L,
    0x00004100L, 0x02004008L, 0x02000100L, 0x00000108L,
    0x00000008L, 0x00004108L, 0x02004000L, 0x02000008L,

    0x02000000L, 0x00004000L, 0x00000100L, 0x02004108L,
    0x02004008L, 0x02000100L, 0x00004108L, 0x02004000L,
    0x00004000L, 0x00000008L, 0x02000008L, 0x00004100L,
    0x02000108L, 0x02004008L, 0x02004100L, 0x00000000L,
    0x00004100L, 0x02000000L, 0x00004008L, 0x00000108L,
    0x02000100L, 0x00004108L, 0x00000000L, 0x02000008L,
    0x00000008L, 0x02000108L, 0x02004108L, 0x00004008L,
    0x02004000L, 0x00000100L, 0x00000108L, 0x02004100L,
    0x02004100L, 0x02000108L, 0x00004008L, 0x02004000L,
```

```
    0x00004000L, 0x00000008L, 0x02000008L, 0x02000100L,
    0x02000000L, 0x00004100L, 0x02004108L, 0x00000000L,
    0x00004108L, 0x02000000L, 0x00000100L, 0x00004008L,
    0x02000108L, 0x00000100L, 0x00000000L, 0x02004108L,
    0x02004008L, 0x02004100L, 0x00000108L, 0x00004000L,
    0x00004100L, 0x02004008L, 0x02000100L, 0x00000108L,
    0x00000008L, 0x00004108L, 0x02004000L, 0x02000008L,

    0x02000000L, 0x00004000L, 0x00000100L, 0x02004108L,
    0x02004008L, 0x02000100L, 0x00004108L, 0x02004000L,
    0x00004000L, 0x00000008L, 0x02000008L, 0x00004100L,
    0x02000108L, 0x02004008L, 0x02004100L, 0x00000000L,
    0x00004100L, 0x02000000L, 0x00004008L, 0x00000108L,
    0x02000100L, 0x00004108L, 0x00000000L, 0x02000008L,
    0x00000008L, 0x02000108L, 0x02004108L, 0x00004008L,
    0x02004000L, 0x00000100L, 0x00000108L, 0x02004100L,
    0x02004100L, 0x02000108L, 0x00004008L, 0x02004000L,
    0x00004000L, 0x00000008L, 0x02000008L, 0x02000100L,
    0x02000000L, 0x00004100L, 0x02004108L, 0x00000000L,
    0x00004108L, 0x02000000L, 0x00000100L, 0x00004008L,
    0x02000108L, 0x00000100L, 0x00000000L, 0x02004108L,
    0x02004008L, 0x02004100L, 0x00000108L, 0x00004000L,
    0x00004100L, 0x02004008L, 0x02000100L, 0x00000108L,
    0x00000008L, 0x00004108L, 0x02004000L, 0x02000008L,
#endif
  },
  { 0x20000010L, 0x00080010L, 0x00000000L, 0x20080800L,
    0x00080010L, 0x00000800L, 0x20000810L, 0x00080000L,
    0x00000810L, 0x20080810L, 0x00080800L, 0x20000000L,
    0x20000800L, 0x20000010L, 0x20080000L, 0x00080810L,
    0x00080000L, 0x20000810L, 0x20080010L, 0x00000000L,
    0x00000800L, 0x00000010L, 0x20080800L, 0x20080010L,
    0x20080810L, 0x20080000L, 0x20000000L, 0x00000810L,
    0x00000010L, 0x00080800L, 0x00080810L, 0x20000800L,
    0x00000810L, 0x20000000L, 0x20000800L, 0x00080810L,
    0x20080800L, 0x00080010L, 0x00000000L, 0x20000800L,
    0x20000000L, 0x00000800L, 0x20080010L, 0x00080000L,
    0x00080010L, 0x20080810L, 0x00080800L, 0x00000010L,
    0x20080810L, 0x00080800L, 0x00080000L, 0x20000810L,
    0x20000010L, 0x20080000L, 0x00080810L, 0x00000000L,
    0x00000800L, 0x20000010L, 0x20000810L, 0x20080800L,
    0x20080000L, 0x00000810L, 0x00000010L, 0x20080010L,
#ifdef BIG_TABLES
    0x20000010L, 0x00080010L, 0x00000000L, 0x20080800L,
    0x00080010L, 0x00000800L, 0x20000810L, 0x00080000L,
    0x00000810L, 0x20080810L, 0x00080800L, 0x20000000L,
    0x20000800L, 0x20000010L, 0x20080000L, 0x00080810L,
    0x00080000L, 0x20000810L, 0x20080010L, 0x00000000L,
    0x00000800L, 0x00000010L, 0x20080800L, 0x20080010L,
    0x20080810L, 0x20080000L, 0x20000000L, 0x00000810L,
    0x00000010L, 0x00080800L, 0x00080810L, 0x20000800L,
    0x00000810L, 0x20000000L, 0x20000800L, 0x00080810L,
    0x20080800L, 0x00080010L, 0x00000000L, 0x20000800L,
    0x20000000L, 0x00000800L, 0x20080010L, 0x00080000L,
    0x00080010L, 0x20080810L, 0x00080800L, 0x00000010L,
    0x20080810L, 0x00080800L, 0x00080000L, 0x20000810L,
    0x20000010L, 0x20080000L, 0x00080810L, 0x00000000L,
    0x00000800L, 0x20000010L, 0x20000810L, 0x20080800L,
    0x20080000L, 0x00000810L, 0x00000010L, 0x20080010L,

    0x20000010L, 0x00080010L, 0x00000000L, 0x20080800L,
    0x00080010L, 0x00000800L, 0x20000810L, 0x00080000L,
    0x00000810L, 0x20080810L, 0x00080800L, 0x20000000L,
    0x20000800L, 0x20000010L, 0x20080000L, 0x00080810L,
    0x00080000L, 0x20000810L, 0x20080010L, 0x00000000L,
    0x00000800L, 0x00000010L, 0x20080800L, 0x20080010L,
    0x20080810L, 0x20080000L, 0x20000000L, 0x00000810L,
    0x00000010L, 0x00080800L, 0x00080810L, 0x20000800L,
    0x00000810L, 0x20000000L, 0x20000800L, 0x00080810L,
    0x20080800L, 0x00080010L, 0x00000000L, 0x20000800L,
    0x20000000L, 0x00000800L, 0x20080010L, 0x00080000L,
    0x00080010L, 0x20080810L, 0x00080800L, 0x00000010L,
    0x20080810L, 0x00080800L, 0x00080000L, 0x20000810L,
    0x20000010L, 0x20080000L, 0x00080810L, 0x00000000L,
    0x00000800L, 0x20000010L, 0x20000810L, 0x20080800L,
    0x20080000L, 0x00000810L, 0x00000010L, 0x20080010L,

    0x20000010L, 0x00080010L, 0x00000000L, 0x20080800L,
```

```
    0x00080010L, 0x00000800L, 0x20000810L, 0x00080000L,
    0x00000810L, 0x20080810L, 0x00080800L, 0x20000000L,
    0x20000800L, 0x20000010L, 0x20080000L, 0x00080810L,
    0x00080000L, 0x20000810L, 0x20080010L, 0x00000000L,
    0x00000800L, 0x00000010L, 0x20080800L, 0x20080010L,
    0x20080810L, 0x20080000L, 0x20000000L, 0x00000810L,
    0x00000010L, 0x00080800L, 0x00080810L, 0x20000800L,
    0x00000810L, 0x20000000L, 0x20000800L, 0x00080810L,
    0x20080800L, 0x00080010L, 0x00000000L, 0x20000800L,
    0x20000000L, 0x00000800L, 0x20080010L, 0x00080000L,
    0x00080010L, 0x20080810L, 0x00080800L, 0x00000010L,
    0x20080810L, 0x00080800L, 0x00080000L, 0x20000810L,
    0x20000010L, 0x20080000L, 0x00080810L, 0x00000000L,
    0x00000800L, 0x20000010L, 0x20000810L, 0x20080800L,
    0x20080000L, 0x00000810L, 0x00000010L, 0x20080010L,
#endif
  },
  { 0x00001000L, 0x00000080L, 0x00400080L, 0x00400001L,
    0x00401081L, 0x00001001L, 0x00001080L, 0x00000000L,
    0x00400000L, 0x00400081L, 0x00000081L, 0x00401000L,
    0x00000001L, 0x00401080L, 0x00401000L, 0x00000081L,
    0x00400081L, 0x00001000L, 0x00001001L, 0x00401081L,
    0x00000000L, 0x00400080L, 0x00400001L, 0x00001080L,
    0x00401001L, 0x00001081L, 0x00401080L, 0x00000001L,
    0x00001081L, 0x00401001L, 0x00000080L, 0x00400000L,
    0x00001081L, 0x00401000L, 0x00401001L, 0x00000081L,
    0x00001000L, 0x00000080L, 0x00400000L, 0x00401001L,
    0x00400081L, 0x00001081L, 0x00001080L, 0x00000000L,
    0x00000080L, 0x00400001L, 0x00000001L, 0x00400080L,
    0x00000000L, 0x00400081L, 0x00400080L, 0x00001080L,
    0x00000081L, 0x00001000L, 0x00401081L, 0x00400000L,
    0x00401080L, 0x00000001L, 0x00001001L, 0x00401081L,
    0x00400001L, 0x00401080L, 0x00401000L, 0x00001001L,
#ifdef BIG_TABLES
    0x00001000L, 0x00000080L, 0x00400080L, 0x00400001L,
    0x00401081L, 0x00001001L, 0x00001080L, 0x00000000L,
    0x00400000L, 0x00400081L, 0x00000081L, 0x00401000L,
    0x00000001L, 0x00401080L, 0x00401000L, 0x00000081L,
    0x00400081L, 0x00001000L, 0x00001001L, 0x00401081L,
    0x00000000L, 0x00400080L, 0x00400001L, 0x00001080L,
    0x00401001L, 0x00001081L, 0x00401080L, 0x00000001L,
    0x00001081L, 0x00401001L, 0x00000080L, 0x00400000L,
    0x00001081L, 0x00401000L, 0x00401001L, 0x00000081L,
    0x00001000L, 0x00000080L, 0x00400000L, 0x00401001L,
    0x00400081L, 0x00001081L, 0x00001080L, 0x00000000L,
    0x00000080L, 0x00400001L, 0x00000001L, 0x00400080L,
    0x00000000L, 0x00400081L, 0x00400080L, 0x00001080L,
    0x00000081L, 0x00001000L, 0x00401081L, 0x00400000L,
    0x00401080L, 0x00000001L, 0x00001001L, 0x00401081L,
    0x00400001L, 0x00401080L, 0x00401000L, 0x00001001L,

    0x00001000L, 0x00000080L, 0x00400080L, 0x00400001L,
    0x00401081L, 0x00001001L, 0x00001080L, 0x00000000L,
    0x00400000L, 0x00400081L, 0x00000081L, 0x00401000L,
    0x00000001L, 0x00401080L, 0x00401000L, 0x00000081L,
    0x00400081L, 0x00001000L, 0x00001001L, 0x00401081L,
    0x00000000L, 0x00400080L, 0x00400001L, 0x00001080L,
    0x00401001L, 0x00001081L, 0x00401080L, 0x00000001L,
    0x00001081L, 0x00401001L, 0x00000080L, 0x00400000L,
    0x00001081L, 0x00401000L, 0x00401001L, 0x00000081L,
    0x00001000L, 0x00000080L, 0x00400000L, 0x00401001L,
    0x00400081L, 0x00001081L, 0x00001080L, 0x00000000L,
    0x00000080L, 0x00400001L, 0x00000001L, 0x00400080L,
    0x00000000L, 0x00400081L, 0x00400080L, 0x00001080L,
    0x00000081L, 0x00001000L, 0x00401081L, 0x00400000L,
    0x00401080L, 0x00000001L, 0x00001001L, 0x00401081L,
    0x00400001L, 0x00401080L, 0x00401000L, 0x00001001L,

