CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FACULDADE DE INFORMÁTICA DE MARÍLIA
BACHARELANDO EM CIÊNCIAS DA COMPUTAÇÃO

IMPLEMENTAÇÃO E DESEMPENHO DO ALGORITMO CRIPTOGRÁFICO MARS

LARISSA BILAR

MARÍLIA, SP
2005

LARISSA BILAR

IMPLEMENTAÇÃO E DESEMPENHO DO ALGORITMO CRIPTOGRÁFICO MARS

Trabalho apresentado como requisitos para conclusão do curso de Bacharelado em Ciências da Computação, realizada no Centro Universitário Eurípides de Marília – UNIVEM.
Orientador:
Prof. Dr. Edward David Moreno Ordonez.

MARÍLIA, SP

2005

BILAR, Larissa

Implementação e desempenho do algoritmo criptográfico Mars/ Larissa Bilar; orientador: Edward David Moreno Ordonez. Marilia, SP: [s/n], 2005

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marilia – Fundação de Ensino Eurípides Soares da Rocha

1. Criptografia 2.Algoritmos Simétricos 3.Mars
4. Desempenho 5.Linguagens C e Java
CDD: 005.1

LARISSA BILAR

IMPLEMENTAÇÃO E DESEMPENHO DO ALGORITMO CRIPTOGRÁFICO MARS

Banca examinadora do trabalho de conclusão de curso apresentado à UNIVEM/F.E.E.S.R., para a obtenção do titulo de Bacharelado em Ciência da Computação. Área de Concentração: Arquitetura de sistemas computacionais.

Resultado: _____

ORIENTADOR: Prof. Dr. Edward David Moreno Ordonez.

1º EXAMINADOR: _____

2º EXAMINADOR: _____

Marilia, ____ de _____ de 2005

DEDICATÓRIA

Dedico este trabalho à minha avó que sempre cuidou de mim, me ensinou tudo que sei, ao meu pai que acreditou em mim, sempre me incentivou e ao meu irmão que sempre esteve ao meu lado me ajudando e sendo meu grande amigo.

AGRADECIMENTOS

A Deus pela vida que tenho e por tudo que conquistei ao longo dela.

A minha família, pelo apoio, incentivo e compreensão em todos momentos.

Ao professor Dr Edward David Moreno Ordonez, pela competente orientação e apoio.

A todas minhas amigas que sempre estiveram ao meu lado em momentos bons e ruins.

Ao meu amigo, Eduardo José, pela sincera amizade durante os quatros anos na faculdade.

Ao meu amigo, companheiro, uma pessoa muito importante na minha vida, que nunca será esquecido, Fernando Lazanha.

A todas as outras pessoas que, mesmo não citadas aqui, me deram muito apoio e incentivo, que, sem os quais, essa pesquisa não chegaria a essa conclusão.

BILAR, Larissa e. Implementação e desempenho do algoritmo criptográfico Mars. 2005. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marilia – Fundação de Ensino Eurípides Soares da Rocha, 2005.

RESUMO

Este trabalho realiza uma análise sobre criptografia e alguns de seus algoritmos. Com ênfase no algoritmo Mars. O algoritmo Mars foi implementado na linguagem C, e foram feitos alguns testes em computadores diferentes para estudar seu desempenho. Logo após foi implementado o mesmo algoritmo em Java, e feitos novos testes para também verificar seu desempenho. Ao fim, foi feita uma comparação sobre as linguagens e seus desempenhos, em relação a esse algoritmo.

Palavras Chave: Criptografia; Algoritmos Simétricos; Mars; Desempenho; Linguagem C e Java.

BILAR, Larissa e. **Implementação e desempenho do algoritmo criptográfico Mars.** 2005. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marilia – Fundação de Ensino Eurípides Soares da Rocha, 2005.

ABSTRACT

This work accomplishes an analysis about cryptography and some of its algorithms. With emphasis in the Mars algorithm. The Mars algorithm was implemented in the C language, and they were made some tests in different computers to study its acting. Therefore the some algorithm was implemented in Java, and made news tests for also to verify its acting. To the end, its was made a comparison about the languages and its acting, in relation to the algorithm.

Keywords: Cryptography; Symmetric Algorithm; Mars; Acting; Language C e Java.

LISTA DE ILUSTRAÇÕES

.

TABELA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DES: Data Encryption Standard

IDEA: International Data Encryption Algorithm

AES: Advanced Encryption Standard

NBS: National Bureau of Standard

NSA: National Security Agency

NIST: National Institute Standard and Technology

ACM: Association for Computing Machinery

PGP: Pretty Good Privacy

API: Application Programming Interfaces

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Considerações Iniciais

Com o avanço cada vez maior das Redes de Computadores, hoje muitas transações comerciais e muitas informações importantes são diariamente transmitidas pela rede. Porém, esta “comodidade” trouxe a preocupação de que informações confidenciais podem estar expostas a intrusos e atacantes de redes que utilizam meios cada vez mais sofisticados para violar a privacidade e integridade dos dados. Uma forma de manter a confidencialidade desses dados é através da criptografia.

A escrita desconhecida, ou melhor, a criptografia, é uma ciência que estuda algoritmos criptográficos capazes de manipular dados e codificar seu conteúdo, onde os mesmos podem ser transmitidos e armazenados sem que haja alterações ou a sua exposição à entidade não autorizada. Isso fornece confidencialidade, não repúdio, identificação e verificação de integridade à mensagem criptografada (Pereira, F.D., 2005).

Em outras palavras, técnicas de criptografia podem ser usadas como um meio efetivo de proteção de informações suscetíveis a ataques, estejam elas armazenadas em um computador ou sendo transmitidas pela rede**.**

Essa é uma técnica usada desde a história antiga, quando o imperador romano Júlio César fazia pequenas substituições de letras em suas mensagens de guerra para que os inimigos não descobrissem seus planos. Com o passar do tempo e com o surgimento dos computadores a criptografia foi mudando muito em relação a sua capacidade, dando assim maior segurança às mensagens.

Existem dois tipos básicos de Sistemas Criptográficos, o chamado Simétrico ou de chave única e o chamado Assimétrico ou de chave pública.

No sistema de criptografia Simétrico existe apenas uma chave secreta que é utilizada tanto para cifrar um dado quanto para decifrar. Nesse sistema deve haver uma chave para cada par de usuários, essas chaves devem ser sempre trocadas por algum canal seguro.

Já no sistema Assimétrico existem duas chaves, uma pública e uma privada, cada usuário deve possuir essas duas chaves, a chave publica é usada para cifrar, e a chave privada para decifrar.

Hoje se pode encontrar vários algoritmos de criptografia, mas existe um modelo que surgiu na década de 70 e se tornou o modelo padrão no mundo, foi o DES (*Data Encryption Standard*), implementado em 1977, e ele pode ser feito tanto em hardware como em software (Pereira,F.D.,2005).

DES (*Data Encryption Standard*) é o algoritmo simétrico mais disseminado no mundo (Fips46,1993).

Com o passar do tempo e o aumento da capacidade computacional, a segurança do DES começou a se comprometer. Foi então que vários outros modelos começaram a ser criados, tais como o:

- IDEA(*Internacional Data Encryption Algorithm*) é utilizado principalmente no mercado financeiro (Pereira,F.D,2005).

- AES(*Advanced Encryption Standard)*(Fips97,2001) é um algoritmo flexível e seguro que utiliza chaves de tamanho variável.

- RC5 foi desenvolvido por Ronald Rivest e publicado em 1994. O RC5 permite que o usuário defina o tamanho da chave, o tamanho do bloco, e o número de iterações do algoritmo de cifragem (Rivest 1995).

Outro modelo existente é o algoritmo Mars que será estudado nesse trabalho.

Pode-se encontrar vários métodos de cifrar dados, desde métodos simples e quase que infantis, até métodos que envolvem teoria matemática e fórmulas complexas.

Os algoritmos criptográficos utilizam, em geral, uma série de combinações, substituições e transposições, e sua eficiência é tão maior quando mais difícil for decifrá-lo sem o conhecimento da chave. Outro fator que dificulta a quebra de um código cifrado é o tamanho da chave. Para chaves de 128 bits por exemplo, temos 2^{128} combinações possíveis. O que torna a busca pela chave correta demasiadamente longa e exaustiva, mesmo para os mais rápidos computadores modernos.

1.2 Justificativa

Esse trabalho vai mostrar como é importante a criptografia hoje em dia, pois a cada dia que passa aumenta ainda mais os ataques e invasões em redes de computador.

Com toda essa evolução sobre os computadores, também evolui o meio de invasão, tornando cada vez mais difícil manter algumas informações em segredo. Uma pessoa que tiver dados violados pode ter grandes prejuízos.

Devido a tudo isso, existem vários meios de criptografia. Cada um com suas características específicas.

1.3 Objetivos

O presente trabalho tem como objetivo o estudo sobre criptografia com ênfase no algoritmo Mars, avaliando as diferenças entre seu desempenho desenvolvido em duas linguagens diferentes: C e Java.

1.4 Organização do trabalho

Esse trabalho foi desenvolvido em cinco capítulos, no primeiro capítulo é feita uma introdução sobre o assunto geral.

Uma explicação mais profunda sobre a criptografia é desenvolvida no segundo capítulo, abordando sua história desde a antiguidade até os dias atuais. Nessa parte do trabalho também são apresentados alguns dos diversos algoritmos de criptografia existentes e uma breve explicação sobre cada um.

O trabalho dá mais ênfase ao algoritmo Mars, portanto, o terceiro capítulo, vai ser reservado apenas para a explicação desse algoritmo, sua implementação na linguagem C e alguns testes em diversas máquinas diferentes para testar seu desempenho.

Logo após, no quarto capítulo, é explicado um pouco sobre a linguagem Java, e implementado o mesmo algoritmo Mars agora na linguagem Java. Também são feitos testes para verificar seu desempenho. Ainda no quarto capítulo é feita uma comparação do desempenho desse algoritmo Mars nas duas linguagens diferentes, C e Java.

Por fim, ao quinto e último capítulo, é feita uma conclusão sobre tudo que foi estudado ao longo desse trabalho.

Capítulo 2

CONCEITOS BÁSICOS DE CRIPTOGRAFIA

2.1 Conceito

A criptografia, palavra oriunda do grego: “Kryptós” (oculto) e “grápheim” (escrever), foi inicialmente conceituada como um conjunto de técnicas ou métodos para transformar uma informação legível em uma informação ilegível. Atualmente, a criptografia é muito mais do que isto, sendo conceituada como o conjunto de princípios e técnicas que visam proporcionar as informações ou dados, armazenados ou em trânsito, os serviços de segurança da confidencialidade, integridade e autenticidade (Burnett, 2002).

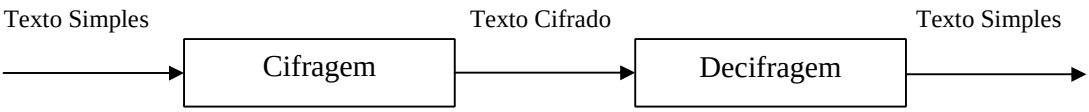

Figura 2.1 Os processos de cifragem e decifragem

Alguns termos específicos serão usados durante este trabalho. Para auxiliar na leitura do mesmo, apresenta-se aqui uma rápida explicação dos seus significados.

Uma mensagem é um texto simples. O processo de tornar o conteúdo de uma mensagem irreconhecível é chamado de encriptação ou cifragem. O processo contrário, ou seja, retornar uma mensagem em texto comum novamente é chamado de decritação ou decifragem, conforme se mostra na figura 2.1.

A técnica de manter mensagens seguras é chamada de criptografia. A técnica ou a arte de tentar descobrir o conteúdo de mensagens cifradas é chamada de criptoanálise, e seus praticantes de criptoanalistas ou atacantes. Assim o conjunto destas duas técnicas é chamado de criptologia.

O algoritmo de criptografia é uma seqüência de procedimentos que envolvem uma matemática capaz de cifrar e decifrar dados sigilosos. O algoritmo pode ser executado por um computador, por um hardware dedicado e por um ser humano, variando a velocidade da execução e a probabilidade e erros em cada determinada situação.

2.2 História

Há registros de um fato que aconteceu numa vila egípcia perto do rio Nilo chamada Menet Khufu, em 1900 a.c Khnumhotep II era um arquiteto do faraó Amenemhet II, ele construiu alguns monumentos para o faraó, que precisavam ser documentados e mantidos em segredo, tais informações eram escritas em tabletes de argila.

O escriba de Khnumhotep II teve a idéia de substituir algumas palavras ou trechos destes tabletes. Caso o documento fosse roubado, o ladrão não encontraria o caminho que o levaria ao tesouro – morreria de fome, perdido nas catacumbas da pirâmide.

Kahn considera isto como o primeiro exemplo documentado da escrita cifrada (Kahn, 1967).

A criptografia na Mesopotâmia ultrapassou a egípcia, chegando a um nível bastante moderno. O primeiro registro, em 1500 a.c.,está numa fórmula para fazer esmaltes para cerâmica. O tablete que contém a formula tem apenas cerca de 8cm x 5cm e foi achado às margens do rio Tigre. Usava símbolos especiais que podem ter vários significados diferentes (Aldeia, 2005).

Nesta época, mercadores assírios usavam “intaglios”, que são peças planas de pedra com símbolos entalhados para a sua identificação. O moderno comércio com assinaturas digitais estava começando a ser inventado.

Em 600 a 500 a.c, Escribas hebreus, escrevendo o livro de Jeremias, usaram a cifra de substituição simples pelo alfabeto reverso conhecida como ATBASH. As cifras mais

conhecidas da época são o ATBASH, o ALBAM e o ATBAH, as chamadas cifras hebraicas (Aldeia, 2005).

Tucídides conta sobre ordens entregues ao príncipe e general espartano Pasanius em 475 a.c. através do que poderia ser o sistema de criptografia militar mais antigo, o scytale ou bastão de Licurgo. Como um dispositivo para esconder mensagens, o scytale consiste num bastão de madeira ao redor do qual se enrola firmemente uma tira de couro ou pergaminho, longa e estreita. Escreve-se a mensagem no sentido do comprimento do bastão, a tira é desenrolada e contém a mensagem cifrada.

Euclides de Alexandria foi um matemático grego que viveu aproximadamente de 330 a.c. a 270 a.c., Euclides compilou e sistematizou a geometria e a teoria dos números da sua época no famoso texto "Elementos" . Este material tem muita influência nos dias da moderna criptologia por computador.

Erastótenes de Cirene, filósofo e geômetra, viveu de 270 a.c.,conhecido por ser criador de um método para identificar números primos, e por calcular o diâmetro da Terra com surpreendente precisão, também constituiu na criptologia atual, em especial, com métodos para calcular e detectar números primos.

O primeiro relato de um algoritmo de criptografia que se tem é conhecido como algoritmo de César (Wer 95), usado pelo imperador Júlio César na Roma Antiga. Era um algoritmo simples que fazia substituições alfabéticas no texto da mensagem. As substituições aconteciam trocando as letras por outras, três posições a frente no alfabeto, ou seja, a letra A seria substituída por D, a letra B por E, e assim por adiante. A seguir um exemplo do funcionamento de algoritmo de César (Hinz, M.A.M., 2000).

Texto simples: Vamos atacar hoje.

Texto cifrado : Zdprv dxdfxu krmh.

Esse algoritmo desenvolvido pelo imperador romano Júlio César enganava seus inimigos. Entretanto, uma pequena melhora neste algoritmo introduz um conceito que é usado até hoje, o conceito de chave. Se ao invés de se substituir as letras três a três, se substituísse elas por K posições, então independente do algoritmo ser conhecido, a chave se torna mais importante que o algoritmo, pois sem o conhecimento dela não se conseguiria decifrar a mensagem.

È bom salientar que o tamanho da chave também é um fator importantíssimo na segurança do algoritmo, uma chave de 2 dígitos possui 16 (2^4) possibilidades de combinações, uma chave de 6 dígitos possui 4.096 ($2^{2 * 6}$) possibilidades. Conseqüentemente quanto mais possibilidades de combinações, mais tempo um criptoanalistas levará para decifrar um texto.

Em 1901, inicia-se a era da comunicação sem fio. Apesar da vantagem de uma comunicação de longa distância sem o uso de fios ou cabos, o sistema é aberto e aumenta o desafio da criptologia.

Em 1921, Edward Hugh Herbern Electric Code (Tkotz, 2003), uma empresa produtora de máquinas de cifragem eletromecânicas baseadas em rotores que giram a cada caractere cifrado. Logo após entre 1933 e 1945, a máquina Enigma, máquina eletromecânica de encriptação com rotores utilizada tanto para cifrar quanto para decifrar, foi aperfeiçoada até se transformar na ferramenta mais importante na Alemanha nazista. O sistema foi quebrado pelo matemático polonês Marian Rejewshi que se baseou apenas em textos cifrados interceptados e numa lista de chaves obtidas através de um espião (Kahn, 1967). A seguir, outros acontecimentos relacionados à utilização da criptografia (Tkotz, 2003):

- 1943 - Colossus, um computador para quebrar códigos.
- 1969 - James Ellis desenvolve um sistema de chaves públicas e chaves privadas separadas.

- 1976 - Diffie–Hellman é um algoritmo baseado no problema do logaritmo discreto, é o criptosistema de chave pública mais antigo ainda em uso.
- A IBM apresenta o método chamado de Lúcifer ao NBS (*National Bureau of Standards*) o qual, após avaliar o algorítmo com a ajuda da NSA (*National Security Agency*), introduz algumas modificações (como as Caixas "S" e uma chave menor) e adota o método como padrão de encriptação de dados para os EUA (Fipspub – 46,1993), conhecido hoje como DES *(Data Encryption Standard*). Hoje o NSB é chamado de *National Institute Standards and Technology*, NIST.
- 1977 – Ronald L.Rivest, Adi Shamir e Leonard M.Adleman começaram a discutir como criar um sistema de chave publica prático. Ron Rivest acabou tendo uma grande idéia e a submeteu à apreciação dos amigos: era uma cifra de chave pública, tanto para confidencialidade quanto para assinaturas digitais, baseada na dificuldade da fatoração de números primos grandes. Foi batizada de RSA, de acordo com as primeiras letras dos sobrenomes dos autores (Tkotz, 2003).
- 1978 – O algoritmo RSA é publicado na revista ACM (*Association for Computing Machinery*).
- 1990 – Xuejia Lai e James Massey publicam na Suíça o artigo "*A Proposal for a New Block Encryption Standard*" ("Uma proposta para um Novo Padrão de Encriptação de Bloco"), o assim chamado IDEA (*International Data Encryption Algorithm*), para substituir o DES. O algoritmo IDEA utiliza uma chave de 128 bits e emprega operações adequadas para computadores de uso geral, tornando as implementações em software mais eficientes (Schneier, 1993).
- 1991 – Phil Zimmermann torna pública sua primeira versão de PGP(*Pretty Good Privacy*) como resposta ao FBI, o qual invoca o direito de acessar qualquer texto claro da comunicação entre cidadãos. O PGP oferece uma segurança alta para o cidadão

comum e, como tal, pode ser encarado como um concorrente de produtos comerciais como o Mailsafe da RSADSI. Entretanto, o PGP é especialmente notável porque foi disponibilizado como *freeware* e, como resultado, tornou-se um padrão mundial enquanto que seus concorrentes da época continuaram absolutamente desconhecidos (Back, 2003).

- 1994 – O professor Ron Rivest, autor dos algoritmos RC2 e RC4 incluídos na biblioteca de criptografia BSAFE do RSADSI, publica a proposta do algoritmo RC5 na Internet. Este algoritmo usa operação não linear com rotação dependente de dados e o usuário pode variar o tamanho do bloco, o número de estágios e o comprimento da chave por ele ser parametrizado. Uma análise feita pelo RSA Labs, mostrada na CRYPTO'95, sugere que w=32 e r=12 proporcionam uma segurança maior que o DES (Aldeia, 2005).
- O algoritmo Blowfish, uma cifra de bloco de 64 bits com uma chave de até 488 bits de comprimento, é projetada por Bruce Schneier.
- O código DES de 56 bits é quebrado por uma rede de 14.000 computadores. O FGP 5.0 *Freeware* é amplamente distribuído para uso não comercial.
- 1998 – O código DES é quebrado novamente, agora em 56 horas por pesquisadores do Vale do Silício (Pereira, F.D., 2005).
- 1999 – O código DES é quebrado em apenas 22 horas e 15 minutos. Através da união da *Electronic Frontier Foundationn* e a *Distributed.Net*, que reuniram em torno de 100.000 computadores pessoais ao DES Cracker pela Internet.
- 2000 – O NIST (*National Institute of Standards and Technology*) anunciou um novo padrão de chave secreta de cifragem, escolhido de 15 candidatos: Cast-256, Crypton, Deal, DFC, E2, Frog, HPC, Loki97, Margenta, Mars, RC6, Rinjndael, Sofer+, Serpent e Twofish. Este novo padrão pretendia substituir o velho algoritmo DES, cujo

tamanho das chaves tornara-se muito pequeno. O Rijindael – um nome comprimido, originário dos seus inventores Rijimen e Daemen - foi escolhido para se tornar novo padrão do AES, *Advanced Encryption Standard* (Fips97, 2001).

Como se pode notar a criptografia foi ampliando seus horizontes, se tornando cada vez mais indispensável para os sistemas computacionais modernos, e acompanha a evolução da humanidade.

2.3 Criptografia Simétrica

A criptografia simétrica é conhecida como Criptografia Convencional.

A cifragem de uma mensagem baseia-se em dois componentes: um algoritmo e uma chave. Um algoritmo é uma transformação matemática. Ele converte uma mensagem clara em uma mensagem cifrada e vice-versa.

Antigamente, a segurança da cifragem estava baseada somente no sigilo do algoritmo criptográfico, isso facilitava a descoberta da mensagem, caso alguém tomasse conhecimento desse algoritmo. Mas com o surgimento da chave, uma cadeia aleatória de bits utilizada em conjunto com o algoritmo, ficou mais seguro a utilização da criptografia.

Vantagens do uso de chaves (Adriano,J.,2005) :

- permite a utilização do mesmo algoritmo para a comunicação com diferentes receptores, apenas trocando a chave;
- pode-se trocar facilmente a chave no caso de violação. Mantendo o mesmo algoritmo.

Quando se cifra um texto, se usa o algoritmo de cifragem junto com a chave secreta, depois para decifrar esse texto, se usa um outro algoritmo, decifragem, junto com a mesma chave secreta usada anteriormente. Portanto, tanto o emissor do documento quanto o receptor

precisa ter conhecimento sobre a chave, mas é preciso garantir uma forma segura para informá-la.

O poder da cifra é medido pelo tamanho da chave, geralmente as chaves de 40 bits são consideradas fracas e as de 128 bits, as mais fortes**.**

Visto que um bit pode ter apenas dois valores, 0 ou 1, uma chave de três dígitos oferecerá 2^3 = 8 possíveis valores para a chave, sendo esses valores: 000,001,010,011,100,101,110,111.

Do ponto de vista do usuário, as chaves de criptografia são similares às senhas de acesso a bancos e a sistema de acesso a computadores. Usando a senha correta, o usuário tem acesso aos serviços, em caso contrário, o acesso é negado. No caso da criptografia, o uso de chaves relaciona-se com o acesso ou não à informação cifrada. O usuário deve usar a chave correta para poder decifrar as mensagens, conforme se mostra na figura 2.2, o usuário usa a mesma chave para cifrar o texto e para decifrar:

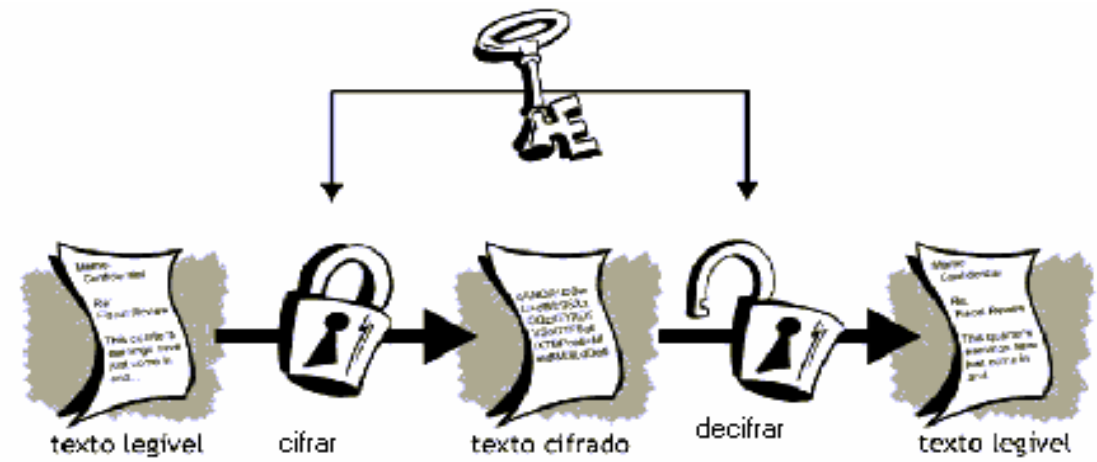

Figura 2.2 Processo de cifrar e decifra usando chave simétrica (Maia, L.P, 2005).

2.3.1 Atacando a chave

Se os invasores puderem descobrir qual é a chave eles podem decifrar os dados. Um método, chamado de ataque de força bruta, consiste em tentar todas as possíveis chaves até que a correta seja identificada. Ele funciona dessa maneira. Suponha que a chave seja um número entre 0 e 100.000.000.000 (cem bilhões). O invasor pega um texto cifrado (talvez apenas um com um valor de 8 ou 16 bytes) e alimenta o algoritmo de criptografia junto com a "alegada chave" de 0. O algoritmo realiza seu trabalho e produz um resultado. Se os dados resultantes parecerem razoáveis, quer dizer que provavelmente é a chave correta. Se for um texto sem sentido, então a suposta chave não é a verdadeira. Nesse caso, tenta-se com o valor 1 em seguida 2, 3, 4 e assim por diante.

Importante lembrar que um algoritmo simplesmente realiza seus passos, independentemente da entrada. Ele não tem nenhuma maneira de saber se o resultado que ele produz é o correto. Mesmo se o valor for próximo da chave, talvez errado em apenas 1, o resultado será um texto sem sentido. Assim, é necessário examinar o resultado para informar se ele pode ser a chave. Invasores inteligentes escrevem programas para examinar o resultado. Verificando se as letras são do alfabeto e juntas fazem um sentido, se for passa-se adiante, senão tenta-se uma nova chave.

Normalmente, leva pouco tempo para tentar uma chave, um invasor provavelmente pode escrever um programa que tente várias chaves por segundo. Por fim, o invasor pode tentar cada número possível entre o zero e 100 bilhões, mas isso talvez não seja necessário. Uma vez que a chave correta for encontrada, não há necessidade de pesquisar ainda mais. Na média, o invasor tentará metade de todas as possíveis chaves — no exemplo anterior, 50 bilhões de chaves — antes de encontrar a correta. As vezes leva mais tempo, às vezes menos, mas em média, aproximadamente metades de todas as possíveis chaves devem ser tentadas (Oliveira, E.E.L. ,2003).

Um invasor para tentar 50 bilhões de chaves levaria muito tempo, talvez três anos, três dias, três minutos. Suponha que se queira manter o segredo seguro por pelo menos três anos, entretanto um invasor leva apenas três minutos para tentar 50 bilhões de valores, então nesse caso deve-se escolher um intervalo maior. Em vez de encontrar um número entre O e 100 bilhões, encontra-se um número entre O e 100 bilhões de bilhões de bilhões. Agora o invasor terá de tentar, em média, várias outras chaves antes de encontrar a correta.

Esse conceito sobre o intervalo de possíveis chaves é conhecido como tamanho de chave. As chaves criptográficas são medidas em bits. Se alguém perguntar, “Qual é o tamanho dessa chave?” a resposta poderia ser 40 bits, 56 bits, 128 bits e assim por diante. Uma chave de 40 bits significa que o intervalo dos possíveis valores é de 0 até 2^{40} (aproximadamente 1 trilhão). Uma chave de 56 bits é de 0 até 2^{56} (aproximadamente 72 quatrilhões). O intervalo de uma chave de 128 bits é tão grande que é mais fácil apenas dizer que ela é uma chave de 128 bits, pois o valor chega até $2^{128.}$

Cada bit que se adicionar ao tamanho da chave dobrará o tempo requerido para um ataque de força bruta. Se uma chave de 40 bits levasse três horas para ser quebrada, uma chave de 41 bits levaria seis horas, uma chave de 42 bits, 12 horas e assim por diante. Isto acontece, pois para cada bit adicional dobra o número de chaves possíveis. Por exemplo, há oito números possíveis para o tamanho de 3 bits:

000 001 010 011 100 101 110 111

Esses são os números de zero até sete. Agora adicione mais um bit:

0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

Todos os números possíveis com 3 bits são possíveis com 4 bits, entretanto cada um desses números é possível “duas vezes”: uma vez com o primeiro bit não-configurado e novamente com ele configurado. Assim se adicionar um bit, dobra-se o número de chaves possíveis. Se dobrar o número de chaves possíveis, dobra-se o tempo médio que um ataque de força bruta leva para encontrar a chave correta.