    0x00001000L, 0x00000080L, 0x00400080L, 0x00400001L,
    0x00401081L, 0x00001001L, 0x00001080L, 0x00000000L,
    0x00400000L, 0x00400081L, 0x00000081L, 0x00401000L,
    0x00000001L, 0x00401080L, 0x00401000L, 0x00000081L,
    0x00400081L, 0x00001000L, 0x00001001L, 0x00401081L,
    0x00000000L, 0x00400080L, 0x00400001L, 0x00001080L,
    0x00401001L, 0x00001081L, 0x00401080L, 0x00000001L,
    0x00001081L, 0x00401001L, 0x00000080L, 0x00400000L,
    0x00001081L, 0x00401000L, 0x00401001L, 0x00000081L,
    0x00001000L, 0x00000080L, 0x00400000L, 0x00401001L,
#endif
  }
};
```

```
    0x00400081L, 0x00001081L, 0x00001080L, 0x00000000L,
    0x00000080L, 0x00400001L, 0x00000001L, 0x00400080L,
    0x00000000L, 0x00400081L, 0x00400080L, 0x00001080L,
    0x00000081L, 0x00001000L, 0x00401081L, 0x00400000L,
    0x00401080L, 0x00000001L, 0x00001001L, 0x00401081L,
    0x00400001L, 0x00401080L, 0x00401000L, 0x00001001L,
#endif
  },
  { 0x08200020L, 0x08208000L, 0x00008020L, 0x00000000L,
    0x08008000L, 0x00200020L, 0x08200000L, 0x08208020L,
    0x00000020L, 0x08000000L, 0x00208000L, 0x00008020L,
    0x00208020L, 0x08008020L, 0x08000020L, 0x08200000L,
    0x00008000L, 0x00208020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x08000020L, 0x00000000L, 0x00208000L,
    0x08000000L, 0x00200000L, 0x08008020L, 0x08200020L,
    0x00200000L, 0x00008000L, 0x08208000L, 0x00000020L,
    0x00200000L, 0x00008000L, 0x08000020L, 0x08208020L,
    0x00008020L, 0x08000000L, 0x00000000L, 0x00208000L,
    0x08200020L, 0x08008020L, 0x08008000L, 0x00200020L,
    0x08208000L, 0x00000020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x00200000L, 0x08200000L, 0x08000020L,
    0x00208000L, 0x00008020L, 0x08008020L, 0x08200000L,
    0x00000020L, 0x08208000L, 0x00208020L, 0x00000000L,
    0x08000000L, 0x08200020L, 0x00008000L, 0x00208020L,
#ifdef BIG_TABLES
    0x08200020L, 0x08208000L, 0x00008020L, 0x00000000L,
    0x08008000L, 0x00200020L, 0x08200000L, 0x08208020L,
    0x00000020L, 0x08000000L, 0x00208000L, 0x00008020L,
    0x00208020L, 0x08008020L, 0x08000020L, 0x08200000L,
    0x00008000L, 0x00208020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x08000020L, 0x00000000L, 0x00208000L,
    0x08000000L, 0x00200000L, 0x08008020L, 0x08200020L,
    0x00200000L, 0x00008000L, 0x08208000L, 0x00000020L,
    0x00200000L, 0x00008000L, 0x08000020L, 0x08208020L,
    0x00008020L, 0x08000000L, 0x00000000L, 0x00208000L,
    0x08200020L, 0x08008020L, 0x08008000L, 0x00200020L,
    0x08208000L, 0x00000020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x00200000L, 0x08200000L, 0x08000020L,
    0x00208000L, 0x00008020L, 0x08008020L, 0x08200000L,
    0x00000020L, 0x08208000L, 0x00208020L, 0x00000000L,
    0x08000000L, 0x08200020L, 0x00008000L, 0x00208020L,

    0x08200020L, 0x08208000L, 0x00008020L, 0x00000000L,
    0x08008000L, 0x00200020L, 0x08200000L, 0x08208020L,
    0x00000020L, 0x08000000L, 0x00208000L, 0x00008020L,
    0x00208020L, 0x08008020L, 0x08000020L, 0x08200000L,
    0x00008000L, 0x00208020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x08000020L, 0x00000000L, 0x00208000L,
    0x08000000L, 0x00200000L, 0x08008020L, 0x08200020L,
    0x00200000L, 0x00008000L, 0x08208000L, 0x00000020L,
    0x00200000L, 0x00008000L, 0x08000020L, 0x08208020L,
    0x00008020L, 0x08000000L, 0x00000000L, 0x00208000L,
    0x08200020L, 0x08008020L, 0x08008000L, 0x00200020L,
    0x08208000L, 0x00000020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x00200000L, 0x08200000L, 0x08000020L,
    0x00208000L, 0x00008020L, 0x08008020L, 0x08200000L,
    0x00000020L, 0x08208000L, 0x00208020L, 0x00000000L,
    0x08000000L, 0x08200020L, 0x00008000L, 0x00208020L,

    0x08200020L, 0x08208000L, 0x00008020L, 0x00000000L,
    0x08008000L, 0x00200020L, 0x08200000L, 0x08208020L,
    0x00000020L, 0x08000000L, 0x00208000L, 0x00008020L,
    0x00208020L, 0x08008020L, 0x08000020L, 0x08200000L,
    0x00008000L, 0x00208020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x08000020L, 0x00000000L, 0x00208000L,
    0x08000000L, 0x00200000L, 0x08008020L, 0x08200020L,
    0x00200000L, 0x00008000L, 0x08208000L, 0x00000020L,
    0x00200000L, 0x00008000L, 0x08000020L, 0x08208020L,
    0x00008020L, 0x08000000L, 0x00000000L, 0x00208000L,
    0x08200020L, 0x08008020L, 0x08008000L, 0x00200020L,
    0x08208000L, 0x00000020L, 0x00200020L, 0x08008000L,
    0x08208020L, 0x00200000L, 0x08200000L, 0x08000020L,
    0x00208000L, 0x00008020L, 0x08008020L, 0x08200000L,
    0x00000020L, 0x08208000L, 0x00208020L, 0x00000000L,
    0x08000000L, 0x08200020L, 0x00008000L, 0x00208020L,
#endif
  }
};
```

```
unsigned long bs_tab[] =
{   0x00000001L, 0x00000002L, 0x00000004L, 0x00000008L,
    0x00000010L, 0x00000020L, 0x00000040L, 0x00000080L,
    0x00000100L, 0x00000200L, 0x00000400L, 0x00000800L,
    0x00001000L, 0x00002000L, 0x00004000L, 0x00008000L,
    0x00010000L, 0x00020000L, 0x00040000L, 0x00080000L,
    0x00100000L, 0x00200000L, 0x00400000L, 0x00800000L,
    0x01000000L, 0x02000000L, 0x04000000L, 0x08000000L,
    0x10000000L, 0x20000000L, 0x40000000L, 0x80000000L
};

extern unsigned long    key[32];

void des_ec(const void *i_blk, void *o_blk, void *key)
{   unsigned long   q0, q1, l0, l1, tt;

    q0 = *(((unsigned long*)i_blk) + 1);
    q1 = *(((unsigned long*)i_blk));

    ip(q0, q1);

    round(q0, q1,  0); round(q1, q0,  2);
    round(q0, q1,  4); round(q1, q0,  6);
    round(q0, q1,  8); round(q1, q0, 10);
    round(q0, q1, 12); round(q1, q0, 14);
    round(q0, q1, 16); round(q1, q0, 18);
    round(q0, q1, 20); round(q1, q0, 22);
    round(q0, q1, 24); round(q1, q0, 26);
    round(q0, q1, 28); round(q1, q0, 30);

    fp(q1, q0);

    *(((unsigned long*)o_blk)) = q0;
    *(((unsigned long*)o_blk) + 1) = q1;
};

void des_dc(const void *i_blk, void *o_blk, void *key)
{   unsigned long   q0, q1, l0, l1, tt;

    q0 = *(((unsigned long*)i_blk) + 1);
    q1 = *(((unsigned long*)i_blk));

    ip(q0, q1);

    round(q0, q1, 30); round(q1, q0, 28);
    round(q0, q1, 26); round(q1, q0, 24);
    round(q0, q1, 22); round(q1, q0, 20);
    round(q0, q1, 18); round(q1, q0, 16);
    round(q0, q1, 14); round(q1, q0, 12);
    round(q0, q1, 10); round(q1, q0,  8);
    round(q0, q1,  6); round(q1, q0,  4);
    round(q0, q1,  2); round(q1, q0,  0);

    fp(q1, q0);

    *(((unsigned long*)o_blk)) = q0;
    *(((unsigned long*)o_blk) + 1) = q1;
};

/* modified encryption routine for DEAL */

void des_ecm(const void *i_blk, void *o_blk, void *key)
{   unsigned long   q0, q1, l0, l1;

    q0 = *(((unsigned long*)i_blk) + 1);
    q1 = *(((unsigned long*)i_blk));

    round(q0, q1,  0); round(q1, q0,  2);
    round(q0, q1,  4); round(q1, q0,  6);
    round(q0, q1,  8); round(q1, q0, 10);
    round(q0, q1, 12); round(q1, q0, 14);
    round(q0, q1, 16); round(q1, q0, 18);
    round(q0, q1, 20); round(q1, q0, 22);
    round(q0, q1, 24); round(q1, q0, 26);
    round(q0, q1, 28); round(q1, q0, 30);

    *(((unsigned long*)o_blk)) = q0;
    *(((unsigned long*)o_blk) + 1) = q1;
};
```

key.c

```
// IBM PC Implementation of the DES Cryptographic Algorithm
//
// Key Schedule Routine
//
// by Dr B R Gladman (gladman@seven77.demon.co.uk)
#include "des.h"
// key shift table
unsigned char   ks_tab[] =
{    0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0
};
unsigned long p2_tab[8][64] =
{
  { 0x00000000L, 0x00000010L, 0x20000000L, 0x20000010L,
    0x00010000L, 0x00010010L, 0x20010000L, 0x20010010L,
    0x00000800L, 0x00000810L, 0x20000800L, 0x20000810L,
    0x00010800L, 0x00010810L, 0x20010800L, 0x20010810L,
    0x00000020L, 0x00000030L, 0x20000020L, 0x20000030L,
    0x00010020L, 0x00010030L, 0x20010020L, 0x20010030L,
    0x00000820L, 0x00000830L, 0x20000820L, 0x20000830L,
    0x00010820L, 0x00010830L, 0x20010820L, 0x20010830L,
    0x00080000L, 0x00080010L, 0x20080000L, 0x20080010L,
    0x00090000L, 0x00090010L, 0x20090000L, 0x20090010L,
    0x00080800L, 0x00080810L, 0x20080800L, 0x20080810L,
    0x00090800L, 0x00090810L, 0x20090800L, 0x20090810L,
    0x00080020L, 0x00080030L, 0x20080020L, 0x20080030L,
    0x00090020L, 0x00090030L, 0x20090020L, 0x20090030L,
    0x00080820L, 0x00080830L, 0x20080820L, 0x20080830L,
    0x00090820L, 0x00090830L, 0x20090820L, 0x20090830L
  },
  { 0x00000000L, 0x00200000L, 0x00000004L, 0x00200004L,
    0x00000400L, 0x00200400L, 0x00000404L, 0x00200404L,
    0x10000000L, 0x10200000L, 0x10000004L, 0x10200004L,
    0x10000400L, 0x10200400L, 0x10000404L, 0x10200404L,
    0x00000001L, 0x00200001L, 0x00000005L, 0x00200005L,
    0x00000401L, 0x00200401L, 0x00000405L, 0x00200405L,
    0x10000001L, 0x10200001L, 0x10000005L, 0x10200005L,
    0x10000401L, 0x10200401L, 0x10000405L, 0x10200405L,
    0x00040000L, 0x00240000L, 0x00040004L, 0x00240004L,
    0x00040400L, 0x00240400L, 0x00040404L, 0x00240404L,
    0x10040000L, 0x10240000L, 0x10040004L, 0x10240004L,
    0x10040400L, 0x10240400L, 0x10040404L, 0x10240404L,
    0x00040001L, 0x00240001L, 0x00040005L, 0x00240005L,
    0x00040401L, 0x00240401L, 0x00040405L, 0x00240405L,
    0x10040001L, 0x10240001L, 0x10040005L, 0x10240005L,
    0x10040401L, 0x10240401L, 0x10040405L, 0x10240405L
  },
  { 0x00000000L, 0x00000002L, 0x00000008L, 0x0000000aL,
    0x00000200L, 0x00000202L, 0x00000208L, 0x0000020aL,
    0x08000000L, 0x08000002L, 0x08000008L, 0x0800000aL,
    0x08000200L, 0x08000202L, 0x08000208L, 0x0800020aL,
    0x00100000L, 0x00100002L, 0x00100008L, 0x0010000aL,
    0x00100200L, 0x00100202L, 0x00100208L, 0x0010020aL,
    0x08100000L, 0x08100002L, 0x08100008L, 0x0810000aL,
    0x08100200L, 0x08100202L, 0x08100208L, 0x0810020aL,
    0x00000100L, 0x00000102L, 0x00000108L, 0x0000010aL,
    0x00000300L, 0x00000302L, 0x00000308L, 0x0000030aL,
    0x08000100L, 0x08000102L, 0x08000108L, 0x0800010aL,
    0x08000300L, 0x08000302L, 0x08000308L, 0x0800030aL,
    0x00100100L, 0x00100102L, 0x00100108L, 0x0010010aL,
    0x00100300L, 0x00100302L, 0x00100308L, 0x0010030aL,
    0x08100100L, 0x08100102L, 0x08100108L, 0x0810010aL,
    0x08100300L, 0x08100302L, 0x08100308L, 0x0810030aL
  },
  { 0x00000000L, 0x01000000L, 0x00001000L, 0x01001000L,
    0x04000000L, 0x05000000L, 0x04001000L, 0x05001000L,
    0x00020000L, 0x01020000L, 0x00021000L, 0x01021000L,
    0x04020000L, 0x05020000L, 0x04021000L, 0x05021000L,
    0x02000000L, 0x03000000L, 0x02001000L, 0x03001000L,
    0x06000000L, 0x07000000L, 0x06001000L, 0x07001000L,
    0x02020000L, 0x03020000L, 0x02021000L, 0x03021000L,
    0x06020000L, 0x07020000L, 0x06021000L, 0x07021000L,
```

```
  0x00002000L, 0x01002000L, 0x00003000L, 0x01003000L,
  0x04002000L, 0x05002000L, 0x04003000L, 0x05003000L,
  0x00022000L, 0x01022000L, 0x00023000L, 0x01023000L,
  0x04022000L, 0x05022000L, 0x04023000L, 0x05023000L,
  0x02002000L, 0x03002000L, 0x02003000L, 0x03003000L,
  0x06002000L, 0x07002000L, 0x06003000L, 0x07003000L,
  0x02022000L, 0x03022000L, 0x02023000L, 0x03023000L,
  0x06022000L, 0x07022000L, 0x06023000L, 0x07023000L
},
{ 0x00000000L, 0x10000000L, 0x00010000L, 0x10010000L,
  0x00000004L, 0x10000004L, 0x00010004L, 0x10010004L,
  0x20000000L, 0x30000000L, 0x20010000L, 0x30010000L,
  0x20000004L, 0x30000004L, 0x20010004L, 0x30010004L,
  0x00100000L, 0x10100000L, 0x00110000L, 0x10110000L,
  0x00100004L, 0x10100004L, 0x00110004L, 0x10110004L,
  0x20100000L, 0x30100000L, 0x20110000L, 0x30110000L,
  0x20100004L, 0x30100004L, 0x20110004L, 0x30110004L,
  0x00001000L, 0x10001000L, 0x00011000L, 0x10011000L,
  0x00001004L, 0x10001004L, 0x00011004L, 0x10011004L,
  0x20001000L, 0x30001000L, 0x20011000L, 0x30011000L,
  0x20001004L, 0x30001004L, 0x20011004L, 0x30011004L,
  0x00101000L, 0x10101000L, 0x00111000L, 0x10111000L,
  0x00101004L, 0x10101004L, 0x00111004L, 0x10111004L,
  0x20101000L, 0x30101000L, 0x20111000L, 0x30111000L,
  0x20101004L, 0x30101004L, 0x20111004L, 0x30111004L
},
{ 0x00000000L, 0x00000008L, 0x00000100L, 0x00000108L,
  0x00000400L, 0x00000408L, 0x00000500L, 0x00000508L,
  0x00020000L, 0x00020008L, 0x00020100L, 0x00020108L,
  0x00020400L, 0x00020408L, 0x00020500L, 0x00020508L,
  0x00000001L, 0x00000009L, 0x00000101L, 0x00000109L,
  0x00000401L, 0x00000409L, 0x00000501L, 0x00000509L,
  0x00020001L, 0x00020009L, 0x00020101L, 0x00020109L,
  0x00020401L, 0x00020409L, 0x00020501L, 0x00020509L,
  0x02000000L, 0x02000008L, 0x02000100L, 0x02000108L,
  0x02000400L, 0x02000408L, 0x02000500L, 0x02000508L,
  0x02020000L, 0x02020008L, 0x02020100L, 0x02020108L,
  0x02020400L, 0x02020408L, 0x02020500L, 0x02020508L,
  0x02000001L, 0x02000009L, 0x02000101L, 0x02000109L,
  0x02000401L, 0x02000409L, 0x02000501L, 0x02000509L,
  0x02020001L, 0x02020009L, 0x02020101L, 0x02020109L,
  0x02020401L, 0x02020409L, 0x02020501L, 0x02020509L
},
{ 0x00000000L, 0x00080000L, 0x01000000L, 0x01080000L,
  0x00000010L, 0x00080010L, 0x01000010L, 0x01080010L,
  0x00200000L, 0x00280000L, 0x01200000L, 0x01280000L,
  0x00200010L, 0x00280010L, 0x01200010L, 0x01280010L,
  0x00000200L, 0x00080200L, 0x01000200L, 0x01080200L,
  0x00000210L, 0x00080210L, 0x01000210L, 0x01080210L,
  0x00200200L, 0x00280200L, 0x01200200L, 0x01280200L,
  0x00200210L, 0x00280210L, 0x01200210L, 0x01280210L,
  0x04000000L, 0x04080000L, 0x05000000L, 0x05080000L,
  0x04000010L, 0x04080010L, 0x05000010L, 0x05080010L,
  0x04200000L, 0x04280000L, 0x05200000L, 0x05280000L,
  0x04200010L, 0x04280010L, 0x05200010L, 0x05280010L,
  0x04000200L, 0x04080200L, 0x05000200L, 0x05080200L,
  0x04000210L, 0x04080210L, 0x05000210L, 0x05080210L,
  0x04200200L, 0x04280200L, 0x05200200L, 0x05280200L,
  0x04200210L, 0x04280210L, 0x05200210L, 0x05280210L
},
{ 0x00000000L, 0x00002000L, 0x08000000L, 0x08002000L,
  0x00000020L, 0x00002020L, 0x08000020L, 0x08002020L,
  0x00000800L, 0x00002800L, 0x08000800L, 0x08002800L,
  0x00000820L, 0x00002820L, 0x08000820L, 0x08002820L,
  0x00040000L, 0x00042000L, 0x08040000L, 0x08042000L,
  0x00040020L, 0x00042020L, 0x08040020L, 0x08042020L,
  0x00040800L, 0x00042800L, 0x08040800L, 0x08042800L,
  0x00040820L, 0x00042820L, 0x08040820L, 0x08042820L,
  0x00000002L, 0x00002002L, 0x08000002L, 0x08002002L,
  0x00000022L, 0x00002022L, 0x08000022L, 0x08002022L,
  0x00000802L, 0x00002802L, 0x08000802L, 0x08002802L,
  0x00000822L, 0x00002822L, 0x08000822L, 0x08002822L,
  0x00040002L, 0x00042002L, 0x08040002L, 0x08042002L,
  0x00040022L, 0x00042022L, 0x08040022L, 0x08042022L,
  0x00040802L, 0x00042802L, 0x08040802L, 0x08042802L,
  0x00040822L, 0x00042822L, 0x08040822L, 0x08042822L
}
```

```
};
void des_ky(void *kval, void *key)
{   unsigned long   v0, v1, l0, l1, tt;
    unsigned short  i;
    l0 = *((unsigned long*)kval);
    l1 = *(((unsigned long*)kval) + 1);
    bit_swap(l1, l0, 4, 0x0f0f0f0f);
    bit_swap(l0, l1, 4, 0x01010101);
    bit_swap(l0, l0, 9, 0x00550055);
    bit_swap(l1, l1, 9, 0x00550055);
    bit_swap(l0, l0, 18, 0x00003333);
    bit_swap(l1, l1, 18, 0x00003333);
    bit_swap(l1, l1, 16, 0x000000ff);
    for(i = 0; i < 16; i++)
    {
        l0 &= 0x0fffffffL; l1 &= 0x0fffffffL;
        if(ks_tab[i])
        {
            l0 = (l0 >> 2) | (l0 << 26);
            l1 = (l1 >> 2) | (l1 << 26);
        }
        else
        {
            l0 = (l0 >> 1) | (l0 << 27);
            l1 = (l1 >> 1) | (l1 << 27);
        }
        v0 = p2_tab[0][byte(l0,0) & 0x3f]
           | p2_tab[1][(byte(l0,1) >> 1) & 0x3f] |
p2_tab[2][byte(l0,2) & 0x1d
                        | (byte(l0,2) >> 1) & 0x20 |
(byte(l0,2) >> 6) & 0x02]
           | p2_tab[3][(byte(l0,0) >> 2) & 0x30 |
(byte(l0,1) >> 7) & 0x01
                        | byte(l0,3) & 0x0e];
        v1 = p2_tab[4][byte(l1,0) & 0x3f]
           | p2_tab[5][byte(l1,1) & 0x3d |
(byte(l1,1) >> 6) & 0x02]
           | p2_tab[6][byte(l1,2) & 0x3f]
           | p2_tab[7][(byte(l1,0) >> 6) & 0x02 |
(byte(l1,2) >> 2) & 0x30
                        | byte(l1,3) & 0x0d];
        ((unsigned long*)key)[2 * i]     =
(v0 & 0x0000ffff) | (v1 << 16);
        ((unsigned long*)key)[2 * i + 1] =
(v1 & 0xffff0000) | (v0 >> 16);
    }
};
```

des.h

```
//  IBM PC Implementation of the DES Cryptographic Algorithm by
//  Dr B. R. Gladman (gladman@seven77.demon.co.uk)
//
//  Some of the techniques in this DES source code are derived
//  from ideas developed by Richard Outerbridge and Eric Young.
//  I gratefully acknowledge their contribution.
//
//  Note on Bit Numbering.  The DES bit numbering is the
//  reverse of that
```

```
//  used on the intel series processors.  Thus to translate
//  between bits
//  and numeric values requires a reversal of bit sequences.
//  To achieve
//  this for external numbers the initial and final DES
//  permutations and
//  the initial key permutation are adjusted to take care
//  of the changed
//  bit order.  The other changes required are in the
//  calculation of the
//  s_box inputs and outputs. The bits numbering reversal
//  on s_box input
//  is obtained by reordering the s_box tables.
//  The bit reversal within
//  output nibbles is done by reordering the exit permutation.