Em resumo, se quiser tornar o trabalho de um invasor mais difícil, deve-se escolher uma chave maior. Chaves mais longas significam maior segurança. Qual o tamanho máximo que uma chave deve ter? Com o passar dos anos, o *RSA Laboratories* propôs alguns desafios. A primeira pessoa ou empresa a quebrar uma mensagem em particular ganharia um prêmio em dinheiro. Alguns desafios foram testes do tempo de um ataque de força bruta. Em 1997, uma chave de 40 bits foi quebrada em três horas e uma chave de 48 bits durou 280 horas. Em 1999, a *Electronic Frontíer Foundation* encontrou uma chave de 56 bits em 24 horas. Em cada um dos casos, pouco mais de 50% do espaço de chave foi pesquisado antes de a chave ser encontrada. Em janeiro de 1997, foi publicado um desafio de 64 bits. Até dezembro de 2000, ele ainda não tinha sido resolvido (Oliveira, E.E.L.2003).

Em todas essas situações, centenas ou até milhares de computadores operavam conjuntamente para quebrar as chaves. Na realidade, com o desafio de 56 bits de DES que a *Electronic Frontier Foundation* quebrou em 24 horas, um dos computadores era um cracker especializado em DES. Esse tipo de computador faz apenas uma função: verifica as chaves de DES. Um invasor que trabalhe secretamente, provavelmente não seria capaz de reunir a força de centenas de computadores e talvez não possua uma máquina especificamente construída para quebrar um algoritmo em particular. Para a maioria dos invasores, essa é a razão pela qual o tempo que leva para quebrar a chave quase certamente seria significativamente mais alto. Por outro lado, se o invasor fosse uma agência governamental de inteligência com grandes recursos, a situação talvez seria diferente.

Pode-se pensar em situações mais complexas. Suponha que se utiliza uma linha de base em uma situação exageradamente pior: examinar 1% do espaço de chave de uma chave de 56 bits leva 1 segundo e examinar 50% leva 1 minuto. Todas as vezes que se adicionar um bit ao tamanho de chave dobra-se o tempo de pesquisa.

Atualmente, 128 bits é o tamanho de chave simétrica mais comumente utilizada. Se a tecnologia avançar e os invasores de força bruta puderem melhorar esses números (talvez eles possam reduzir para alguns anos o tempo das chaves de 128 bits), então se precisaria utilizar uma chave de 256 bits.

Mesmo com o avanço da tecnologia, nunca será uma chave mais longa do que 512 bits (64 bytes). Suponha que cada átomo no universo conhecido (há aproximadamente 2^{300}) fosse um computador e que cada um desses computadores pudessem verificar 2^{300} chaves por segundo. Isso levaria cerca de 2162 milênios para pesquisar 1% do espaço de chave de uma chave de 512 bits. De acordo com a teoria do Big Bang, o tempo decorrido desde a criação do universo é menor de 224 milênios. Em outras palavras, é altamente improvável que tecnologia vá tão longe para forçá-lo a utilizar uma chave que seja “muito grande”.

2.3.2 Exemplos de Algoritmos Simétricos

Nesta serão discutidos alguns dos vários algoritmos de criptografia, mas as chaves não são intercambiáveis entre algoritmos. Por exemplo, suponha que se precisa criptografar os dados utilizando o algoritmo *Triple Digital Encryption Standard*, melhor conhecido como 3 DES. Se tentar decifrar os dados utilizando a cifragem de blocos *Advanced Encryption Standard* (AES), mesmo se utilizar a mesma chave, o resultado correto não será obtido.

A seguir se apresentam alguns algoritmos simétricos e suas características:

- *Digital Encryption Standard(DES*) é o algoritmo mais disseminado no mundo. Foi criado pela IBM em 1977 e, apesar de permitir cerca de 72 quadrilhões de combinações(2 elevado a 56), seu tamanho de chave(56 bits) é considerado pequeno, tendo sido quebrado por "força bruta" em 1977 em um desafio lançado na internet. O NIST(*National Institute of Standards and Technology),* que lançou o desafio mencionado, recertificou o DES pela última vez em 1993 e desde então está recomendando o 3DES. Assim o NIST propôs um substituto ao DES que deveria aceitar chaves de 128, 192 e 256 bits, operar com blocos de 128 bits, ser eficiente, flexível e estar livre de "*royalties*".

- *Triple-Des (3 DES),* é uma simples variação do DES, utilizando-o em três ciframento sucessivos, podendo empregar uma versão com dois ou com três chaves diferentes. É seguro, porém muito lento para ser um algoritmo padrão (Maia, L.P., 2005)

- *RC2 (*Maia, L.P., 2005), projetado por Ron Rivest da empresa *RSA Data Security Inc*. é utilizado no protocolo S/MIME, voltado para criptografia de e-mail corporativo. Também possui chave de tamanho variável. Rivest também é o autor do RC4 e RC6, este último foi concorrente ao AES (*Advanced Encryption Standard*).

- *IDEA (International Data Encryption Algorithm),* o International Data Encryption Algorithm foi criado em 1991 por James Massey e Xuejia Lai e possui patente da suíça *ASCOM SYSTEC*. O algoritmo é estruturado seguindo as mesmas linhas gerais do DES. Mas na maioria dos microprocessadores, uma implementação por

software do IDEA é mais rápida do que uma implementação por software do DES. O IDEA é utilizado principalmente no mercado financeiro e no PGP, o programa para criptografia de e-mail pessoal mais disseminado no mundo (Maia, L.P, 2005).

- Mars, algoritmo apresentado ao AES pela IBM Corporation (Hinz,M.A.M.,2005). Com bloco de 128 bits e uma chave de tamanho variável, indo de 128 bits até 400 bits. Sua implementação se dá em três fases. A primeira fase implementa uma rápida mistura dos dados do texto fonte, a inserção da chave e 8 voltas da transformação Type-3 Feistel. A segunda é a fase mais importante do processo, ela consiste de 16 voltas da transformação Type-3 Feistel, destas 16 voltas, 8 foram implementadas em *forward mode* e 8 em *backward mode*. A terceira e última fase é, essencialmente, o inverso da primeira fase.

- *Serpent*, algoritmo apresentado ao AES por três pesquisadores, são eles: Ross Anderson da Inglaterra, Eli Biham de Israel e Lars Knudsen da Noruega (Hinz,M.A.M., 2000). Possui um bloco de 128 bits dividido em 4 palavras de 32 bits cada e trabalha com um tamanho de chave de 128, 192 ou 256 bits. Basicamente esse algoritmo constitui-se de uma permutação inicial, 32 voltas onde se aplicam as funções criptográficas e uma permutação final.

2.4 Criptografia Assimétrica

A criptografia de chaves públicas foi inventada em 1976 por Whitfield Diffie e Martin Hellman a fim de resolver o problema da distribuição de chaves, encontrado nos algoritmos simétrico.

Está baseada no conceito de par de chaves: uma chave privada e uma chave pública. Qualquer uma das chaves é utilizada para cifrar uma mensagem e a outra para decifrá-la, conforme mostra a figura 2.3. As mensagens cifradas com uma das chaves do par só podem ser decifradas com a outra chave correspondente. A chave privada deve ser mantida secreta, enquanto a chave pública disponível livremente para qualquer interessado.

Figura 2.3 Processo de cifrar e decifrar usando chaves assimétricas (Maia, L.P, 2005).

Uma grande vantagem deste sistema é permitir que qualquer um possa enviar uma mensagem secreta, apenas utilizando a chave pública de quem irá recebê-la. Como a chave pública está amplamente disponível, não há necessidade do envio de chaves como é feito no modelo simétrico. A confidencialidade da mensagem é garantida, enquanto a chave privada estiver segura. Caso contrário quem possuir acesso à chave privada terá acesso às mensagens.

O sistema Assimétrico oferece assinatura digital, fato que não acontece com o uso de chaves simétricas. Além disso, o gerenciamento e a distribuição de chaves também são mais simples em comparação com algoritmos simétricos.

A assinatura digital produz uma mensagem que só uma pessoa poderia produzir, mas que todos possam verificar. A assinatura é um conjunto inforjável de dados assegurando o nome do autor, e funciona como uma assinatura de documentos, ou seja, que determinada pessoa concordou com o que estava escrito. Isso faz com que a pessoa que assinou o documento se responsabilize por ele (Pereira, F.D., 2005).

Porém a assinatura digital não pode ser empregada na prática isoladamente. Ela precisa de um mecanismo fundamental para a sua utilização, esse mecanismo é a função *Hashing*. Sua utilização como componente da assinatura digital se faz necessário devido à lentidão dos algoritmos assimétricos, em geral de 1000 vezes mais lentos do que os simétricos (Maia,L.P., 2005).

A função Hashing gera um valor pequeno, de tamanho fixo, derivado da mensagem que se pretende assinar, de qualquer tamanho. Sendo assim, essa função oferece agilidade nas assinaturas digitais, além de integridade confiável.

Os algoritmos de chave privada exploram propriedades específicas dos números primos e, principalmente, a dificuldade de fatorá-los, mesmo em computadores rápidos.

A seguir aparecem algumas descrições de alguns exemplos de algoritmos assimétricos:

- *RSA***,** algoritmo criado por Ron Rivest, Adi Shamir e Len Adleman em 1977. É, atualmente, o algoritmo de chave pública mais amplamente utilizado, ele utiliza números primos.Gerar a chave pública envolve multiplicar dois números primos grandes. Derivar a chave privada a partir da chave pública envolve fatorar um grande número. Assim, a segurança do RSA baseia-se na dificuldade de fatoração de números grandes. Uma chave RSA de 512 bits foi quebrada em 1999 pelo Instituto Nacional de Pesquisa da Holanda, com o apoio de cientistas de mais 6

países. Levou cerca de sete meses e foram utilizadas 300 estações de trabalho para a quebra (Maia,L.P., 2005).

- *ElGamal*, algoritmo que envolve a manipulação matemática de grandes quantidades numéricas. Sua segurança advém de algo denominado problema de logaritmo discreto. Sendo assim, esse algoritmo tem na sua segurança, a dificuldade de se calcular logaritmos discretos em um corpo finito.

- *Diffie-Hellman,* algoritmo que contém a chave pública mais antiga ainda em uso, ele também é baseado no problema de logaritmo discreto. Porém esse algoritmo não permite nem cifragem nem assinatura digital. O sistema foi criado para permitir a dois indivíduos entrarem em um acordo ao compartilharem um segredo tal como uma chave, muito embora eles somente troquem mensagens em público.

- *Curvas Elípticas***,** os sistemas criptográficos de curvas elípticas consistem em modificações de outros sistemas (o ElGamal, por exemplo), que passam a trabalhar no domínio dos corpos finitos. Eles possuem o potencial de proverem sistemas criptográficos de chave pública mais segura, com chaves de menor tamanho. Muitos algoritmos de chave pública, como o Diffie-Hellman, o ElGamal e o Schnorr podem ser implementados em curvas elípticas sobre corpos finitos. Assim, ficam resolvidos uns dos maiores problemas dos algoritmos de chave públicos: o grande tamanho de suas chaves. Porém, os algoritmos de curvas elípticas atuais, embora possuam o potencial de serem rápidos, são em geral mais demorados do que o RSA (Maia,L.P., 2005).

Foi apresentado nesse capítulo o conceito, a importância da criptografia no cotidiano das pessoas e alguns exemplos de algoritmos criptográficos existentes. Logo mais, será abordada a explicação mais completa sobre o algoritmo Mars, um algoritmo simétrico.

2.5 Projeto AES

Quando o algoritmo DES começou a dar sinais de que sua vida útil estava se aproximando do fim, mesmo com o desenvolvimento do 3-DES (triplo DES), o NIST (*National Institute of Standards and Technology*), que é o órgão do departamento de comércio dos EUA encarregado de aprovar padrões para o Governo Federal dos Estados Unidos, decidiu que o governo precisava de um novo padrão criptográfico para uso não-confidencial.

Devido a controvérsias existentes no DES sobre suspeitas do NSA (*National Security Agency),* o órgão do governo americano especializado em decifrar códigos ter ocultado um a porta dos fundos (*backdoor*) que pudesse facilitar ainda mais a decifração do DES por parte dela, se o NIST anunciasse um novo padrão os especialistas em criptografia concluiriam automaticamente que a NSA havia mesmo criado uma porta dos fundos no DES, e assim ela poderia ler tudo que fosse cifrado com ele causando a não utilização do padrão e sua possível extinção.

O NIST então decidiu patrocinar um concurso de criptografia. Em janeiro de 1997, pesquisadores do mundo inteiro foram convidados a submeter propostas para um novo padrão, chamado AES (*Advanced Encryption Standard*). O concurso possuía algumas regras tais como:

1. O algoritmo teria de ser uma cifra de bloco simétrica
2. Todo o projeto teria de ser público
3. Deveriam ser admitidos tamanhos de chaves iguais a 128, 192 e 256 bits
4. Teriam de ser possíveis implementações de software e hardware

5. O algoritmo teria de ser público ou licenciado em termos não-discriminatórios.

Foram selecionadas 15 propostas sérias na fase inicial, os algoritmos foram (Nist01): CAST-256, DEAL, DFC, E2, FROG, HPC, LOKI197, Magenta, MARS, RC6, Rjindael, Safer+, Serpent e Twofish. Essas propostas foram organizadas para serem apresentadas em conferências públicas onde os participantes tentavam encontrar falhas nelas. Após esse processo houveram 5 finalistas, escolhidos pelo NIST em 1998, a decisão tomada foi baseada nos requisitos de segurança, eficiência, simplicidade, flexibilidade e memória (importante para sistemas embarcados)

Os cincos finalistas foram:

1. Rijndael (de Joan Daemen e Vincent Rijmen, 86 votos)
2. Serpent (de Ross Anderson, Eli Biham e Lars Knudsen, 59 votos)
3. Twofish (de uma equipe liderada por Bruce Schneier, 31 votos)
4. RC6 (da RSA Laboratories, 23 votos)
5. MARS (da IBM, 13 votos)

O NIST anunciou em 2000 que o Rijndael finalmente seria o escolhido e em 2001, ele se tornou o novo padrão desejado, conhecido como AES (*Advanced Encryption Standard*) e foi publicado pelo *Federal Information Processing Standard* FIPS197.

Devido à grande abertura da competição não houve controvérsias quanto a possível confiabilidade do algoritmo, ou seja, o projeto AES cumpriu com sua finalidade de escolha de um novo padrão sem desconfianças.