#ifndef byte
#  define byte(x,n)    ((unsigned char)((x) >> (8 * (n))))
#endif

#ifndef _MSC_VER

#define rotr(x,n)    (((x) >> ((int)(n))) | ((x) << (32 - (int)(n))))
#define rotl(x,n)    (((x) << ((int)(n))) | ((x) >> (32 - (int)(n))))

#else

#include <stdlib.h>

#pragma intrinsic(_lrotr,_lrotl)
#define rotr(x,n)   _lrotr(x,n)
#define rotl(x,n)   _lrotl(x,n)

#endif

#define bit_swap(a,b,n,m)       \
    tt = ((a >> n) ^ b) & m;    \
    b ^= tt; a ^= (tt << n)

#define ip_old(x,y)                         \
    bit_swap((x),(y),  4, 0x0f0f0f0fL); \
    bit_swap((y),(x), 16, 0x0000ffffL); \
    bit_swap((x),(y),  2, 0x33333333L); \
    bit_swap((y),(x),  8, 0x00ff00ffL); \
    bit_swap((x),(y),  1, 0x55555555L); \
    (x) = rotl((x), 1);                 \
    (y) = rotl((y), 1)

#define fp_old(x,y)                         \
    (y) = rotr((y), 1);                 \
    (x) = rotr((x), 1);                 \
    bit_swap((x),(y),  1, 0x55555555L); \
    bit_swap((y),(x),  8, 0x00ff00ffL); \
    bit_swap((x),(y),  2, 0x33333333L); \
    bit_swap((y),(x), 16, 0x0000ffffL); \
    bit_swap((x),(y),  4, 0x0f0f0f0fL)

#define ip(x,y)                     \
    (x) = rotr((x), 4);             \
    tt = ((x) ^ (y)) & 0x0f0f0f0fL; \
    (y) ^= tt;                      \
    (x) = rotr((x) ^ tt, 12);       \
    tt = ((y) ^ (x)) & 0xffff0000L; \
    (y) ^= tt;                      \
    (x) = rotr((x) ^ tt, 18);       \
    tt = ((x) ^ (y)) & 0x33333333L; \
    (y) ^= tt;                      \
    (x) = rotr((x) ^ tt, 22);       \
    tt = ((y) ^ (x)) & 0xff00ff00L; \
    (y) ^= tt;                      \
    (x) = rotr((x) ^ tt,  9);       \
    tt = ((x) ^ (y)) & 0x55555555L; \
    (x) = rotl((x) ^ tt, 2);        \
    (y) = rotl((y) ^ tt, 1)

#define fp(x,y)                     \
```

```
    (y) = rotr((y), 1);             \
    (x) = rotr((x), 2);             \
    tt = ((x) ^ (y)) & 0x55555555L; \
    (y) ^= tt;                      \
    (x) = rotl((x) ^ tt,  9);       \
    tt = ((y) ^ (x)) & 0xff00ff00L; \
    (y) ^= tt;                      \
    (x) = rotl((x) ^ tt, 22);       \
    tt = ((x) ^ (y)) & 0x33333333L; \
    (y) ^= tt;                      \
    (x) = rotl((x) ^ tt, 18);       \
    tt = ((y) ^ (x)) & 0xffff0000L; \
    (y) ^= tt;                      \
    (x) = rotl((x) ^ tt, 12);       \
    tt = ((x) ^ (y)) & 0x0f0f0f0fL; \
    (y) ^= tt;                      \
    (x) = rotl((x) ^ tt, 4)
#ifdef  BIG_TABLES
#define round(x0,x1,ki)                                                 \
    l1  = (rotr(x1, 4) ^ *(((unsigned long*)key) + ki + 1));    \
    l0  = (x1 ^ *(((unsigned long*)key) + ki));                 \
    x0 ^= sx_tab[0][byte(l0,0)] | sx_tab[1][byte(l1,0)]         \
        | sx_tab[2][byte(l0,1)] | sx_tab[3][byte(l1,1)]         \
        | sx_tab[4][byte(l0,2)] | sx_tab[5][byte(l1,2)]         \
        | sx_tab[6][byte(l0,3)] | sx_tab[7][byte(l1,3)]
#else
#define round(x0,x1,ki)                                 \
    l1  = (rotr(x1, 4) ^ *(((unsigned long*)key) + ki + 1))
& 0x3f3f3f3f;   \
    l0  = (x1 ^ *(((unsigned long*)key) + ki))
& 0x3f3f3f3f;                   \
    x0 ^= sx_tab[0][byte(l0,0)] | sx_tab[1][byte(l1,0)]         \
        | sx_tab[2][byte(l0,1)] | sx_tab[3][byte(l1,1)]         \
        | sx_tab[4][byte(l0,2)] | sx_tab[5][byte(l1,2)]         \
        | sx_tab[6][byte(l0,3)] | sx_tab[7][byte(l1,3)]
#endif
#ifdef  __cplusplus
extern "C"
{
    void des_ky(void *kval, void *key);
    void des_ec(const void *i_blk, void *o_blk, void *key);
    void des_dc(const void *i_blk, void *o_blk, void *key);
};
#else
void des_ky(void *kval, void *key);
void des_ec(const void *i_blk, void *o_blk, void *key);
void des_dc(const void *i_blk, void *o_blk, void *key);
#endif
```

Apêndice B

Implementação do S-DES em Linguagem C

Este anexo consiste num exemplo de implementação do modelo simplificado do cifrador S-DES em linguagem de programação C.

sdes.h

```
void p10(char key[]);
void shift(char key[]);
void p8(char key[], char k[]);
void ip(char input[]);
void ip_1(char input[]);
void ep(char four_bit[], char eigth_bit[]);
void separa(char input[], char four_bit_1[], char four_bit_2[]);
void xor(char input[], char x[], int size);
void p4(char r_s0[], char r_s1[], char r[]);
void s_box(char l[], char r_s0[], int op);
void fk(char input[], char k[]);
void sw(char input[]);
// Variaveis globais de S0 e S1
//
char s0[4][4][2] = {
"01", "00", "11", "10",
"11", "10", "01", "00",
"00", "10", "01", "11",
"11", "01", "11", "10",
};
char s1[4][4][2] = {
"00", "01", "10", "11",
"10", "00", "01", "11",
"11", "00", "01", "00",
"10", "01", "00", "11",
};

void p10(char key[]) {
char aux;
aux = key[0];
key[0] = key[2];
key[2] = key[1];
key[1] = key[4];
```

```
key[4] = key[3];
key[3] = key[6];
key[6] = aux;
aux = key[5];
key[5] = key[9];
key[9] = aux;
aux = key[7];
key[7] = key[8];
key[8] = aux;
}
void shift(char key[]) {
char p1[5], p2[5];
char aux, aux2;
int i;
for(i=0; i<10; i++) {
if(i<5)
p1[i] = key[i];
else
p2[i-5] = key[i];
}
aux = p1[0];
aux2 = p2[0];
for(i=0;i<4;i++){
p1[i] = p1[i+1];
p2[i] = p2[i+1];
}
p1[4] = aux;
p2[4] = aux2;
for(i=0;i<10;i++) {
if(i<5)
key[i] = p1[i];
else
key[i] = p2[i-5];
}
key[10] = '\0';
}
void p8(char key[], char k[]){
char aux[10];
strcpy(aux, key);
k[0] = key[5];
k[1] = key[2];
k[2] = key[6];
k[3] = key[3];
k[4] = key[7];
k[5] = key[4];
k[6] = key[9];
k[7] = key[8];
k[8] = '\0';
strcpy(key, aux);
}
void ip(char input[]) {
char aux;
aux = input[0];
input[0] = input[1];
input[1] = input[5];
input[5] = input[7];
input[7] = input[6];
input[6] = input[4];
input[4] = input[3];
input[3] = aux;
}
void ip_1(char input[]) {
char aux;
aux = input[0];
input[0] = input[3];
input[3] = input[4];
input[4] = input[6];
```

```
input[6] = input[7];
input[7] = input[5];
input[5] = input[1];
input[1] = aux;
}

void ep(char four_bit[], char eigth_bit[]) {
eigth_bit[0] = four_bit[3];
eigth_bit[1] = four_bit[0];
eigth_bit[2] = four_bit[1];
eigth_bit[3] = four_bit[2];
eigth_bit[4] = four_bit[1];
eigth_bit[5] = four_bit[2];
eigth_bit[6] = four_bit[3];
eigth_bit[7] = four_bit[0];
// eigth_bit[8] = '\0';
}

void separa(char input[], char l[], char r[]) {
int i;
for(i=0; i < 8; i++) {
if(i < 4)
l[i] = input[i];
else
r[i-4] = input[i];
}
}

void xor(char input[], char k[], int size) {
int i;
for (i=0; i < size; i++)  {
if (input[i] == k[i])
input[i] = '0';
else
input[i] = '1';
}
}

void p4(char r_s0[], char r_s1[], char r[]) {
r[0] = r_s0[1];
r[1] = r_s1[1];
r[2] = r_s1[0];
r[3] = r_s0[0];
}

void s_box(char l[], char r_s[], int op) {
char bin[2];
int i, j;

if(l[0] == '0')
if(l[3] == '0') i = 0;
else i = 1;
else
if(l[3] == '0') i = 2;
else i = 3;

if(l[1] == '0')
if(l[2] == '0') j = 0;
else j = 1;
else
if(l[2] == '0') j = 2;
else j = 3;

if(op == 0){
r_s[0] = s0[i][j][0];
r_s[1] = s0[i][j][1];
}
else {
r_s[0] = s1[i][j][0];
r_s[1] = s1[i][j][1];
}

}

void fk(char input[], char k[]) {

int i;
```

```
int j;
char l[4], r[4], aux_l[4], aux_r[4];
char r_s0[2], r_s1[2];

separa(input, l, r);
for (i=0;i<4;i++) {
aux_l[i] = l[i];
aux_r[i] = r[i];
}
ep(r, input);
xor(input,k,8);

separa(input,l ,r);
s_box(l, r_s0, 0);
s_box(r, r_s1, 1);

p4(r_s0, r_s1, r);
xor(aux_l,r,4);
for (i=0; i<4; i++) {
input[i] = aux_l[i];
input[i+4] = aux_r[i];
}
}

void sw(char input[]) {
char aux;
int i;
for(i=0; i<4; i++) {
aux = input[i+4];
input[i+4] = input[i];
input[i] = aux;
}
}
```

sdes.c

```
#include <stdio.h>
#include <string.h>
#include "./sdes.h"

int parameters(int argc);
int verify_key(char key[]);
// Bloco Principal
//
int main(int argc, char *argv[]) {

// Declaracao de variaveis.
int  i;
        char key[10], tmp_key[10];
char k1[8], k2[8];
char input[8], output[8];
FILE *f_input, *f_output;

i = 0;

// Verificacao de parametros
//
if (parameters(argc) != 1) {
if (parameters(argc) == 2)
printf("Poucos parametros para o programa. \n");
else
printf("Muitos paramentros para o programa. \n");
printf("Sintaxe correta: ./encrypt input output 10-b-key.\n");
exit(0);
}

// Abrir arquivo principal
//
if (!(f_input = fopen(argv[1],"r"))) {
printf("Erro ao abrir o arquivo %s\n", argv[0]);
exit(0);
}