A tabela 2.1 apresenta os algoritmos selecionados na fase inicial e seus respectivos países e autores:

Tabela 2.1 – Sumário dos algoritmos concorrentes do AES (BIHAM, 1999)

Algoritmo	**País**	**Autores**
LOKI97	Austrália	Lawrie Brown, Josef Pieprzk, Jennifer Seberry
RJINDAEL	Bélgica	Joan Daemen, Vincent Rijmen
CAST-256	Canadá	Entrust Technologies
DEAL	Canadá	Outerbridge, Knudsen
FROG	Costa Rica	TecApro internacional S.A.
DFC	França	Centre National pour la Recherche Scientifique (CNRS)
MAGENTA	Alemanha	Deutsche Telekom AG
E2	Japão	Nippon Telegraph and Telephone Corporation (NTT)
CRYPTON	Coréia	Future Systems, Inc.
HPC	EUA	Rich Schroeppel
MARS	EUA	IBM
RC6	EUA	RSA Laboratories
SAFER+	EUA	Cylink Corporation
TWOFISH	EUA	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Fergunson
SERPENT	Inglaterra, Israel, Noruega	Ross Anderson, Eli Biham, Lars Knudsen

A tabela 2.2 apresenta as estruturas utilizadas pelos algoritmos concorrentes do AES, nela pode-se perceber a grande influência causada por Feistel, uma vez que grande parte dos algoritmos utiliza a estrutura proposta por esse pesquisador.

Tabela 2.2 – Características Gerais dos concorrentes do AES (BIHAM, 1999)

Algoritmo	Estrutura	Iterações
LOKI97	Feistel	16
RJINDAEL	Square	10, 12, 14
CAST-256	Feistel Estendido	48
DEAL	Feistel	6, 8
FROG	Especial	8
DFC	Feistel	8
MAGENTA	Feistel	6, 8
E2	Feistel	12
CRYPTON	Square	12
HPC	Omni	8
MARS	Feistel Estendido	32
RC6	Feistel	20
SAFER+	SP Network	8, 12, 16
TWOFISH	Feistel	16
SERPENT	SP Network	32

2.5.1 Breve descrição dos 5 finalistas

O MARS (IBM, 1999) é um algoritmo de criptografia simétrico que foi apresentado ao AES pela IBM. Possui um bloco de 128 bits e uma chave de tamanho variável, indo de 128 bits até 400 bits. Este algoritmo trabalha com uma palavra de 32 bits, usando então, 4 palavras por bloco.

A implementação do MARS se dá basicamente em três fases. A primeira fase implementa uma rápida mistura dos dados do texto fonte, a inserção da chave e 8 voltas da transformação *Type-3 Feistel*. A segunda fase é a fase mais importante do processo, ela consiste de 16 voltas da transformação *Type-3 Feistel*, destas 16 voltas, 8 foram implementadas em *forward mode (modo para frente)* e 8 em *backward mode(modod para trás).* A terceira e última fase é, essencialmente, o inverso da primeira fase.

O algoritmo RC6 (Rivest, 1995) foi desenvolvido pesquisadores do MIT e dos Laboratórios RSA e é proveniente do RC5. O RC6 é talvez o mais simples dos algoritmos

apresentados, além de sua simplicidade o RC6, ao contrário da maioria dos algoritmos criptográficos existentes, se destaca por não usar *S-Boxes (caixas de substituição).*

O RC6 é parametrizado por 3 parâmetros: *w* que é o tamanho do bloco, *r* que denota o número de voltas e *b* que é o tamanho da chave em bytes. Para ir ao encontro das definições do NIST, o tamanho do bloco é de 32 bits, o número de voltas é igual a 20 e o tamanho da chave pode variar de 0 até 255 bits.

O RC6 trabalha com 4 palavras de *w* bits cada uma que serão chamadas de registradores, estes 4 registradores (*A, B, C, D*) são usados tanto para receber o texto de entrada quanto para devolver o texto de saída.

O algoritmo Rjindael (Daemem, 1999) foi desenvolvido por dois pesquisadores belgas, o Rijndael é um algoritmo de blocos iterativos com tamanho de bloco e de chave variáveis, podendo ser especificados independentemente para 128, 192 ou 256 bits.

O Rijndael diferencia-se da maioria dos outros algoritmos que são usados atualmente, pois não usa uma estrutura do tipo *Feistel* na sua fase de rotação. Numa estrutura *Feistel,* os bits de um estado intermediário são transpostos em uma outra posição sem serem alterados; no Rijndael, a fase de rotação é composta de transformações uniformes inversíveis distintas chamadas de *layers*.

O Serpent (Anderson, 1999) foi um algoritmo apresentado ao AES por três pesquisadores, são eles: Ross Anderson da Inglaterra, Eli Biham de Israel e Lars Knudsen da Noruega.

O algoritmo Serpent possui um bloco de 128 bits dividido em 4 palavras de 32 bits cada e trabalha com um tamanho de chave de 128, 192 ou 256 bits.

Basicamente o algoritmo Serpent constitui-se de: uma permutação *inicial*; 32 voltas onde se aplicam as funções criptográficas e uma permutação final.

As funções criptográficas que são executadas a cada uma das 32 voltas são: uma inserção da chave, uma passagem pelas *S-Boxes e* uma transformação linear, que é descartada na última volta

O Twofish é um algoritmo apresentado ao AES por um grupo de pesquisadores da *Counterpane Systems*. O algoritmo Twofish (Schneier, 1998) possui um bloco de 128 bits e chaves que podem ter um tamanho de 128, 192 ou 256 bits.

Algumas estruturas matemáticas e técnicas de manipulação de dados são de vital importância na implementação do Twofish, entre elas podemos destacar: *Feistel Network*, *SBoxes*, Matrizes MDS, *Pseudo-Hadamard Transforms* (PHT) e *Whitening*.

2.5.2 Comparação dos 5 finalistas

A comparação dos algoritmos é realizada de maneira que eles possam ser analisados quanto aos fatores relevantes associados a algoritmos de criptografia, como desempenho, segurança e flexibilidade.

2.5.2.1 Complexidade Computacional

O NIST como já vimos anteriormente fazia algumas exigências para que um algoritmo pudesse ser submetido ao AES, uma dessas exigências era que os algoritmos pudessem ser aplicados tanto em software como em hardware, devido a isso os algoritmos criptográficos não são muito complexos computacionalmente.

A complexidade computacional pode ser entendida como o conjunto de alguns fatores tais como: memória necessária, portabilidade, etc.

2.5.2.2 Requisitos de Memória

Uma das características de grande importância para um algoritmo criptográfico é a capacidade dele cifrar ou decifrar a informação usando pouca memória, sendo assim, esse algoritmo poderia ser aplicado em máquinas com grande quantidade de memória ou em pequenos *smart cards* que possuem muito pouca memória (menos de 1KB).

A economia de memória é almejada principalmente na implementação em hardware do algoritmo criptográfico. Por exemplo, os *smart cards*, onde são implementados algoritmos de criptografia possuem no máximo 256 bytes de memória RAM (Schneier 2000).

A tabela 2.3 a seguir, segundo Schneier (2000), especifica a memória utilizada pelos algoritmos implementados em *smart cards*.

Tabela 2.3 – Memória Mínima Requerida para Implementação em *smart cards*

Algoritmo	Memória Mínima Requerida (bytes)
MARS	100
RC6	210
Rijndael	52
Serpent	50
Twofish	60

2.5.2.3 Velocidade de Processamento

A velocidade de processamento é um fator importante, pois o algoritmo deve apresentar segurança e um bom desempenho.

Para se realizar esse tipo de avaliação é necessária a realização de testes nas mais diversas plataformas.

Os gráficos a seguir, segundo Schneier (2000) mostram em ciclos de clock, o desempenho dos algoritmos cifrados na Linguagem C, utilizando os tamanhos de chave exigidos pelo NIST – 128, 192 e 256 bits.

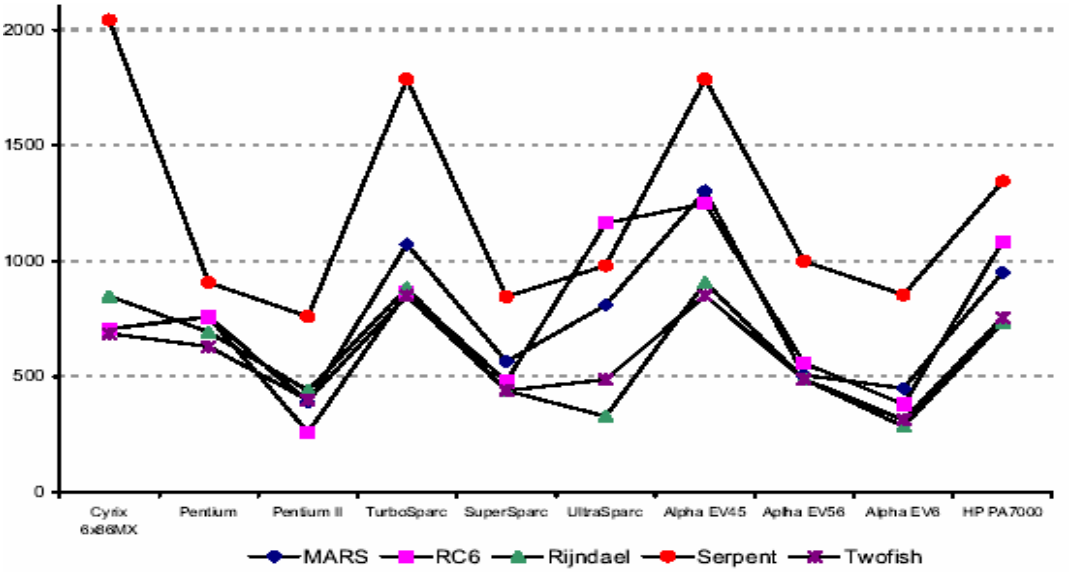

Figura 2.4 - Velocidade de cifrar para uma chave de 128 bits em C (Biham, 1999).

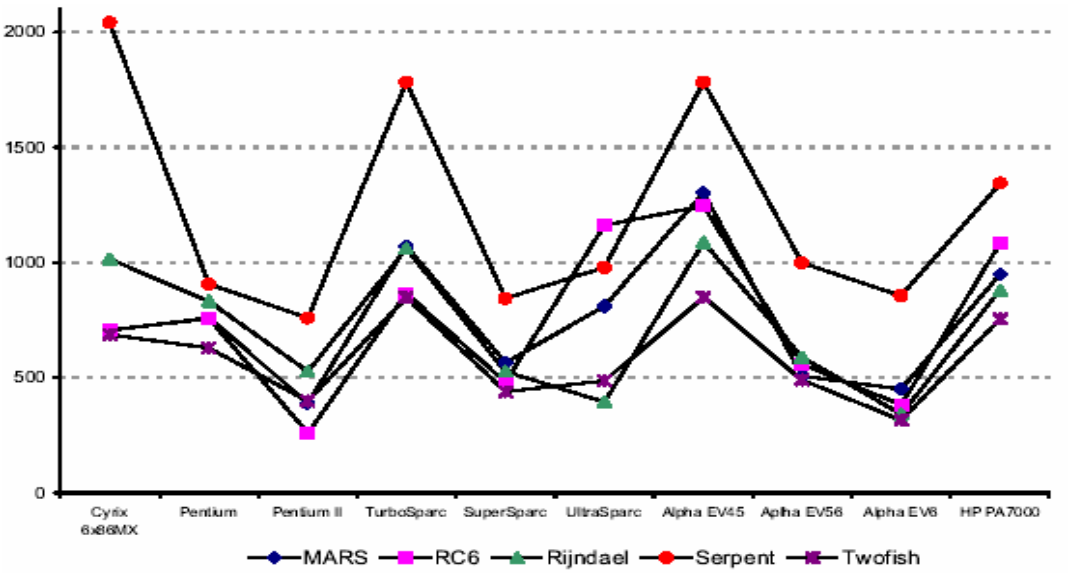

Figura 2.5 - Velocidade de cifrar para uma chave de 192 bits em C (Biham, 1999).

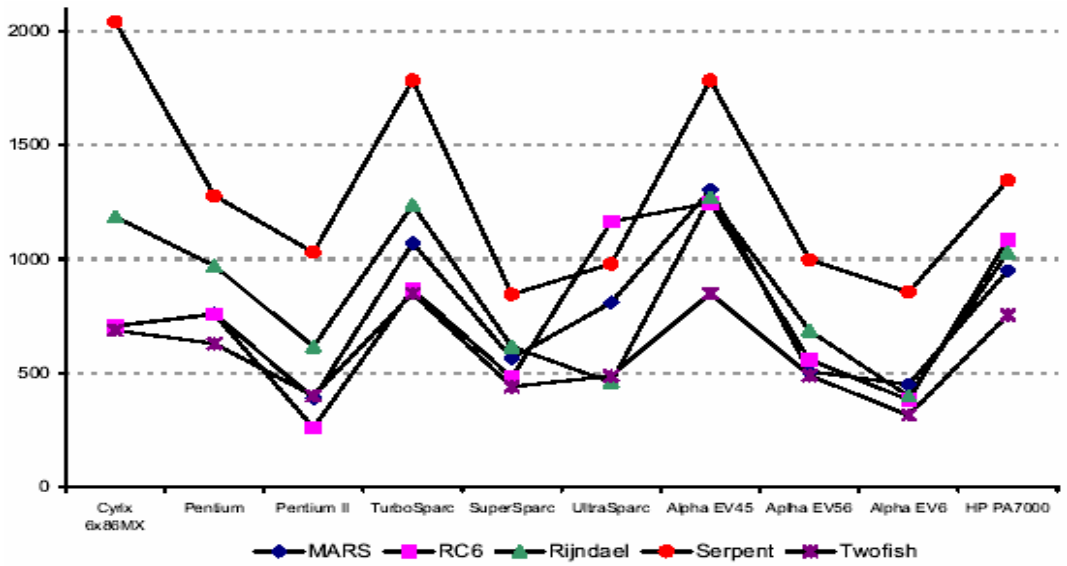

Figura 2.6 - Velocidade de cifrar para uma chave de 256 bits em C

(Biham, 1999).

2.5.2.4 Funções Criptográficas

A maioria dos algoritmos criptográficos geralmente utiliza estruturas e funções criptográficas semelhantes. Tais estruturas podem ser “levemente” modificadas ou possuírem outra denominação, mas o conceito permanece basicamente o mesmo.

Mas isso não significa que os algoritmos se utilizem apenas dessas funções criptográficas, ou seja, eles podem também utilizar outras novas funções.

A tabela 2.4 especifica as funções criptográficas utilizadas pelos 5 algoritmos finalistas do projeto AES.

Tabela 2.4 – Utilização das funções criptográficas (Hinz, M.A.M.,2000).

	XOR	ADD	S-Box	Feistel	Deslocamento	Multiplicação
MARS	**X**	**X**	**X**	**X**	**X**	**X**
RC6	**X**	**X**		**X**	**X**	**X**
Rijndael	**X**	**X**	**X**		**X**	**X**
Serpent	**X**		**X**		**X**	
Twofish	**X**	**X**	**X**	**X**	**X**	**X**

Teoricamente quanto maior o número de estruturas diferentes que um algoritmo utiliza para cifrar e decifrar maior a segurança proporcionada, pois a informação sofre maiores mudanças, mas por outro lado há uma perda na flexibilidade em virtude de algumas estruturas serem mais fáceis de serem implementadas em software e outras em hardware.

Apesar do algoritmo Mars não ter sido o grande vencedor do projeto AES, muitas soluções de segurança ainda usam esse algoritmo. Por esse motivo, esse projeto de pesquisa dedica-se ao algoritmo Mars, que será mais bem estudado nos próximos capítulos.

Capítulo 3

ALGORITMO MARS IMPLEMENTADO EM C

3.1 Definição do algoritmo

Mars é um algoritmo de criptografia simétrico que usa um bloco de 128 bits e uma chave de tamanho variável, podendo mudar de 128 bits até 400 bits. O algoritmo foi projetado pela IBM para ser um candidato ao padrão avançado do AES (*Advanced Encryption Standard*), e ficou entre os 5 últimos finalistas, mas não conseguiu ser o grande vencedor, perdendo para o algoritmo Rinjndael, conhecido como AES.

O algoritmo Mars oferece mais segurança que o algoritmo Triple DES, e consegue ser mais rápido que o DES. A combinação de rapidez, segurança e flexibilidade, fazem desse algoritmo uma ótima escolha para cifrar informações que devem ser mantidas em segredo.

O algoritmo Mars pode ser usado em várias aplicações, incluindo a proteção de login e senhas, mensagens em e-mail, entre várias outras.

O algoritmo Mars trabalha com uma palavra de 32 bits, usando então, 4 palavras por bloco, sendo que cada bloco suporta 8 bits.

3.2 Funcionamento do algorítmo

O algoritmo Mars tem como entrada 4 palavras de 32 bits. A implementação acontece em três fases, a primeira é uma mistura rápida de dados do texto fonte, a inserção da chave e 8 voltas da transformação *Type-3 Feistel*. A seguir um trecho do código, onde as variáveis a,b,c,d recebem os dados do texto fonte e a inserção da chave (representada por l_key[]) e as 8 voltas são realizadas com a chamada da função f_mix.

```
a = in_blk[0] + l_key[0];
b = in_blk[1] + l_key[1];
c = in_blk[2] + l_key[2];
d = in_blk[3] + l_key[3];
                    f_mix(a,b,c,d); a += d;
                    f_mix(b,c,d,a); b += c;
                    f_mix(c,d,a,b);
                    f_mix(d,a,b,c);
                    f_mix(a,b,c,d); a += d;
                    f_mix(b,c,d,a); b += c;
                    f_mix(c,d,a,b);
  f_mix(d,a,b,c);
```

Na segunda fase acontecem 16 voltas da transformação *Type-3 Feistel*, onde 8 voltas foram implementadas em *forward mode(modo para frente)* e 8 em *backward mode (modo para trás)*. As 16 voltas acontecem quando se chama a função f_Ktr, como mostra o código seguinte:

```
f_ktr(a,b,c,d, 4); f_ktr(b,c,d,a, 6); f_ktr(c,d,a,b, 8); f_ktr(d,a,b,c,10);
f_ktr(a,b,c,d,12); f_ktr(b,c,d,a,14); f_ktr(c,d,a,b,16); f_ktr(d,a,b,c,18);
f_ktr(a,d,c,b,20); f_ktr(b,a,d,c,22); f_ktr(c,b,a,d,24); f_ktr(d,c,b,a,26);
f_ktr(a,d,c,b,28); f_ktr(b,a,d,c,30); f_ktr(c,b,a,d,32); f_ktr(d,c,b,a,34);
```

A terceira fase é o inverso da primeira (Hinz, M.A.M.,2000).

Mars usa operações lógicas do tipo or, xor, e operações aritméticas do tipo adição, subtração, multiplicação, rotação. As operações de adição, subtração e xor são operações

simples que são usadas para misturar valores. A combinação de multiplicação com rotação acontece para se obter mais segurança.

Também faz uso de um vetor, os S-Boxes, que são caixas de substituição não lineares que visam misturar o texto cifrado para que se torne mais difícil a sua decifragem. Elas variam de tamanho de entrada e saída. Nesse algoritmo, as S-Boxes contém 512 entradas, mas para simplificar o entendimento será usado 2 S-Boxes de 256 entradas cada uma (IBM – Corporation, 1999).

Usa-se no algoritmo o método Type-3 Feistel, que transforma qualquer função em permutação. Ela faz o mapeamento dependente de uma chave, de uma string de entrada dentro de uma string de saída.

Na fase um, a primeira operação a ser feita é a inserção de uma chave (representada na figura 3.1 pela letra K) em cada palavra de dados (representada na figura 3.1 pela letra D). Seguem-se então 8 voltas da transformação Type-3 Feistel. Em cada volta é utilizada uma palavra para modificar as outras três palavras, fazendo com que a cada volta as palavras mudem de posição: a atual primeira palavra destino torna-se a segunda palavra fonte; a atual segunda palavra destino se torna a terceira palavra fonte; a atual terceira palavra destino torna-se a quarta palavra fonte e a quarta palavra destino torna-se a primeira palavra fonte. Para a realização de toda essas combinações de dados, são aplicadas operações de rotação, xor e adição(ADD) nessa primeira fase (Dellani, P.R.,2004).

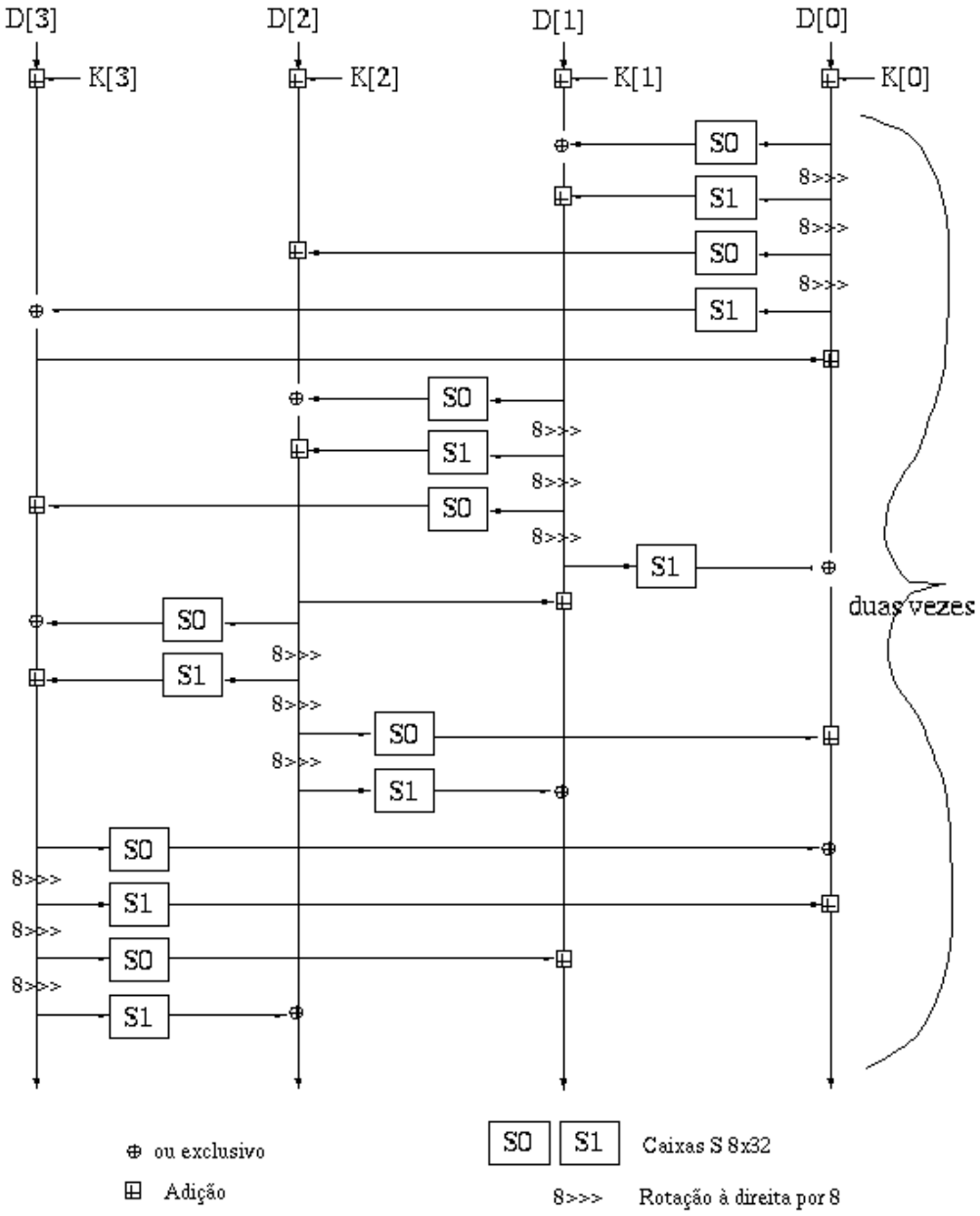

Figura 3.1 Fase Um – Mixagem dos dados no
Algoritmo Mars (Hinz, M.A.M. , 2000).

Na fase dois, acontece um laço de 16 voltas, iterações a função Type-3 Feistel, em cada laço é aplicada uma função E, que faz uso das operações de multiplicação, rotação, XOR, ADD. Para garantir uma maior segurança contra ataques, nas oito primeiras rodadas (modo para frente), a primeira e a segunda saídas da função E são somadas à primeira e à

segunda palavra alvo, e é feita uma operação de ou-exclusivo entre a terceira saída e a terceira palavra alvo. Já nas outras oito rodadas (modo para trás), se adiciona a primeira e a segunda saída da função E à terceira e segunda palavras alvo, e é executada uma operação de ou-exclusivo entre a terceira saída com a primeira palavra alvo, conforme se mostra na figura 3.2.

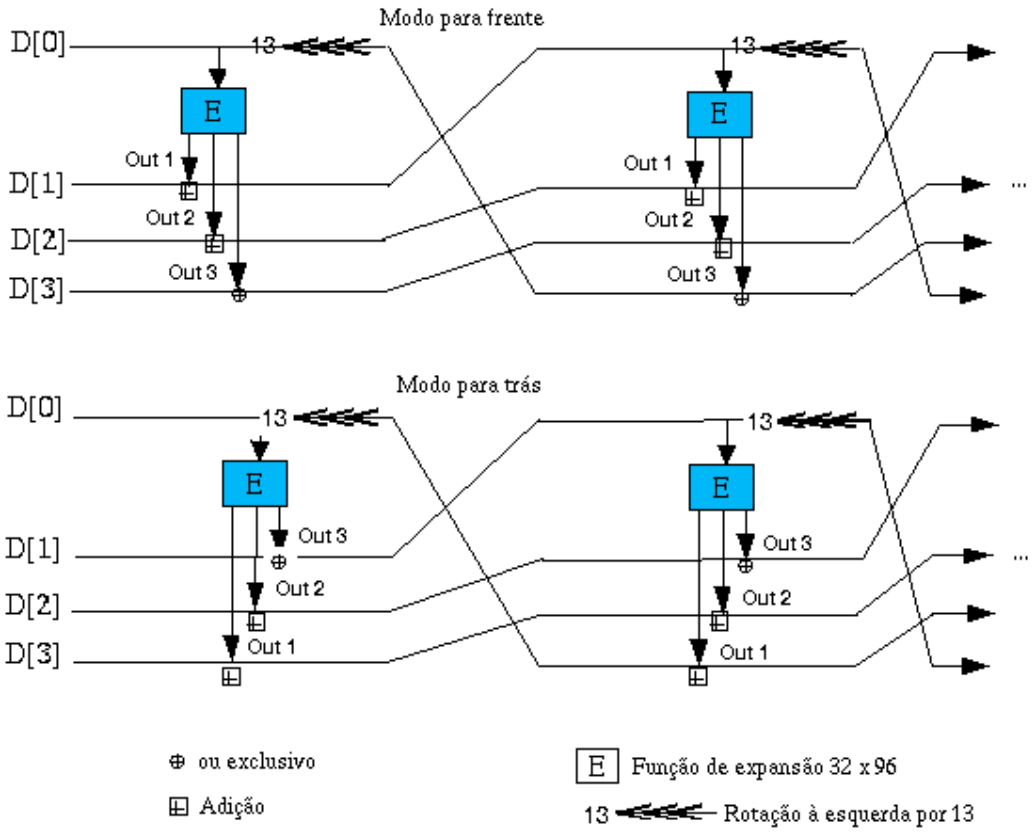

Figura 3.2 Fase dois do algoritmo Mars (Hinz, M.A.M. , 2000).

A função E usa uma palavra de dados e mais duas palavras chaves para produzir três palavras de saída. É feito o uso de variáveis temporárias L,M,R. Algumas das operações que ocorrem na função E são: M recebe o valor da entrada adicionado a uma chave, R armazena o valor da primeira entrada rotacionado 13 posições à esquerda, L armazena o valor que é conseguido na S-Box usando o valor de M como índice, R é rotacionado cinco posições a esquerda, M é rotacionado à esquerda, usando os 5 bits menos significativos de R é feito um

xor de R dentro de L, rotaciona-se novamente R cinco posições à esquerda, feito um xor de r dentro de L e outro xor de R (Dellani, P.R., 2004).

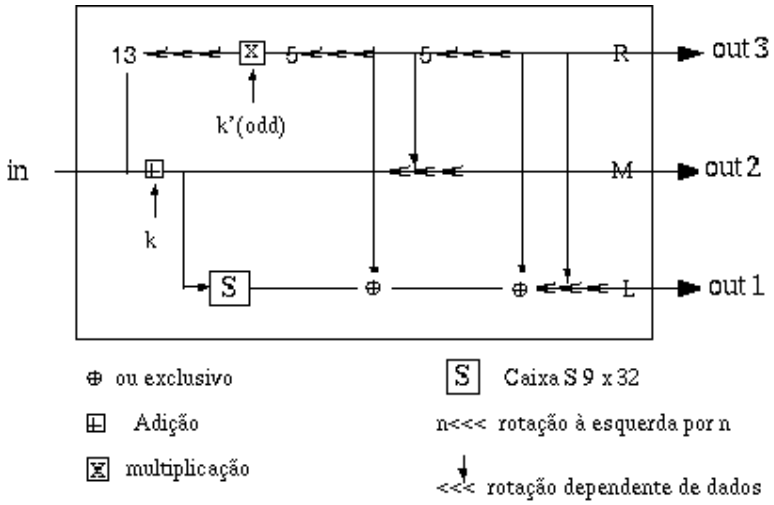

Figura 3.3 Função E, usada na segunda fase do algoritmo Mars (Hinz, M.A.M.,2000)

A fase três, última fase deste algoritmo, é praticamente igual a primeira fase, com a diferença de que os dados são processados na ordem inversa. Esta fase consiste de 8 voltas da transformação Type-3 Feistel, seguida da subtração das palavras chaves das palavras de dados, conforme mostra a figura 3.4.