//Verificar se a chave e binaria
//
if (verify_key(argv[3]) != 1) {
```

```
if (verify_key(argv[3]) == 2) {
printf("Comprimento de Chave Inválida\n");
exit(0);
}
else {
printf("Valor de Chave Inválido\n");
exit(0);
}
}
strcpy(key,argv[3]);

//Agendamento de chaves
p10(key);
shift(key);
p8(key,k1);
shift(key); shift(key);
p8(key, k2);

// Ler o arquivo de entrada...
rewind(f_input);
fread(input,sizeof(char),8,f_input);
input[8] = '\0';

// Encriptacao
ip(input);
fk(input,k1);
sw(input);
fk(input,k2);
ip_1(input);

// Abrir arquivo de saida
//
if (!(f_output = fopen(argv[2],"w"))) {
printf("Erro ao abrir o arquivo %s\n", argv[2]);
exit(0);
}

for (i=0; i<8; i++)
fprintf(f_output,"%c", input[i]);
fclose(f_output);

// Decriptacao
ip(input);
fk(input,k2);
sw(input);
fk(input,k1);
ip_1(input);

return 1;
}

// Funcao que verifica paramentros OK
int parameters(int argc) {
if (argc < 4)
return 2;
else if (argc > 4)
return 3;
else
return 1;
}

// Funcao que verifica chave OK
int verify_key(char key[]){

char chave[10];
int i, err;
err = 0;

if (strlen(key) != 10 )
return 2;
else
for (i=0; i < 10; i++)
if ((key[i] != '1') && (key[i] != '0')){
return 3;
err = 1;
}
if (err == 0)
```

```
return 1;
}
```

Apêndice C

Divulgação do código do cifrador RC4

Este anexo consiste na mensagem original postada anonimamente em 9 de Setembro de 1994 na lista de mensagens *Cypherpunks*[1] e que divulgou o código do cifrador RC4.

A mensagem

```
(fwd) RC4 Algorithm revealed.

      To: cypherpunks@toad.com
      Subject: (fwd) RC4 Algorithm revealed.
      From: tcmay@netcom.com (Timothy C. May)
      Date: Wed, 14 Sep 1994 00:16:51 -0700 (PDT)
      Cc: tcmay@netcom.com (Timothy C. May)
      Sender: owner-cypherpunks@toad.com


Someone (probably one of you!) has posted this item to several
newsgroups.

Note the forgery, a la port 25, of "David Sterndark"'s name.

I, too, am shocked, simply shocked, and will be nofifying the "Casa
Blanca" of this breach.

-TCM

Newsgroups: sci.crypt,alt.security,comp.security.misc,alt.privacy
Path: netcom.com!sterndark
From: sterndark@netcom.com (David Sterndark)
Subject: RC4 Algorithm revealed.
Message-ID: <sternCvKL4B.Hyy@netcom.com>
Sender: sterndark@netcom.com
Organization: NETCOM On-line Communication Services
(408 261-4700 guest)
Date: Wed, 14 Sep 1994 06:35:31 GMT

I am shocked,  shocked, I tell you,  shocked, to discover
that the cypherpunks have illegaly and criminally revealed
a crucial RSA trade secret and harmed the security of
America by reverse engineering the RC4 algorithm and
publishing it to the world.

On Saturday morning an anonymous cypherpunk wrote:
```

[1] http://cypherpunks.venona.com/date/1994/09/msg00434.html

SUBJECT: RC4 Source Code

I've tested this. It is compatible with the RC4 object module that comes in the various RSA toolkits.

```
/* rc4.h */
typedef struct rc4_key
{
     unsigned char state[256];
     unsigned char x;
     unsigned char y;
} rc4_key;
void prepare_key(unsigned char *key_data_ptr,int key_data_len,
rc4_key *key);
void rc4(unsigned char *buffer_ptr,int buffer_len,rc4_key * key);

/*rc4.c */
#include "rc4.h"
static void swap_byte(unsigned char *a, unsigned char *b);
void prepare_key(unsigned char *key_data_ptr, int key_data_len,
rc4_key *key)
{
     unsigned char swapByte;
     unsigned char index1;
     unsigned char index2;
     unsigned char* state;
     short counter;

     state = &key->state[0];
     for(counter = 0; counter < 256; counter++)
     state[counter] = counter;
     key->x = 0;
     key->y = 0;
     index1 = 0;
     index2 = 0;
     for(counter = 0; counter < 256; counter++)
     {
          index2 = (key_data_ptr[index1] + state[counter] +
             index2) % 256;
          swap_byte(&state[counter], &state[index2]);

          index1 = (index1 + 1) % key_data_len;
     }
}
void rc4(unsigned char *buffer_ptr, int buffer_len, rc4_key *key)
{
     unsigned char x;
     unsigned char y;
     unsigned char* state;
     unsigned char xorIndex;
     short counter;

     x = key->x;
     y = key->y;

     state = &key->state[0];
     for(counter = 0; counter < buffer_len; counter ++)
     {
          x = (x + 1) % 256;
          y = (state[x] + y) % 256;
          swap_byte(&state[x], &state[y]);

          xorIndex = (state[x] + state[y]) % 256;

          buffer_ptr[counter] ^= state[xorIndex];
      }
      key->x = x;
      key->y = y;
}
static void swap_byte(unsigned char *a, unsigned char *b)
{
     unsigned char swapByte;
```

```
        swapByte = *a;
        *a = *b;
        *b = swapByte;
    }
```

Another cypherpunk, this one not anonymous, tested the output from this algorithm against the output from official RC4 object code

Date: Tue, 13 Sep 94 18:37:56 PDT
From: ekr@eit.COM (Eric Rescorla)
Message-Id: <9409140137.AA17743@eitech.eit.com>
Subject: RC4 compatibility testing
Cc: cypherpunks@toad.com

One data point:

I can't say anything about the internals of RC4 versus the algorithm that Bill Sommerfeld is rightly calling 'Alleged RC4', since I don't know anything about RC4's internals.

However, I do have a (legitimately acquired) copy of BSAFE2 and so I'm able to compare the output of this algorithm to the output of genuine RC4 as found in BSAFE. I chose a set of test vectors and ran them through both algorithms. The algorithms appear to give identical results, at least with these key/plaintext pairs.

I note that this is the algorithm _without_ Hal Finney's proposed modification

(see <199409130605.XAA24133@jobe.shell.portal.com>).

The vectors I used (together with the ciphertext they produce) follow at the end of this message.

-Ekr

Disclaimer: This posting does not reflect the opinions of EIT.

```
--------------------results follow--------------
Test vector 0
Key: 0x01 0x23 0x45 0x67 0x89 0xab 0xcd 0xef
Input: 0x01 0x23 0x45 0x67 0x89 0xab 0xcd 0xef
0 Output: 0x75 0xb7 0x87 0x80 0x99 0xe0 0xc5 0x96

Test vector 1
Key: 0x01 0x23 0x45 0x67 0x89 0xab 0xcd 0xef
Input: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0 Output: 0x74 0x94 0xc2 0xe7 0x10 0x4b 0x08 0x79

Test vector 2
Key: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Input: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0 Output: 0xde 0x18 0x89 0x41 0xa3 0x37 0x5d 0x3a

Test vector 3
Key: 0xef 0x01 0x23 0x45
Input: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0 Output: 0xd6 0xa1 0x41 0xa7 0xec 0x3c 0x38 0xdf 0xbd 0x61

Test vector 4
Key: 0x01 0x23 0x45 0x67 0x89 0xab 0xcd 0xef
Input: 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
```

```
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x01
0 Output: 0x75 0x95 0xc3 0xe6 0x11 0x4a 0x09 0x78 0x0c 0x4a 0xd4
0x52 0x33 0x8e 0x1f 0xfd 0x9a 0x1b 0xe9 0x49 0x8f
0x81 0x3d 0x76 0x53 0x34 0x49 0xb6 0x77 0x8d 0xca
0xd8 0xc7 0x8a 0x8d 0x2b 0xa9 0xac 0x66 0x08 0x5d
0x0e 0x53 0xd5 0x9c 0x26 0xc2 0xd1 0xc4 0x90 0xc1
0xeb 0xbe 0x0c 0xe6 0x6d 0x1b 0x6b 0x1b 0x13 0xb6
0xb9 0x19 0xb8 0x47 0xc2 0x5a 0x91 0x44 0x7a 0x95
0xe7 0x5e 0x4e 0xf1 0x67 0x79 0xcd 0xe8 0xbf 0x0a
0x95 0x85 0x0e 0x32 0xaf 0x96 0x89 0x44 0x4f 0xd3
0x77 0x10 0x8f 0x98 0xfd 0xcb 0xd4 0xe7 0x26 0x56
0x75 0x00 0x99 0x0b 0xcc 0x7e 0x0c 0xa3 0xc4 0xaa
0xa3 0x04 0xa3 0x87 0xd2 0x0f 0x3b 0x8f 0xbb 0xcd
0x42 0xa1 0xbd 0x31 0x1d 0x7a 0x43 0x03 0xdd 0xa5
0xab 0x07 0x88 0x96 0xae 0x80 0xc1 0x8b 0x0a 0xf6
0x6d 0xff 0x31 0x96 0x16 0xeb 0x78 0x4e 0x49 0x5a
0xd2 0xce 0x90 0xd7 0xf7 0x72 0xa8 0x17 0x47 0xb6
0x5f 0x62 0x09 0x3b 0x1e 0x0d 0xb9 0xe5 0xba 0x53
0x2f 0xaf 0xec 0x47 0x50 0x83 0x23 0xe6 0x71 0x32
0x7d 0xf9 0x44 0x44 0x32 0xcb 0x73 0x67 0xce 0xc8
0x2f 0x5d 0x44 0xc0 0xd0 0x0b 0x67 0xd6 0x50 0xa0
0x75 0xcd 0x4b 0x70 0xde 0xdd 0x77 0xeb 0x9b 0x10
0x23 0x1b 0x6b 0x5b 0x74 0x13 0x47 0x39 0x6d 0x62
0x89 0x74 0x21 0xd4 0x3d 0xf9 0xb4 0x2e 0x44 0x6e
0x35 0x8e 0x9c 0x11 0xa9 0xb2 0x18 0x4e 0xcb 0xef
0x0c 0xd8 0xe7 0xa8 0x77 0xef 0x96 0x8f 0x13 0x90
0xec 0x9b 0x3d 0x35 0xa5 0x58 0x5c 0xb0 0x09 0x29
0x0e 0x2f 0xcd 0xe7 0xb5 0xec 0x66 0xd9 0x08 0x4b
0xe4 0x40 0x55 0xa6 0x19 0xd9 0xdd 0x7f 0xc3 0x16
0x6f 0x94 0x87 0xf7 0xcb 0x27 0x29 0x12 0x42 0x64
0x45 0x99 0x85 0x14 0xc1 0x5d 0x53 0xa1 0x8c 0x86
0x4c 0xe3 0xa2 0xb7 0x55 0x57 0x93 0x98 0x81 0x26
0x52 0x0e 0xac 0xf2 0xe3 0x06 0x6e 0x23 0x0c 0x91
0xbe 0xe4 0xdd 0x53 0x04 0xf5 0xfd 0x04 0x05 0xb3
0x5b 0xd9 0x9c 0x73 0x13 0x5d 0x3d 0x9b 0xc3 0x35
0xee 0x04 0x9e 0xf6 0x9b 0x38 0x67 0xbf 0x2d 0x7b
0xd1 0xea 0xa5 0x95 0xd8 0xbf 0xc0 0x06 0x6f 0xf8
0xd3 0x15 0x09 0xeb 0x0c 0x6c 0xaa 0x00 0x6c 0x80
0x7a 0x62 0x3e 0xf8 0x4c 0x3d 0x33 0xc1 0x95 0xd2
0x3e 0xe3 0x20 0xc4 0x0d 0xe0 0x55 0x81 0x57 0xc8
0x22 0xd4 0xb8 0xc5 0x69 0xd8 0x49 0xae 0xd5 0x9d
0x4e 0x0f 0xd7 0xf3 0x79 0x58 0x6b 0x4b 0x7f 0xf6
0x84 0xed 0x6a 0x18 0x9f 0x74 0x86 0xd4 0x9b 0x9c
0x4b 0xad 0x9b 0xa2 0x4b 0x96 0xab 0xf9 0x24 0x37
0x2c 0x8a 0x8f 0xff 0xb1 0x0d 0x55 0x35 0x49 0x00
0xa7 0x7a 0x3d 0xb5 0xf2 0x05 0xe1 0xb9 0x9f 0xcd
0x86 0x60 0x86 0x3a 0x15 0x9a 0xd4 0xab 0xe4 0x0f
0xa4 0x89 0x34 0x16 0x3d 0xdd 0xe5 0x42 0xa6 0x58
0x55 0x40 0xfd 0x68 0x3c 0xbf 0xd8 0xc0 0x0f 0x12
0x12 0x9a 0x28 0x4d 0xea 0xcc 0x4c 0xde 0xfe 0x58
0xbe 0x71 0x37 0x54 0x1c 0x04 0x71 0x26 0xc8 0xd4
0x9e 0x27 0x55 0xab 0x18 0x1a 0xb7 0xe9 0x40 0xb0
0xc0
```

--

We have the right to defend ourselves and our
property, because of the kind of animals that we James A. Donald
are. True law derives from this right, not from
the arbitrary power of the omnipotent state. jamesd@netcom.com

--
.....
Timothy C. May | Crypto Anarchy: encryption, digital money,
tcmay@netcom.com | anonymous networks, digital pseudonyms, zero
408-688-5409 | knowledge, reputations, information markets,
W.A.S.T.E.: Aptos, CA | black markets, collapse of governments.
Higher Power: 2^859433 | Public Key: PGP and MailSafe available.
"National borders are just speed bumps on the information
superhighway."

--
.....
Timothy C. May | Crypto Anarchy: encryption, digital money,
tcmay@netcom.com | anonymous networks, digital pseudonyms, zero
408-688-5409 | knowledge, reputations, information markets,
W.A.S.T.E.: Aptos, CA | black markets, collapse of governments.
Higher Power: 2^859433 | Public Key: PGP and MailSafe available.
"National borders are just speed bumps on the information
superhighway."

Apêndice D

Do RC5 ao RC6 em seis passos

O projeto do RC6 poderia ser visto de forma progressiva a partir do RC5, através dos passos descritos a seguir [RIV 98a].

Em cada um dos figuras mostradas a seguir (D.1 até D.6), a parte hachurada destaca a modificação feita no passo que está sendo apresentado, em relação ao passo anterior:

Passo 1 O processo é iniciado com a meia-rodada básica do RC5, conforme ilustra a figura D.1.

```
for i = 1 to r do {
   A = ((A ⊕ B) ⋘ B) + S[i]
   (A,B)=(B,A)
}
```

Figura D.1: Do RC5 ao RC6: Passo 1

Passo 2 São executadas duas rodadas do RC5 em paralelo: uma com os registradores A e B e outra com os registradores C e D, conforme ilustra a figura D.2.

Para atender aos requisitos do AES, um cifrador de bloco precisaria manipular blocos de entrada e saída de 128 *bits*. O RC5 é um cifrador de bloco excepcionalmente rápido, porém modificá-lo para trabalhar sobre blocos de 128 *bits* da forma mais natural resultaria no uso de dois registradores de trabalho de 64 *bits*. As arquiteturas de hardwares existentes e as linguagens de programação para o AES, de forma geral, ainda não suportam operações de 64 *bits* de uma forma eficiente e limpa. Desta forma, o projeto foi modificado para utilizar 4 registradores de 32 *bits*, ao invés de 2 registradores de 64 *bits*. Como conseqüência, tem-se a vantagem de fazer duas rotações por rodada, ao invés de apenas uma, como

encontrada em uma meia-rodada do RC5, e também mais *bits* de dados são utilizados para determinar a quantidade de rotações em cada rodada.

```
for i = 1 to r do {
   A = ((A ⊕ B) ⋘ B) + S[2i]
   C = ((C ⊕ D) ⋘ D) + S[2i+1]
   (A,B)=(B,A)
   (C,D) = (D,C)
}
```

Figura D.2: Do RC5 ao RC6: Passo 2

Passo 3 Na etapa de troca dos registradores, ao invés de serem trocados A com B e C com D, é executada uma permutação dos registradores, a saber, (A,B,C,D) = (B,C,D,A), de forma que a computação feita entre A e B se misture com a computação feita entre C e D, conforme ilustra a figura D.3.

```
for i = 1 to r do {
   A = ((A ⊕ B) ⋘ B) + S[2i]
   C = ((C ⊕ D) ⋘ D) + S[2i+1]
   (A,B,C,D) = (B,C,D,A)
}
```

Figura D.3: Do RC5 ao RC6: Passo 3

Passo 4 São misturadas a computação de AB com CD mais um pouco ainda, trocando a origem das quantidades de rotação entre as duas computações, conforme ilustra a figura D.4.

```
for i = 1 to r do {
   A = ((A ⊕ B) ⋘ D) + S[2i]
   C = ((C ⊕ D) ⋘ B) + S[2i+1]
   (A,B,C,D) = (B,C,D,A)
}
```

Figura D.4: Do RC5 ao RC6: Passo 4

Passo 5 Ao invés de ser utilizados B e D diretamente, versões transformadas destes registradores são utilizadas, conforme ilustra figura D.5. Aqui, o objetivo de segurança é que a quantidade de rotações dependente dos dados, que será derivada da saída desta transformação, deveria depender de todos os *bits* da palavra de entrada. Além disto, a transformação deveria fornecer uma boa mistura dentro da palavra. A escolha particular desta transformação para o RC6 é a função $f(x) = x \times (2x + 1)(\text{mod } 2^w)$, seguida de uma rotação à esquerda de 5 *bits*. Esta transformação parece ir ao encontro dos objetivos de segurança, enquanto toma vantagem de primitivas simples que são implementadas eficientemente na maioria dos modernos processadores, como é o caso das operações de adição, multiplicação e deslocamento circular de *bits*. É importante notar que $f(x)$ é um-para-um módulo 2^w, e que os *bits* de mais alta ordem de $f(x)$, que determinam a quantidade de rotações utilizadas, dependem fortemente de todos os *bits* de x.

```
for i = 1 to r do {
   t = (B x (2B + 1)) ⋘ 5
   A = ((A ⊕ t) ⋘ u) + S[2i]
   u = (D x (2D + 1)) ⋘ 5
   C = ((C ⊕ u) ⋘ t) + S[2i+1]
   (A,B,C,D) = (B,C,D,A)
}
```

Figura D.5: Do RC5 ao RC6: Passo 5

Passo 6 No início e no fim das r rodadas, etapas de pré e pós processamento são acrescentadas. Sem estas etapas, o texto aberto revela parte da entrada para a primeira rodada de cifração e o texto cifrado revela parte da entrada para a última rodada de cifração. As etapas de pré e pós processamento ajudam a disfarçar esta característica e deixam o RC6 com sua aparência final, conforme ilustra a figura D.6.

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do {
   t = (B x (2B + 1)) ⋘ 5
   u = (D x (2D + 1)) ⋘ 5
   A = ((A ⊕ t) ⋘ u) + S[2i]
   C = ((C ⊕ u) ⋘ t) + S[2i+1]
   (A,B,C,D) = (B,C,D,A)
}
A = A + S[2r+2]
C = C + S[2r+3]
```

Figura D.6: Do RC5 ao RC6: Passo 6

No desenvolvimento mostrado anteriormente, a decisão de expandir para quatro registradores de 32 *bits* foi feita em primeiro lugar, por questões de desempenho. Posteriormente, foi feita a decisão de usar a função quadrática $f(x) = x \times (2x + 1)(\text{mod } 2^w)$. Se fosse mantida a decisão de utilizar apenas dois registradores, o esquema de cifração do RC6 seria o apresentado na figura D.7.

```
B = B + S[0]
for i = 1 to r do {
   t = (B x (2B + 1)) ⋘ 5
   A = ((A ⊕ t) ⋘ t) + S[i]
   (A,B) = (B,A)
}
A = A + S[r+1]
```

Figura D.7: Esquema de cifração do RC6 utilizando 2 registradores

Uma vez que o suporte em C para aritmética de 64 *bits* melhore, esta variante do RC6 pode vir a ser adotada, sendo, desta forma, importante o conhecimento de sua existência.