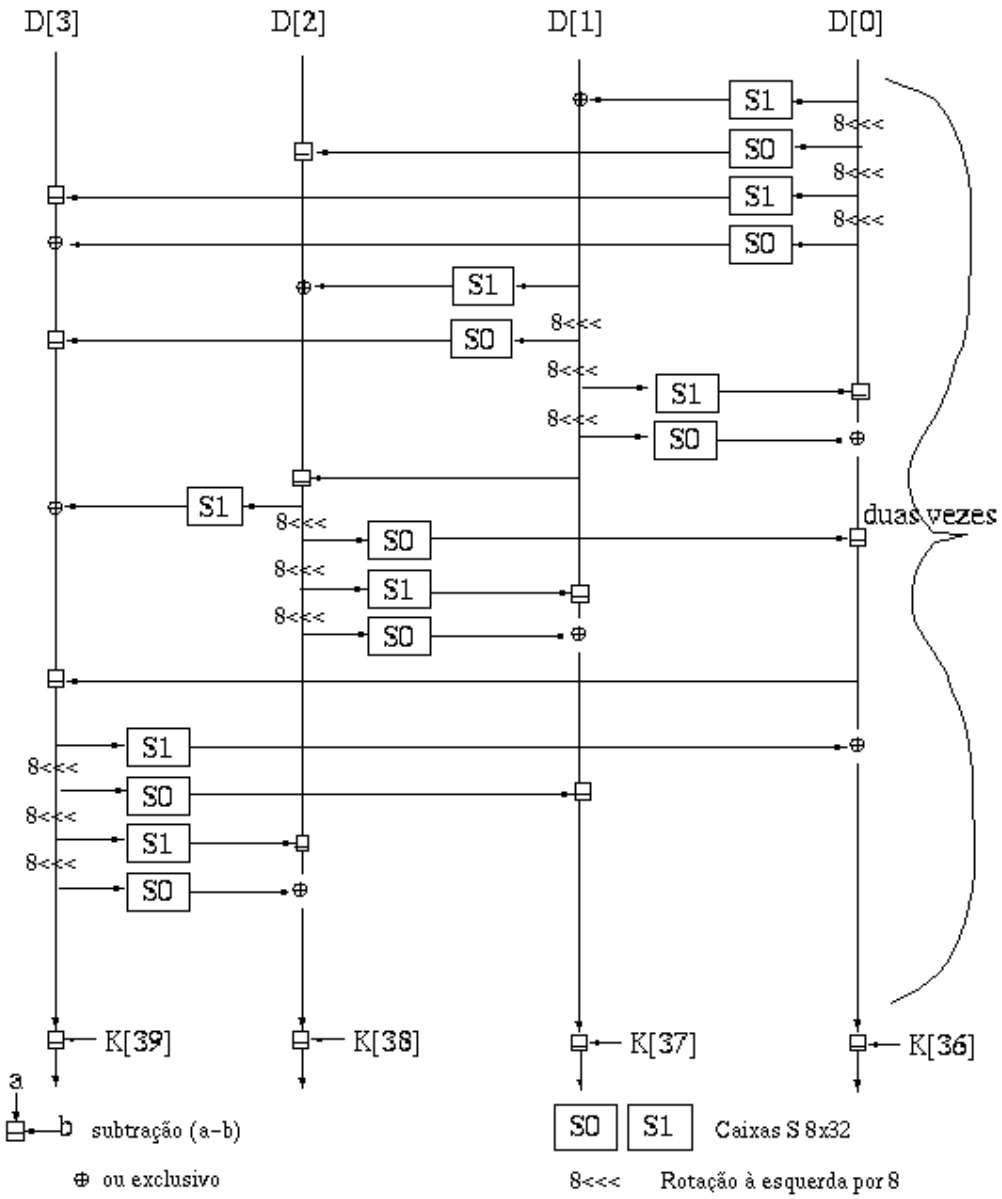

Figura 3.4 Fase três – Mixagem para trás
do algoritmo Mars (Hinz, M.A.M. , 2000).

3.3 Teste de Desempenho em C

Foram feitos testes em computadores diferentes usando o mesmo programa Mars implementado na linguagem C, cuja explicação de seu funcionamento foi apresentada

anteriormente, e seu código encontra-se em anexo. Busca-se identificar o tempo que um arquivo de extensão doc, mp3, mpg demora em cifrar e decifrar.

Na tabela 3.1 encontra-se o desempenho de um computador Celeron 633 Mhz, 240Mb Ram, Windows 98, usando um chave de tamanho 128 bits. Pode-se observar que o tempo que é gasto para cifrar um arquivo texto de tamanho 132Kb é menor que o tempo para cifrar um arquivo de música de tamanho 37,9 Mb.

Tabela 3.1. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem C, usando um Celeron 633 Mhz.

	Tempo para cifrar	Tempo para decifrar
Arquivo.doc 132 Kb	0.55 segundos	0.44 segundos
Musica.mp3 2,83 MB	2.20 segundos	1.21 segundos
Vídeo.mpg 37,9 MB	37.13 segundos	22.79 segundos

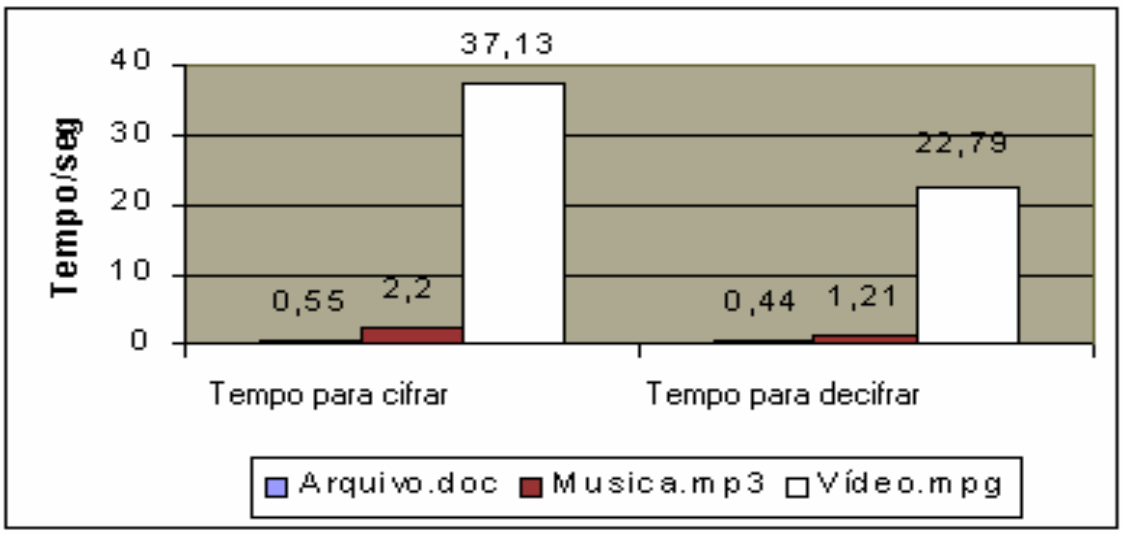

Figura 3.5. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem C, usando um Celeron 633 Mhz.

Desempenho de um computador Pentium 41.60 GHz, windows XP, 256 Mb RAM, usando chave de tamanho 128 bits.

Tabela 3.2. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem C, usando um Pentiun 41.60 Ghz.

	Tempo para cifrar	Tempo para decifrar
Arquivo.doc 132Kb	0.015 segundos	0.016 segundos
Musica.mp3 2,83MB	0.406 segundos	0.468 segundos
Vídeo.mpg 37,9MB	14.61 segundos	15.67 segundos

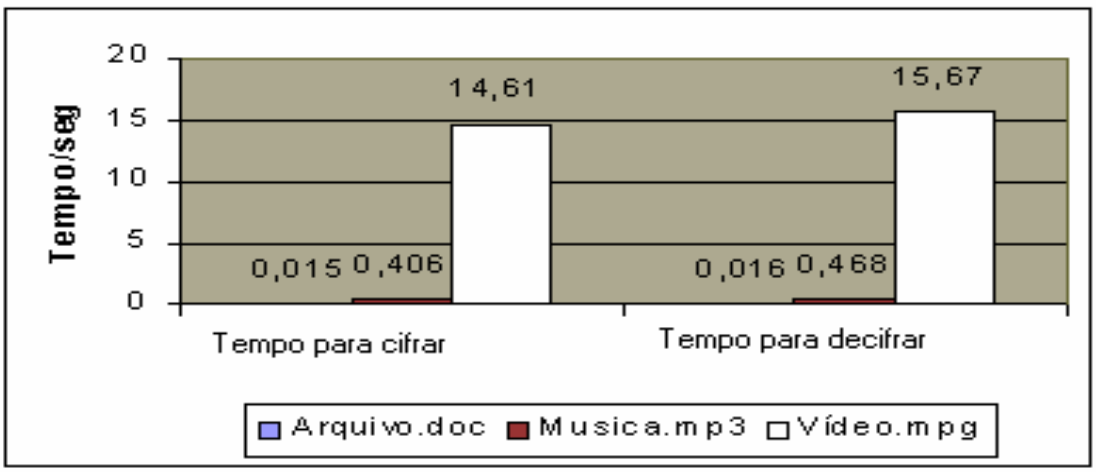

Figura 3.6. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem C, usando um Pentiun 41.60 Ghz.

Desempenho de um computador Celeron 466 Mhz , 63 Mb RAM, windows 98, usando chave de tamanho 128 bits.

Tabela 3.3. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem C, usando um Celeron 466 Mhz.

	Tempo para cifrar	Tempo para decifrar
Arquivo.doc 132kb	0.44 segundos	0.33 segundos
Musica.mp3 2,83MB	1.87 segundos	1.32 segundos
Vídeo.mpg 37,9MB	30.48 segundos	30.92 segundos

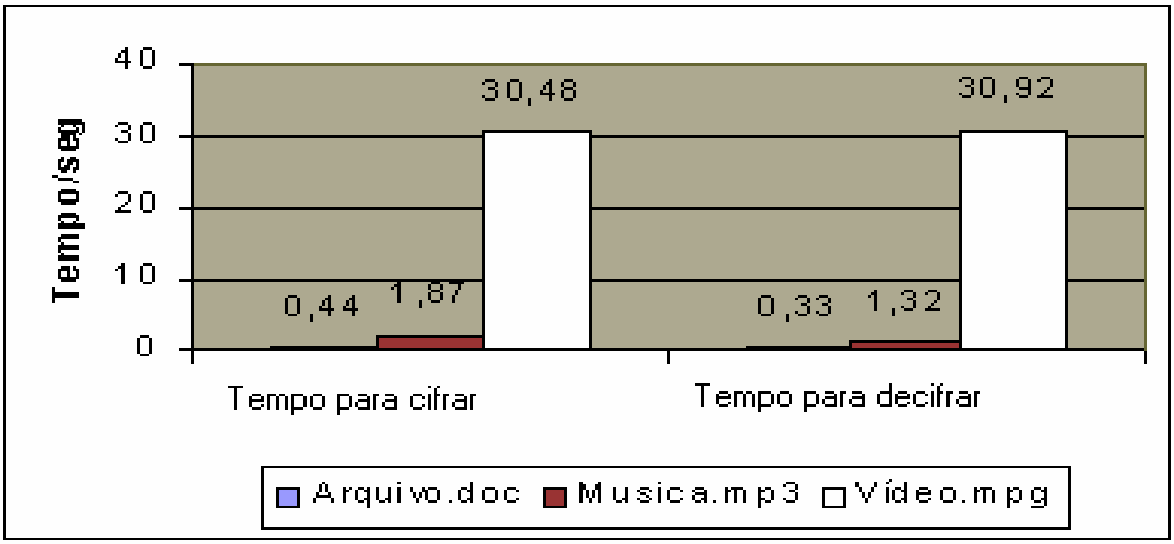

Figura 3.7. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem C, usando um Celeron 466 Mhz.

No capitulo 2 foram apresentados alguns algoritmos de criptografia usando chaves de tamanho diferentes, chaves de 128, 192, 256 bits e seu desempenho, implementado na linguagem C, onde a velocidade foi medida em clock.

Conforme mostra a figura 2.4, 2.5, 2.6, pode-se concluir que o algoritmo Mars, estudado nesse trabalho, apresenta um desempenho diferente para cada máquina que é testado, se testado na mesma máquina mudando apenas a chave 128 para 192, o algoritmo apresenta o mesmo desempenho. Mas quando muda a chave para 256 usando as mesmas máquinas, o algoritmo apresenta um desempenho diferente apenas para computadores Pentiun e Pentiun II.

3.4 Conclusão do teste de desempenho

Foram feitos testes em 2 computadores Celeron, variando apenas a freqüência de um computador para outro, e um teste em um Pentium 41.60 Ghz, usando-se o algoritmo Mars implementado na linguagem C.

Os arquivos com extensão doc, são mais rápidos para cifrar ou decifrar, mas não existe uma grande diferença de tempo em relação aos arquivos de musica. Já os arquivos de vídeo, são os mais demorados, devido ao seu tamanho ser maior que os outros arquivos, pode ser notado tal afirmação nos gráficos anteriormente apresentados.

Capítulo 4

ALGORITMO MARS NA LINGUAGEM JAVA

4.1 Historia da linguagem Java

A história da linguagem Java tem início em 1991 com o *Green Project*. Este projeto pretendia se tornar a próxima geração de software embarcado e acabou sendo o “ponta pé inicial” da idéia Java. Em 1991-1992 James Gosling, Mike Sheridan e Patrick Naughton iniciaram seus trabalhos com a linguagem "Oak", o primeiro antecedente do Java (Loiola,M. A.).

O projeto Oak foi incorporado então ao *FirstPerson*, um produto idealizado como a solução para o emergente mercado de TV iterativa. Em 1993 a Sun tentou vender o Oak como linguagem de programação para Set-top boxes, mas falhou na tentativa. Ainda em 1993, Marc Anderseen e o grupo NCSA lançaram o primeiro browser gráfico para Web, o Mosaic, assim começou a era da "World Wide Web" (Deitel, H.M.. 2003).

Em 23 de maio de 1995 ocorreu o lançamento oficial da Linguagem Java. Desde então a Sun já fez três revisões principais da linguagem: a versão 1.02 lançada em 1996 suportava conectividade com banco de dados e objetos distribuídos. A versão 1.1, lançada em 1997 adicionava um modelo robusto de eventos, internacionalização e o modelo de componentes Java Beans. A versão 1.2 (Java 2) lançada no final de 1998 trouxe o toolkit de interface para o usuário chamado Swing que finalmente permitia aos programadores escrever aplicativos GUI realmente portáveis (Loiola,M. A., 2005).

Java possui extensões para se trabalhar com aplicativos desktop (J2SE), aplicativos corporativos (J2EE) e até mesmo aplicativos destinados a dispositivos fisicamente pequenos como celular, palm-top,etc,(J2ME). A versão atual do J2SE é o J2SE 1.5 codinome "Tiger" que foi lançada em fevereiro de 2004 e tem em foco a facilidade de desenvolvimento, escalabilidade e performance, monitoramento e gerenciamento de desktop (Deitel, H.M, 2003).

Java hoje é usado para criar páginas na Web com conteúdo iterativo e dinâmico, para desenvolver aplicativos corporativos de grande porte, para aprimorar a funcionalidade de servidores da Word Wide Web (os computadores que fornecem o conteúdo que se observa nos navegadores da Web), fornecer aplicativos para dispositivos destinados ao consumidor final (como telefones celulares, pagers e assistentes pessoais digitais) e para muitas outras finalidades (Deitel, H.M, 2003).

Uma vantagem da programação em Java, é que ela disponibiliza, uma biblioteca com varias classe já prontas, que contém métodos que realizam tarefas e retornam o valor obtido. Essas classes são também conhecidas como Java APIs(*Application Programming Interfaces* – interfaces de programas aplicativos) (Deitel, H.M., 2003).

4.2 APIs usadas na criptografia

O Java, que possui como um de seus lemas a segurança, fornece uma API poderosa para se trabalhar com criptografia de dados. Dois métodos existentes, *Message Digest* e Assinaturas Digitais.

Message Digest, são funções hash que geram um código de tamanho fixo, a partir de dados de tamanha arbitrário, mas esses dados não podem ser decifrados. Esses códigos são

usados para seguranças de senhas, não podem ser decifrados, os códigos hash precisa ser regerado e comparado com a seqüência disponível anteriormente. Se ambos se igualarem, o acesso é liberado.

Assinaturas Digitais têm como utilidade autenticar o remetente da informação e dar garantia de que o dado é confiável. Elas trabalham com uma chave pública e uma chave privada. É de responsabilidade do remetente fornecer a chave pública aos destinatários. No ato do envio, o remetente gera uma assinatura para o dado que deseja enviar usando a chave privada. O destinatário recebe o dado e a assinatura, e valida a informação usando sua chave pública. Se a validação for efetuada com sucesso, o destinatário tem a garantia de que a mensagem foi enviada por um remetente confiável, possuidor da chave privada (Cunha, R. F).

4.3 Implementação do Algoritmo Mars em Java

A implementação do algoritmo foi feita baseada no código do algoritmo já estudado na linguagem C, descrito no capítulo anterior.

Existiram algumas dificuldades na programação em Java, pois ela não possui alguns recursos que estavam sendo usados na linguagem C, recursos como ponteiros, alguns tipos de variáveis, mas com base em muito estudo, tudo conseguiu ser bem transformado até se chegar ao objetivo final, que era a transformação do algoritmo na linguagem Java.

As variáveis utilizadas foram transformadas em atributos, como mostra a seguir um pequeno trecho do código do algoritmo Mars :

```
private static long l_key[] = new long [40];
private  String buff[] = new String [64];
private int nc;
private long a, b, c, d, l, m, r ;
```

As funções em métodos; a seguir dois métodos usados no algoritmo:

```
public long rotr(long x,long n)
  { return x>>n;
        }
public long rotl(long x,long n)
    {  return x<<n;
  }
```

O código na linguagem Java encontra-se no apêndice A.

4.4 Teste de Desempenho em Java e comparação com Desempenho em C

Neste capítulo se apresentam resultados dos testes de cifrar e decifrar arquivos texto, música e vídeo, para obter o desempenho do algoritmo Mars implementado na linguagem Java, assim como foi feito no capítulo anterior utilizando a linguagem C. São também feita comparações dos gráficos obtidos.

Desempenho de um computador Celeron 633 Mhz, 240MB Ram, Windows 98, usando chave de tamanho 128 bits, na linguagem Java, para cifrar e decifrar arquivos texto, música e vídeo, é apresentado na figura 4.1.

Tabela 4.1. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem Java, usando um Celeron 633 Mhz.

	Tempo para cifrar	Tempo para decifrar
Arquivo.doc 132 Kb	1.65 segundos	1.48 segundos
Musica.mp3 2,83 MB	31.09 segundos	28.23 segundos
Vídeo.mpg 37,9 MB	352.73 segundos	392.78 segundos

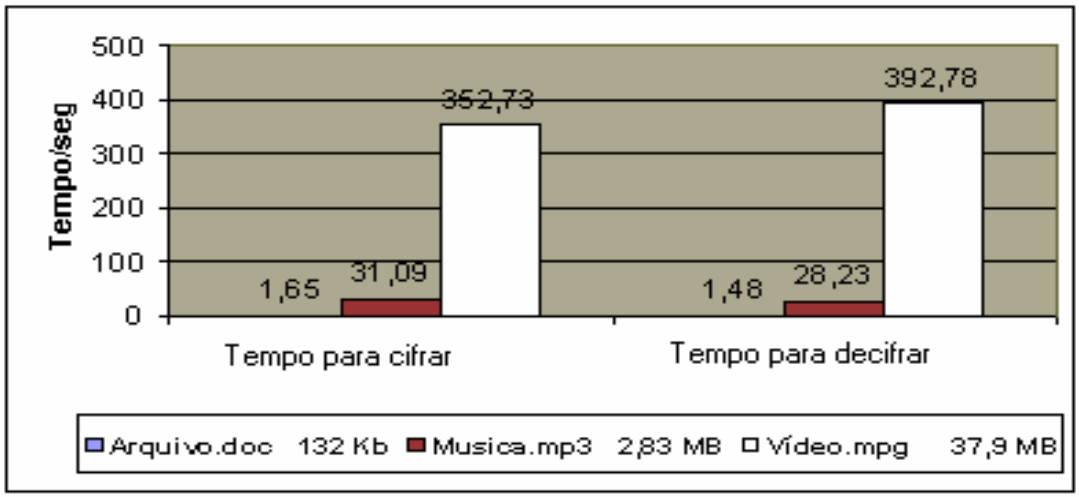

Figura 4.1. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem Java, usando um Celeron 633 Mhz.

Tabela 4.2. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem Java usando um Pentiun 41.60 Ghz.

	Tempo para cifrar	Tempo para decifrar
Arquivo.doc 132Kb	0.609 segundos	0.531 segundos
Musica.mp3 2,83MB	11.016 segundos	10.656 segundos
Vídeo.mpg 37,9MB	146.265 segundos	145.453 segundos

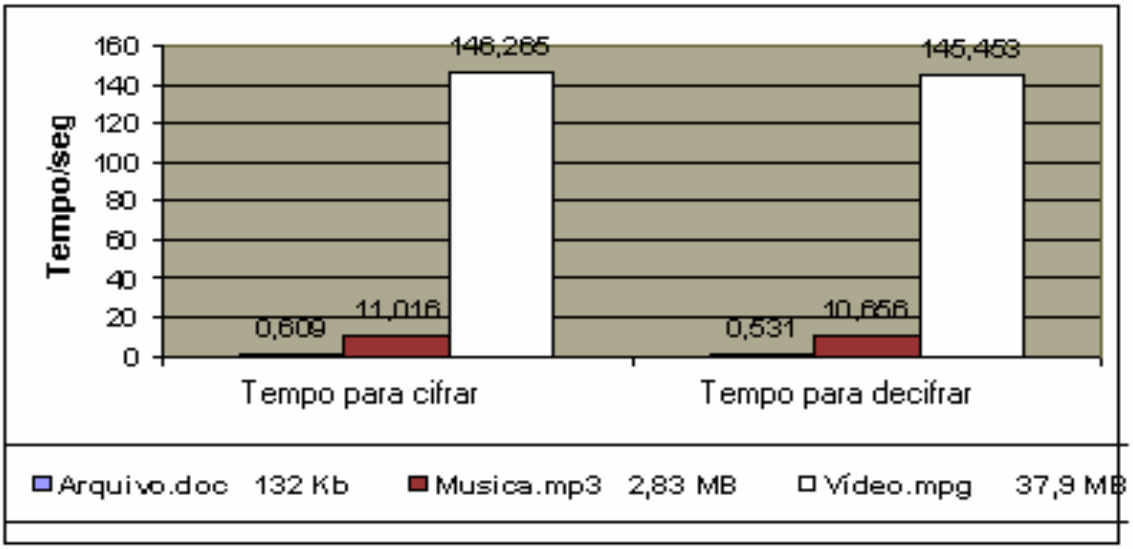

Figura 4.2. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem Java, usando um Pentiun 41.60 Ghz.

Tabela 4.3. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem Java, usando um Celeron 466 Mhz.

	Tempo para cifrar	Tempo para decifrar
Arquivo.doc 132kb	1.59 segundos	1.43 segundos
Musica.mp3 2,83MB	31.3 segundos	31.48 segundos
Vídeo.mpg 37,9MB	419.41 segundos	426.83 segundos

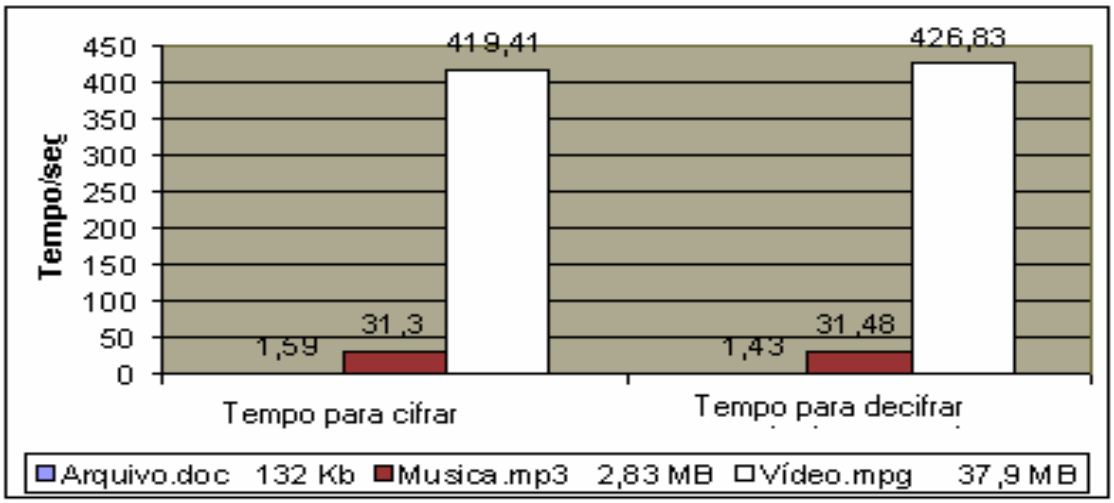

Figura 4.3. Tempo médio para cifrar e decifrar usando algoritmo Mars implementado na linguagem Java, usando um Celeron 466 Mhz.

Através dos testes feitos pode-se observar que o tempo para cifrar e decifrar não depende do arquivo que estamos utilizando, e sim do seu tamanho. O arquivo texto apresenta um tempo menor que o vídeo, ou a música por ter apenas 132Kb enquanto vídeo tem 2,83 MB.

Comparando o desempenho na linguagem C e o desempenho na linguagem Java, conclui-se que a implementação em Java gasta um tempo (tempo foi medido em segundos) maior que a linguagem C para cifrar ou decifrar, uma diferença, que acontece devido a C ser uma linguagem executável que consegue executar diretamente do sistema operacional e tem acesso direto a todos os recursos do sistema, incluindo memória e sistema de arquivos, enquanto Java precisa de um interpretador para executar, gastando assim mais tempo para executar que a linguagem C.

CONCLUSÃO

No presente trabalho conclui-se que a criptografia é muito importante nos dias atuais e que o algoritmo Mars, alvo deste trabalho apresenta um desempenho melhor quando implementado na linguagem C, comparando-o com a implementação na linguagem Java.

Java é uma linguagem interpretada que passa pelos seguintes passos até sua execução:

- O programa é criado no editor e salvo em disco;
- O compilador traduz o programa para bytecodes (linguagem entendida pelo interpretador Java) e armazena em disco;
- O carregador da classe coloca os bytecodes na memória principal;
- Verificador de bytecodes verificam se todos os bytecodes são validos;
- Interpretador lê os bytecodes e os traduz para uma linguagem que o computador entenda.

Para uma maior velocidade de execução na linguagem Java necessita-se uma maior espaço de memória, pois os bytecodes compilados, em geral, ficam três vezes maiores que o original.

O tamanho da chave é muito importante na criptografia, pois quanto maior a chave, maior a segurança observa-se no algoritmo. Mas o tamanho da chave influenciou na velocidade do algoritmo Mars apenas quando foi usada uma chave de 256 bits em computadores Pentiun e Pentiun II, para outros computadores e chaves de tamanho 128 e 192 o desempenho não apresenta uma diferença significante.

Para uma continuação deste trabalho sugere os seguintes trabalhos:

- A implementação em hardware do algoritmo Mars;

- Novos testes nos mesmos computadores, mudando apenas o tamanho da memória, para verificar se acontece alguma diferença no desempenho;
- Uma quantidade maior de testes nos computadores;
- Testes com chaves diferentes e análise do impacto .

REFERÊNCIAS BIBLIOGRÁFICAS

Araújo, G; ”News Generation”, publicado pela Rede National de ensino e Pesquisa em 1977; http://www.rnp.br/newsgen/9803/https.html, acesso março 2005.

Adriano, J; “Criptografia”; http://criptografia.rg3.net/, acesso março 2005.

“Aldeia NumaBoa”; editora do site: vovó Vicki, 1998 http://www.numaboa.com.br/criptografia/intro.php, acesso março 2005.

[IBM 99] IBM Corporation. Mars – a candidate cipher for AES. 1999.

Cunha, R. F; “Assinatura Digital”. Instituto Tecnológico de Aeronáutica.

Dellani,P.R., “O Algoritmo de Encriptação Mars e o Aes”, 2004.

Descrição,Comparações e Comentários”,http://www.lisha.ufsc.Br, acesso julho 2005.

Deitel, H.M., Deitel, P.J.; “ Java Como Programar ”, 4ª edição, Bookman 2003.

Fips46-2.Federal Information Processing Stantards Publication 46-2, Data Encryption Standard, Dezembro,1993.

Fips97, Federal Information Processing Stantards Publication 97, Advanced Encryption Stantards(AES, Novembro,2001).

Hinz, M.A.M., “Um esutdo descritivo de novos algoritmos de criptografia”, Universidade de Pelotas Instituto de Física e Matemática , Pelotas, 2000.

Loiola, M. A ., http://www.comp.pucpcaldas.br/~01550252544/java.html, acesso em outubro 2005.

Maia, L.P.,1998, “Monografia sobre criptografia baseada em Identidade”; http://www.linux.ime.usp.br/~cef/mac49904/monografias/rec/cesarse/monografia.html, acesso março 2005.