Apêndice E

Implementação do cifrador RC6 em Linguagem C

Este anexo consiste num exemplo de implementação do cifrador RC6 em linguagem de programação C.

```
/* rc6 (TM)
 * Unoptimized sample implementation of Ron Rivest's submission to the
 * AES bakeoff.
 *
 * Salvo Salasio, 19 June 1998
 *
 * Intellectual property notes:  The name of the algorithm (RC6) is
 * trademarked; any property rights to the algorithm or the trademark
 * should be discussed with discussed with the authors of the defining
 * paper "The RC6(TM) Block Cipher": Ronald L. Rivest (MIT),
 * M.J.B. Robshaw (RSA Labs), R. Sidney (RSA Labs), and Y.L. Yin
 * (RSA Labs),
 * distributed 18 June 1998 and available from the lead author's
 * web site.
 *
 * This sample implementation is placed in the public domain by
 * the author,
 * Salvo Salasio.  The ROTL and ROTR definitions were cribbed from
 * RSA Labs'
 * RC5 reference implementation.
 */

#include <stdio.h>

/* RC6 is parameterized for w-bit words, b bytes of key, and
 * r rounds.  The AES version of RC6 specifies b=16, 24, or 32;
 * w=32; and r=20.
 */

#define w 32 /* word size in bits */
#define r 20 /* based on security estimates */

#define P32 0xB7E15163 /* Magic constants for key setup */
#define Q32 0x9E3779B9

/* derived constants */
#define bytes   (w / 8)    /* bytes per word */
#define c       ((b + bytes - 1) / bytes) /* key in words, rounded up */
#define R24     (2 * r + 4)
#define lgw     5                         /* log2(w) -- wussed out */

/* Rotations */
```

```
#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))
unsigned int S[R24 - 1]; /* Key schedule */
void rc6_key_setup(unsigned char *K, int b)
{
int i, j, s, v;
unsigned int L[(32 + bytes - 1) / bytes]; /* Big enough for max b */
unsigned int A, B;
L[c - 1] = 0;
for (i = b - 1; i >= 0; i--)
L[i / bytes] = (L[i / bytes] << 8) + K[i];
S[0] = P32;
for (i = 1; i <= 2 * r + 3; i++)
S[i] = S[i - 1] + Q32;
A = B = i = j = 0;
v = R24;
if (c > v) v = c;
v *= 3;
for (s = 1; s <= v; s++)
{
A = S[i] = ROTL(S[i] + A + B, 3);
B = L[j] = ROTL(L[j] + A + B, A + B);
i = (i + 1) % R24;
j = (j + 1) % c;
}
}
void rc6_block_encrypt(unsigned int *pt, unsigned int *ct)
{
unsigned int A, B, C, D, t, u, x;
int i, j;
A = pt[0];
B = pt[1];
C = pt[2];
D = pt[3];
B += S[0];
D += S[1];
for (i = 2; i <= 2 * r; i += 2)
{
t = ROTL(B * (2 * B + 1), lgw);
u = ROTL(D * (2 * D + 1), lgw);
A = ROTL(A ^ t, u) + S[i];
C = ROTL(C ^ u, t) + S[i + 1];
x = A;
A = B;
B = C;
C = D;
D = x;
}
A += S[2 * r + 2];
C += S[2 * r + 3];
ct[0] = A;
ct[1] = B;
ct[2] = C;
ct[3] = D;
}
void rc6_block_decrypt(unsigned int *ct, unsigned int *pt)
{
unsigned int A, B, C, D, t, u, x;
int i, j;
A = ct[0];
B = ct[1];
C = ct[2];
D = ct[3];
C -= S[2 * r + 3];
A -= S[2 * r + 2];
for (i = 2 * r; i >= 2; i -= 2)
{
x = D;
```

```
D = C;
C = B;
B = A;
A = x;
u = ROTL(D * (2 * D + 1), lgw);
t = ROTL(B * (2 * B + 1), lgw);
C = ROTR(C - S[i + 1], t) ^ u;
A = ROTR(A - S[i], u) ^ t;
}
D -= S[1];
B -= S[0];
pt[0] = A;
pt[1] = B;
pt[2] = C;
pt[3] = D;
}
struct test_struct
{
int keylen;
unsigned char key[32];
unsigned int pt[4];
unsigned int ct[4];
} tests[] =
{
{ 16,   {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00000000, 0x00000000, 0x00000000, 0x00000000},
{0x36a5c38f, 0x78f7b156, 0x4edf29c1, 0x1ea44898},
},
{ 16,   {0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
 0x01, 0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78},
{0x35241302, 0x79685746, 0xbdac9b8a, 0xf1e0dfce},
{0x2f194e52, 0x23c61547, 0x36f6511f, 0x183fa47e},
},
{ 24,   {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00000000, 0x00000000, 0x00000000, 0x00000000},
{0xcb1bd66c, 0x38300b19, 0x163f8a4e, 0x82ae9086},
},
{ 24,   {0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
 0x01, 0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78,
 0x89, 0x9a, 0xab, 0xbc, 0xcd, 0xde, 0xef, 0xf0},
{0x35241302, 0x79685746, 0xbdac9b8a, 0xf1e0dfce},
{0xd0298368, 0x0405e519, 0x2ae9521e, 0xd49152f9},
},
{ 32,   {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
{0x00000000, 0x00000000, 0x00000000, 0x00000000},
{0x05bd5f8f, 0xa85fd110, 0xda3ffa93, 0xc27e856e},
},
{ 32,   {0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
 0x01, 0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78,
 0x89, 0x9a, 0xab, 0xbc, 0xcd, 0xde, 0xef, 0xf0,
 0x10, 0x32, 0x54, 0x76, 0x98, 0xba, 0xdc, 0xfe},
{0x35241302, 0x79685746, 0xbdac9b8a, 0xf1e0dfce},
{0x161824c8, 0x89e4d7f0, 0xa116ad20, 0x485d4e67},
},
{       0,
}
};
int
main()
{
unsigned int ct[4], pt[4];
int i;
struct test_struct *p;
for (p = tests, i = 1; p->keylen; p++, i++)
{
```

```
rc6_key_setup(p->key, p->keylen);

rc6_block_encrypt(p->pt, ct);
printf("Test   %d:  %08x %08x %08x %08x\n",
i, ct[0], ct[1], ct[2], ct[3]);
printf("Should be: %08x %08x %08x %08x\n",
p->ct[0], p->ct[1], p->ct[2], p->ct[3]);
rc6_block_decrypt(ct, pt);
printf("Plain:     %08x %08x %08x %08x\n",
pt[0], pt[1], pt[2], pt[3]);
printf("Should be: %08x %08x %08x %08x\n\n",
p->pt[0], p->pt[1], p->pt[2], p->pt[3]);
}

return 0;
}
```

Apêndice F

Implementação do S-RC6 em Linguagem C

Este anexo consiste na implementação do modelo simplificado do RC6, o SRC-6, em linguagem de programação C. Este programa recebe um texto aberto na forma de 4 valores inteiros entre 0 e 3. A chave também é informada, novamente na forma de dois valores inteiros entre 0 e 3. Após estes valores serem lidos, são executadas as rotinas de expansão da chave, de cifração e de decifração. A cada processamento os valores são impressos na tela, de forma a facilitar o acompanhamento do algoritmo. Para se conseguir os mesmos valores encontrados na descrição do SRC-6, na seção 6.4.4, informe 2 2 2 2 como o texto aberto e 2 2 como a chave.

src6.cpp

```
/*
 * Expansao da chave
 */
#include <stdio.h>
#define w 2
#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))
void expande_chave();
void cifra();
void decifra();
unsigned int S[4]={2,0,2,0};
unsigned int L[4]={2,2,2,2};
unsigned int A, B ;
int main()
{
printf ("\nAlgoritmo de Cifração RC6 Simplificado");
printf ("\n Chave[4 valores entre 0 e 3 separados por espaços]: ");
scanf("%u %u %u %u",&L[0],&L[1],&L[2],&L[3]);
expande_chave();
```

```
getchar();
printf ("\n Texto aberto[2 valores entre 0 e 3 \
separados por espaços]: ");
scanf("%u %u",&A,&B);
cifra();
decifra();
}
void expande_chave()
{
unsigned int j,i,A,B;
A = B = i = j = 0;
printf ("\n%2s %2s %2s %2s %2s %2s %2s %2s %2s %2s %2s %2s",\
"j","i","S0","S1","S2","S3","L0","L1","L2","L3","A","B");
printf ("\n%2u %2u %2u %2u %2u %2u %2u %2u %2u %2u %2u %2u",\
j,i,S[0],S[1],S[2],S[3],L[0],L[1],L[2],L[3],A,B);
for (j = 1; j <= 12; j++)
{
A = S[i] = ROTL(((S[i] + A + B)&3), 1)&3;
B = L[i] = ROTL(((L[i] + A + B)&3), A + B)&3;
i= (i + 1)&3 ;
printf ("\n%2u %2u %2u %2u %2u %2u %2u %2u %2u %2u %2u %2u",\
j,i,S[0],S[1],S[2],S[3],L[0],L[1],L[2],L[3],A,B);
}
printf ("\n\n");
}
void cifra()
{
unsigned int t=0,x,i=1;
printf ("\n%2s %2s %2s %2s","i","A","B","t");
printf ("\n%2u %2u %2u %2u",i,A,B,t);
B = (B + S[0])&3;
printf ("\n%2u %2u %2u %2u",i,A,B,t);
for (i=1;i<=2;i++)
{
t = ROTL((B * (2 * B + 1) ), 1) &3;
A = (ROTL(A ^ t, t) + S[i]) &3;
printf ("\n%2u %2u %2u %2u",i,A,B,t);
x = A;  A = B;  B = x;
printf ("\n%2u %2u %2u %2u",i,A,B,t);
}
A = (A + S[3])&3;
printf ("\n%2u %2u %2u %2u",i,A,B,t);
printf ("\n\n");
}
void decifra()
{
unsigned int t=0,x,i=2;
printf ("\n%2s %2s %2s","A","B","t");
printf ("\n%2u %2u %2u",A,B,t);
A = (A - S[3])&3;
printf ("\n%2u %2u %2u",A,B,t);
for (i=2;i>=1;i--)
{
x = A;  A = B;  B = x;
printf ("\n%2u %2u %2u",A,B,t);
t = ROTL(B * (2 * B + 1), 1)&3;
A = (ROTL(A - S[i],t) ^ t)&3;
printf ("\n%2u %2u %2u",A,B,t);
}
B = (B - S[0])&3;
printf ("\n%2u %2u %2u",A,B,t);
printf ("\n\n");
}
```

Apêndice G

Implementação do S-RC6 em Linguagem C Builder

Este anexo consiste na implementação do modelo simplificado do RC6, o SRC-6, em linguagem de programação C Builder. O programa é executado somente na plataforma Windows. Este programa possui uma interface gráfica e permite que se varie os valores de *w*, *r* e *b*.

src6.cpp

```
#include <vcl.h>
#pragma hdrstop

#include "URC6Simplificado.h"
#include "US.cpp"
#include "UL.cpp"
#include "math.h"
#include "UExpansao.cpp"
#include "UEncriptacao.cpp"
#include "string.h"

#pragma package(smart_init)
#pragma resource "*.dfm"

//Variaveis para armazenagem das subchaves
//Numero e(P) e razao aurea(Q)
const unsigned int P32 = 0xb7e15163;
const unsigned int Q32 = 0x9e3779b9;

//variaveis globais
//Chave
unsigned int *L = NULL;
//SubChave Expandida
unsigned int *S = NULL;
//Numero de Passadas
int r;
//Tamanho da Palavra em bits
int w;
//Tamanho de L
int c;
//Tamanho da Chave em bytes
int b;
```

```
//2^n-1 para operacao & bit a bit
unsigned int auxw;
//Funcoes de criptografia
/*
  Funcao de Rotacao a esquerda e direita
  Argumentos
  x = palavra a rotacionar
  y = quantas rotacoes a fazer
  w = tamanho da palavra em bits
  Retorna o valor rotacionado
*/
int ROTL(int x, int y, int w) {
  int pot2 = 1;
  pot2<<=w;
  return (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))%pot2;
}
int ROTR(int x, int y, int w) {
  int pot2 = 1;
  pot2<<=w;
  return (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))%pot2;
}
/*
  Funcao bit_set
  Argumentos
  bits = palavra contendo a sequencia de bits
  pos  = posicao que se deseja atribuir um valor
  state= valor a atribuir (0 ou 1)
  Atribui o valor state ao bit pos
*/
void bit_set(unsigned char *bits, int pos, int state) {
unsigned char mask;
int i;
mask = 0x80;
for (i = 0; i<(pos % 8); i++ )
mask = mask >> 1;
if(state)
bits[pos/8] = bits[pos/8] | mask;
else
bits[pos/8] = bits[pos/8] & (~mask);
return;
}
/*
  Funcao bit_get
  Argumentos
  bits = palavra contendo a sequencia de bits
  pos  = posicao que se deseja ler o valor
  retorna o valor de determinado bit
*/
int bit_get(const unsigned char *bits, int pos) {
unsigned char mask;
int i;
mask = 0x80;
for (i=0;i<(pos%8);i++)
mask = mask >>1;
return (((mask & bits[(int)(pos/8)]) == mask ) ? 1:0);
}
/*
  Funcao que converte um texto em Hexadecimal para seu
  equivalente em Char, Exemplo AA --> 1010 1010
  Argumentos
  char *pcTxtIn (terminado em '\0')
  char *pcTxtOut
*/
void HexaToChar(char *pcTxtIn, char *pcTxtOut) {
  unsigned int iTam = ceil(strlen(pcTxtIn)/2.0);
  unsigned int iAux,val1,val2;
   for(iAux=0;iAux<iTam;iAux++){
    val1 = (pcTxtIn[iAux*2]>='0' && pcTxtIn[iAux*2]<='9')?
                  pcTxtIn[iAux*2]-'0':pcTxtIn[iAux*2]-'A'+10;
    val2 = (pcTxtIn[iAux*2+1]>='0' && pcTxtIn[iAux*2+1]<='9')?
```

```
                    pcTxtIn[iAux*2+1]-'0':pcTxtIn[iAux*2+1]-'A'+10;
    pcTxtOut[iAux] = (val1 << 4) ^ val2;
  }
}
/*
  Funcao que converte um Char em texto hexadecimal
  equivalente em Hexadecimal, Exemplo 1010 1010 --> AA
  Argumentos
  char *pcTxtIn (terminado em '\0') texto em Hexa
  char *pcTxtOut -> variavel que recebe a conversao (destino)
  int iTam -> tamanho em bytes do Texto
*/
void CharToHexa(char *pcTxtIn, char *pcTxtOut, int iTam) {
  unsigned int iAux,val1,val2;
   for(iAux=0;iAux<iTam;iAux++){
    pcTxtOut[iAux*2]   = ((pcTxtIn[iAux] >> 4 )>=0 &&
(pcTxtIn[iAux] >> 4) <=9)?
                    (pcTxtIn[iAux] >> 4)+'0':(pcTxtIn[iAux]>>4)+'A'-10;
    pcTxtOut[iAux*2+1] = ((pcTxtIn[iAux] & 15)>=0 &&
(pcTxtIn[iAux] &15) <=9)?
                    (pcTxtIn[iAux]&15)+'0':(pcTxtIn[iAux]&15)+'A'-10;
  }
}
/*
  Funcao Inicializa Sub_Chave
  Argumentos
  r = numero de passadas
  w = tamanho da palavra em bits
*/
void Inicializa_SubChave(int r,int w) {
  if(S) delete S;
  S = new unsigned int[r+2];
  S[0] = P32 >> (sizeof(unsigned int)*8 - w) ;
  for(int i=1;i<r+2;i++)
    S[i] = (S[i-1] + ( Q32 >> (sizeof(unsigned int)*8 - w) )) & auxw;
}
/*Funcao Carrega_Chave
  Argumentos
  key = Ponteiro para a chave em caracter
  b   = Tamanho da chave em bytes
  w   = tamanho da palavra
  Retorno
  Tamanho do vetor L
*/
void Carrega_Chave(char *key, int b, int w) {
  int i,j;
  //calcula o Tamanho de L
  c = int(ceil(b*8.0/float(w)));
  int PB,P,TI=sizeof(unsigned int);
  unsigned char aux[4];
  if(L) delete L;
  L = new unsigned int[c];
  for(i=0;i<c;i++) {
    L[i] = 0;
    memcpy(aux,&L[i],sizeof(unsigned int));
    for(j=0;j<w;j++) {
      PB = TI*8 - w + j;
      P = (TI-PB/8-1)*8+PB%8;
      bit_set(aux, P, bit_get(key,i*w+j));
    }
    memcpy(&L[i],aux,sizeof(unsigned int));
  }
}
void Mostra_SubChave()
{
  int i;
  frmSubChave->strgSubChave->RowCount = r+3;
  frmSubChave->strgSubChave->Cells[1][0] = "S";
  frmSubChave->Height = frmSubChave->strgSubChave->
DefaultRowHeight*(r+3)+
                        frmSubChave->strgSubChave->
```

```
GridLineWidth*(r+4) + 29;
  for(i=1;i<=r+2;i++) {
    frmSubChave->strgSubChave->Cells[0][i]=IntToStr(i-1);
    frmSubChave->strgSubChave->Cells[1][i]=IntToStr(S[i-1]);
  }
  frmSubChave->Show();
}
void Mostra_L()
{
  int i;
  frmL->strgL->RowCount = c+1;  //Devido a linha de cabecalho
  frmL->strgL->Cells[1][0] = "L";
  frmL->Height = frmL->strgL->DefaultRowHeight*(c+1)+
                        frmL->strgL->GridLineWidth*(c+2) + 29;
  for(i=1;i<=c;i++) {
    frmL->strgL->Cells[0][i]=IntToStr(i-1);
    frmL->strgL->Cells[1][i]=IntToStr(L[i-1]);
  }
  frmL->Show();
}
void Mostra_Expansao(unsigned int i, unsigned int j,
unsigned int A, unsigned int B)
{
  int iAux,linha;
  frmExpansao->strgExpansao->RowCount =
frmExpansao->strgExpansao->RowCount+1;
  linha = frmExpansao->strgExpansao->RowCount-1;
  frmExpansao->Height = frmExpansao->strgExpansao->
DefaultRowHeight*(linha+2)+
                        frmExpansao->strgExpansao->
GridLineWidth*(linha+3) + 29;
  //Cabecalho da linha
  frmExpansao->strgExpansao->Cells[0][linha]=
StrToInt(linha-1);
  //Valores de i,j,A,B
  frmExpansao->strgExpansao->Cells[1][linha]=IntToStr(i);
  frmExpansao->strgExpansao->Cells[2][linha]=IntToStr(j);
  frmExpansao->strgExpansao->Cells[3][linha]=IntToStr(A);
  frmExpansao->strgExpansao->Cells[4][linha]=IntToStr(B);
  //Carrega o vetor S de tamanho r+2
  for(iAux=0;iAux<r+2;iAux++) {
    frmExpansao->strgExpansao->Cells[5+iAux][linha]=
IntToStr(S[iAux]);
  }
  //Carrega o vetor L de tamanho c
  for(iAux=0;iAux<c;iAux++) {
    frmExpansao->strgExpansao->Cells[5+r+2+iAux][linha]=
IntToStr(L[iAux]);
  }
  frmExpansao->Show();
}
void Mostra_Encriptacao(unsigned int i, unsigned int A,
unsigned int B, unsigned int t)
{
  int iAux,linha;
  frmEncriptacao->strgEncriptacao->RowCount =
frmEncriptacao->strgEncriptacao->RowCount+1;
  linha = frmEncriptacao->strgEncriptacao->RowCount-1;
  frmEncriptacao->Height =
frmEncriptacao->strgEncriptacao->
DefaultRowHeight*(linha+2)+
        frmEncriptacao->strgEncriptacao->
GridLineWidth*(linha+3) + 29;
  //Valores de i,A,B,t
  frmEncriptacao->strgEncriptacao->Cells[0][linha]=IntToStr(i);
  frmEncriptacao->strgEncriptacao->Cells[1][linha]=IntToStr(A);
  frmEncriptacao->strgEncriptacao->Cells[2][linha]=IntToStr(B);
  frmEncriptacao->strgEncriptacao->Cells[3][linha]=IntToStr(t);
  frmEncriptacao->Show();
}
```

```
void expande_chave()
{
unsigned int j,i,A,B;
    int iNumPassadas = 3*(r+2)<c? c : 3*(r+2);
    A = B = i = j = 0;
    Mostra_Expansao(i,j,A,B);
for (j = 1; j <= iNumPassadas; j++)
    {
A = S[i%(r+2)] = ROTL((S[i%(r+2)] + A + B)& auxw, 3,w) & auxw;
B = L[i%c] = ROTL((L[i%c] + A + B)& auxw, A + B,w) & auxw;
i= (i + 1);
        Mostra_Expansao(i,j,A,B);
}
}
void cifra(unsigned char *pcBlock)
{
unsigned int t=0,x,i=0,A,B,iNumRot;
    A = pcBlock[0];
    B = pcBlock[1];
Mostra_Encriptacao(i,A,B,t);
B = (B + S[0]) & auxw;
    iNumRot = ceil(log(w)/log(2));
for (i=1;i<=r;i++)
{
t = ROTL(B*(2*B+1),iNumRot,w)&auxw;
A = ( ROTL(A ^ t, t ,w) + S[i] ) & auxw;
        Mostra_Encriptacao(i,A,B,t);
x = A;  A = B;  B = x;
        Mostra_Encriptacao(i,A,B,t);
}
A = (A + S[i]) & auxw;
    Mostra_Encriptacao(i,A,B,t);
    pcBlock[0]=A;
    pcBlock[1]=B;
}
void decifra(unsigned char *pcBlock)
{
unsigned int t=0,x,i=r+1,A,B,iNumRot;
    A = pcBlock[0];
    B = pcBlock[1];
Mostra_Encriptacao(i,A,B,t);
A = (A - S[i]) & auxw;
    iNumRot = ceil(log(w)/log(2));
for (i=r;i>0;i--)
{
        x = A;  A = B;  B = x;
t = ROTL(B*(2*B+1),iNumRot,w)&auxw;
A = (ROTR(A - S[i] , t ,w) ^ t )& auxw;
        Mostra_Encriptacao(i,A,B,t);
}
B = (B - S[0]) & auxw;
    Mostra_Encriptacao(i,A,B,t);
    pcBlock[0]=A;
    pcBlock[1]=B;
}


TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}


void __fastcall TForm1::btCriptografarDepClick(TObject *Sender)
{
  int iAux;
  unsigned char *pcKey,*pcPlain;
  unsigned char *pcTempKey,*pcTempPlain;
  unsigned char val1,val2;
  r = StrToInt(edNumRod->Text);
```

```
  w = StrToInt(edTamPal->Text);
  b = StrToInt(edTamChave->Text);
  auxw = (1 << w) -1;
  Inicializa_SubChave(r,w);
  //Mostra_SubChave();
  pcKey = new unsigned char[b];
  pcTempKey = edChave->Text.c_str();
  HexaToChar(pcTempKey,pcKey);
  Carrega_Chave(pcKey,b,w);
  //Mostra_L();
  //configura tamanho da tabela de saida
  frmExpansao->strgExpansao->ColCount = 4 + r + 2 + c + 1;
  frmExpansao->strgExpansao->RowCount = 2;
  frmExpansao->Width = frmExpansao->strgExpansao->
DefaultColWidth * (4+r+2+c+2)+
  frmExpansao->strgExpansao->GridLineWidth * (4+r+2+c+3);
  //Monta Cabecalho
  //Valores de i,j,A,B
  frmExpansao->strgExpansao->Cells[1][0]="i";
  frmExpansao->strgExpansao->Cells[2][0]="j";
  frmExpansao->strgExpansao->Cells[3][0]="A";
  frmExpansao->strgExpansao->Cells[4][0]="B";
  for(iAux=0;iAux<r+2;iAux++) {
    frmExpansao->strgExpansao->Cells[5+iAux][0]=
"S["+IntToStr(iAux)+"]";
  }
  //Carrega o vetor L de tamanho c
  for(iAux=0;iAux<c;iAux++) {
    frmExpansao->strgExpansao->Cells[5+r+2+iAux][0]=
"L["+IntToStr(iAux)+"]";
  }
  expande_chave();
  //configura tamanho da tabela de saida
  frmEncriptacao->strgEncriptacao->RowCount = 1;
  frmEncriptacao->Width =
  frmEncriptacao->strgEncriptacao->DefaultColWidth * 4+
  frmEncriptacao->strgEncriptacao->GridLineWidth * 4;
  //Monta Cabecalho
  //Valores de i,j,A,B
  frmEncriptacao->strgEncriptacao->Cells[0][0]="i";
  frmEncriptacao->strgEncriptacao->Cells[1][0]="A";
  frmEncriptacao->strgEncriptacao->Cells[2][0]="B";
  frmEncriptacao->strgEncriptacao->Cells[3][0]="t";
  pcPlain = new unsigned char[b];
  if( Sender == btCriptografarDep)
    pcTempPlain = edTextoPlano->Text.c_str();
  else
    pcTempPlain = edTextoCifrado->Text.c_str();
  HexaToChar(pcTempPlain,pcPlain);
  if( Sender == btCriptografarDep) {
    cifra(pcPlain);
    CharToHexa(pcPlain, pcTempPlain,edTextoPlano->
Text.Length()/2);
    edTextoCifrado->Text =
    AnsiString((char *) pcTempPlain,edTextoPlano->
Text.Length());
  } else {
    decifra(pcPlain);
    CharToHexa(pcPlain, pcTempPlain,edTextoCifrado->
Text.Length()/2);
    edTextoPlano->Text =
       AnsiString((char *) pcTempPlain,edTextoCifrado->
Text.Length());;
  }
}

void __fastcall TForm1::Sair1Click(TObject *Sender)
{
  Close();
}

void __fastcall TForm1::edTamPalExit(TObject *Sender)
{
```

```
  edTamTextoPlano->Text=IntToStr(ceil(StrToInt(edTamPal->
Text)/2.0));
  edTextoPlano->MaxLength = 4 ; //StrToInt(edTamTextoPlano->
Text);
}
```