Oliveira, E.E.L; “Teste, Verificação, Comparação e Implementação de Algorítmos de Criptografia ”, Fundação Eurípdes Soares da Rocha, Marília, 2003.

Trinta,F.A.M.,Macedo, R. C; “Um estudo sobre criptografia e assinatura digital ”; http://www.di.ufpe.br/~flash/ais98/cripto/criptografia.htm, acesso em fevereiro 2005.

Pereira, F. D; “Um criptoprocessador VLIW para algorítmos criptográficos simétricos. Projeto e desenvolvimento em FPGAS. Tese de mestrado: Fundação Eurípdes Soares da Rocha, Marília 2004.

Rivest, R., The RC5 Encryption Algorithm, Fast Software Encryption, 2nd. International Workshop, Lec. Note in Comp.SCI.1008, pp86-96, Springer – Verlag,1995.

“Verdade Absoluta”; http://www.absoluta.org/cripty_h.htm, acesso março 2005.

Apêndice A

Algoritmo Mars implementado na linguagem Java

```
import java.io.*;
public class Mars{
private static long vk[] = new long[47];
public Mars(){}
private static long s_box[] =
{
   0x09d0c479l, 0x28c8ffe0l, 0x84aa6c39l, 0x9dad7287l, /* 0x000    */
   0x7dff9be3l, 0xd4268361l, 0xc96da1d4l, 0x7974cc93l,
   0x85d0582el, 0x2a4b5705l, 0x1ca16a62l, 0xc3bd279dl,
   0x0f1f25e5l, 0x5160372fl, 0xc695c1fbl, 0x4d7ff1e4l,
   0xae5f6bf4l, 0x0d72ee46l, 0xff23de8al, 0xb1cf8e83l, /* 0x010    */
   0xf14902e2l, 0x3e981e42l, 0x8bf53eb6l, 0x7f4bf8acl,
   0x83631f83l, 0x25970205l, 0x76afe784l, 0x3a7931d4l,
   0x4f846450l, 0x5c64c3f6l, 0x210a5f18l, 0xc6986a26l,
   0x28f4e826l, 0x3a60a81cl, 0xd340a664l, 0x7ea820c4l, /* 0x020    */
   0x526687c5l, 0x7eddd12bl, 0x32a11d1dl, 0x9c9ef086l,
   0x80f6e831l, 0xab6f04adl, 0x56fb9b53l, 0x8b2e095cl,
   0xb68556ael, 0xd2250b0dl, 0x294a7721l, 0xe21fb253l,
   0xae136749l, 0xe82aae86l, 0x93365104l, 0x99404a66l, /* 0x030    */
   0x78a784dcl, 0xb69ba84bl, 0x04046793l, 0x23db5c1el,
   0x46cae1d6l, 0x2fe28134l, 0x5a223942l, 0x1863cd5bl,
   0xc190c6e3l, 0x07dfb846l, 0x6eb88816l, 0x2d0dcc4al,
   0xa4ccae59l, 0x3798670dl, 0xcbfa9493l, 0x4f481d45l, /* 0x040    */
   0xeafc8ca8l, 0xdb1129d6l, 0xb0449e20l, 0x0f5407fbl,
   0x6167d9a8l, 0xd1f45763l, 0x4daa96c3l, 0x3bec5958l,
   0xababa014l, 0xb6ccd201l, 0x38d6279fl, 0x02682215l,
   0x8f376cd5l, 0x092c237el, 0xbfc56593l, 0x32889d2cl, /* 0x050    */
   0x854b3e95l, 0x05bb9b43l, 0x7dcd5dcdl, 0xa02e926cl,
   0xfae527e5l, 0x36a1c330l, 0x3412e1ael, 0xf257f462l,
   0x3c4f1d71l, 0x30a2e809l, 0x68e5f551l, 0x9c61ba44l,
   0x5ded0ab8l, 0x75ce09c8l, 0x9654f93el, 0x698c0ccal, /* 0x060    */
   0x243cb3e4l, 0x2b062b97l, 0x0f3b8d9el, 0x00e050dfl,
   0xfc5d6166l, 0xe35f9288l, 0xc079550dl, 0x0591aee8l,
   0x8e531e74l, 0x75fe3578l, 0x2f6d829al, 0xf60b21ael,
   0x95e8eb8dl, 0x6699486bl, 0x901d7d9bl, 0xfd6d6e31l, /* 0x070    */
   0x1090acefl, 0xe0670dd8l, 0xdab2e692l, 0xcd6d4365l,
   0xe5393514l, 0x3af345f0l, 0x6241fc4dl, 0x460da3a3l,
```

0x7bcf3729l, 0x8bf1d1e0l, 0x14aac070l, 0x1587ed55l,
0x3afd7d3el, 0xd2f29e01l, 0x29a9d1f6l, 0xefb10c53l, /* 0x080 */
0xcf3b870fl, 0xb414935cl, 0x664465edl, 0x024acac7l,
0x59a744c1l, 0x1d2936a7l, 0xdc580aa6l, 0xcf574ca8l,
0x040a7a10l, 0x6cd81807l, 0x8a98be4cl, 0xaccea063l,
0xc33e92b5l, 0xd1e0e03dl, 0xb322517el, 0x2092bd13l, /* 0x090 */
0x386b2c4al, 0x52e8dd58l, 0x58656dfbl, 0x50820371l,
0x41811896l, 0xe337ef7el, 0xd39fb119l, 0xc97f0df6l,
0x68fea01bl, 0xa150a6e5l, 0x55258962l, 0xeb6ff41bl,
0xd7c9cd7al, 0xa619cd9el, 0xbcf09576l, 0x2672c073l, /* 0x0a0 */
0xf003fb3cl, 0x4ab7a50bl, 0x1484126al, 0x487ba9b1l,
0xa64fc9c6l, 0xf6957d49l, 0x38b06a75l, 0xdd805fcdl,
0x63d094cfl, 0xf51c999el, 0x1aa4d343l, 0xb8495294l,
0xce9f8e99l, 0xbffcd770l, 0xc7c275ccl, 0x378453a7l, /* 0x0b0 */
0x7b21be33l, 0x397f41bdl, 0x4e94d131l, 0x92cc1f98l,
0x5915ea51l, 0x99f861b7l, 0xc9980a88l, 0x1d74fd5fl,
0xb0a495f8l, 0x614deed0l, 0xb5778eeal, 0x5941792dl,
0xfa90c1f8l, 0x33f824b4l, 0xc4965372l, 0x3ff6d550l, /* 0x0c0 */
0x4ca5fec0l, 0x8630e964l, 0x5b3fbbd6l, 0x7da26a48l,
0xb203231al, 0x04297514l, 0x2d639306l, 0x2eb13149l,
0x16a45272l, 0x532459a0l, 0x8e5f4872l, 0xf966c7d9l,
0x07128dc0l, 0x0d44db62l, 0xafc8d52dl, 0x06316131l, /* 0x0d0 */
0xd838e7cel, 0x1bc41d00l, 0x3a2e8c0fl, 0xea83837el,
0xb984737dl, 0x13ba4891l, 0xc4f8b949l, 0xa6d6acb3l,
0xa215cdcel, 0x8359838bl, 0x6bd1aa31l, 0xf579dd52l,
0x21b93f93l, 0xf5176781l, 0x187dfddel, 0xe94aeb76l, /* 0x0e0 */
0x2b38fd54l, 0x431de1dal, 0xab394825l, 0x9ad3048fl,
0xdfea32aal, 0x659473e3l, 0x623f7863l, 0xf3346c59l,
0xab3ab685l, 0x3346a90bl, 0x6b56443el, 0xc6de01f8l,
0x8d421fc0l, 0x9b0ed10cl, 0x88f1a1e9l, 0x54c1f029l, /* 0x0f0 */
0x7dead57bl, 0x8d7ba426l, 0x4cf5178al, 0x551a7ccal,
0x1a9a5f08l, 0xfcd651b9l, 0x25605182l, 0xe11fc6c3l,
0xb6fd9676l, 0x337b3027l, 0xb7c8eb14l, 0x9e5fd030l,

0x6b57e354l, 0xad913cf7l, 0x7e16688dl, 0x58872a69l, /* 0x100 */
0x2c2fc7dfl, 0xe389ccc6l, 0x30738df1l, 0x0824a734l,
0xe1797a8bl, 0xa4a8d57bl, 0x5b5d193bl, 0xc8a8309bl,
0x73f9a978l, 0x73398d32l, 0x0f59573el, 0xe9df2b03l,
0xe8a5b6c8l, 0x848d0704l, 0x98df93c2l, 0x720a1dc3l, /* 0x110 */
0x684f259al, 0x943ba848l, 0xa6370152l, 0x863b5ea3l,

0xd17b978bl, 0x6d9b58efl, 0x0a700dd4l, 0xa73d36bfl,
0x8e6a0829l, 0x8695bc14l, 0xe35b3447l, 0x933ac568l,
0x8894b022l, 0x2f511c27l, 0xddfbcc3cl, 0x006662b6l, /* 0x120 */
0x117c83fel, 0x4e12b414l, 0xc2bca766l, 0x3a2fec10l,
0xf4562420l, 0x55792e2al, 0x46f5d857l, 0xceda25cel,
0xc3601d3bl, 0x6c00ab46l, 0xefac9c28l, 0xb3c35047l,
0x611dfee3l, 0x257c3207l, 0xfdd58482l, 0x3b14d84fl, /* 0x130 */
0x23becb64l, 0xa075f3a3l, 0x088f8eadl, 0x07adf158l,
0x7796943cl, 0xfacabf3dl, 0xc09730cdl, 0xf7679969l,
0xda44e9edl, 0x2c854c12l, 0x35935fa3l, 0x2f057d9fl,
0x690624f8l, 0x1cb0bafdl, 0x7b0dbdc6l, 0x810f23bbl, /* 0x140 */
0xfa929a1al, 0x6d969a17l, 0x6742979bl, 0x74ac7d05l,
0x010e65c4l, 0x86a3d963l, 0xf907b5a0l, 0xd0042bd3l,
0x158d7d03l, 0x287a8255l, 0xbba8366fl, 0x096edc33l,
0x21916a7bl, 0x77b56b86l, 0x951622f9l, 0xa6c5e650l, /* 0x150 */
0x8cea17d1l, 0xcd8c62bcl, 0xa3d63433l, 0x358a68fdl,
0x0f9b9d3cl, 0xd6aa295bl, 0xfe33384al, 0xc000738el,
0xcd67eb2fl, 0xe2eb6dc2l, 0x97338b02l, 0x06c9f246l,
0x419cf1adl, 0x2b83c045l, 0x3723f18al, 0xcb5b3089l, /* 0x160 */
0x160bead7l, 0x5d494656l, 0x35f8a74bl, 0x1e4e6c9el,
0x000399bdl, 0x67466880l, 0xb4174831l, 0xacf423b2l,
0xca815ab3l, 0x5a6395e7l, 0x302a67c5l, 0x8bdb446bl,
0x108f8fa4l, 0x10223edal, 0x92b8b48bl, 0x7f38d0eel, /* 0x170 */
0xab2701d4l, 0x0262d415l, 0xaf224a30l, 0xb3d88abal,
0xf8b2c3afl, 0xdaf7ef70l, 0xcc97d3b7l, 0xe9614b6cl,
0x2baebff4l, 0x70f687cfl, 0x386c9156l, 0xce092ee5l,
0x01e87da6l, 0x6ce91e6al, 0xbb7bcc84l, 0xc7922c20l, /* 0x180 */
0x9d3b71fdl, 0x060e41c6l, 0xd7590f15l, 0x4e03bb47l,
0x183c198el, 0x63eeb240l, 0x2ddbf49al, 0x6d5cba54l,
0x923750afl, 0xf9e14236l, 0x7838162bl, 0x59726c72l,
0x81b66760l, 0xbb2926c1l, 0x48a0ce0dl, 0xa6c0496dl, /* 0x190 */
0xad43507bl, 0x718d496al, 0x9df057afl, 0x44b1bde6l,
0x054356dcl, 0xde7ced35l, 0xd51a138bl, 0x62088cc9l,
0x35830311l, 0xc96efca2l, 0x686f86ecl, 0x8e77cb68l,
0x63e1d6b8l, 0xc80f9778l, 0x79c491fdl, 0x1b4c67f2l, /* 0x1a0 */
0x72698d7dl, 0x5e368c31l, 0xf7d95e2el, 0xa1d3493fl,
0xdcd9433el, 0x896f1552l, 0x4bc4ca7al, 0xa6d1baf4l,
0xa5a96dccl, 0x0bef8b46l, 0xa169fda7l, 0x74df40b7l,
0x4e208804l, 0x9a756607l, 0x038e87c8l, 0x20211e44l, /* 0x1b0 */
0x8b7ad4bfl, 0xc6403f35l, 0x1848e36dl, 0x80bdb038l,

```
    0x1e62891cl, 0x643d2107l, 0xbf04d6f8l, 0x21092c8cl,
    0xf644f389l, 0x0778404el, 0x7b78adb8l, 0xa2c52d53l,
    0x42157abel, 0xa2253e2el, 0x7bf3f4ael, 0x80f594f9l, /* 0x1c0    */
    0x953194e7l, 0x77eb92edl, 0xb3816930l, 0xda8d9336l,
    0xbf447469l, 0xf26d9483l, 0xee6faed5l, 0x71371235l,
    0xde425f73l, 0xb4e59f43l, 0x7dbe2d4el, 0x2d37b185l,
    0x49dc9a63l, 0x98c39d98l, 0x1301c9a2l, 0x389b1bbfl, /* 0x1d0    */
    0x0c18588dl, 0xa421c1bal, 0x7aa3865cl, 0x71e08558l,
    0x3c5cfcaal, 0x7d239ca4l, 0x0297d9ddl, 0xd7dc2830l,
    0x4b37802bl, 0x7428ab54l, 0xaeee0347l, 0x4b3fbb85l,
    0x692f2f08l, 0x134e578el, 0x36d9e0bfl, 0xae8b5fcfl, /* 0x1e0    */
    0xedb93ecfl, 0x2b27248el, 0x170eb1efl, 0x7dc57fd6l,
    0x1e760f16l, 0xb1136601l, 0x864e1b9bl, 0xd7ea7319l,
    0x3ab871bdl, 0xcfa4d76fl, 0xe31bd782l, 0x0dbeb469l,
    0xabb96061l, 0x5370f85dl, 0xffb07e37l, 0xda30d0fbl, /* 0x1f0    */
    0xebc977b6l, 0x0b98b40fl, 0x3a4d0fe6l, 0xdf4fc26bl,
    0x159cf22al, 0xc298d6e2l, 0x2b78ef6al, 0x61a94ac0l,
    0xab561187l, 0x14eea0f0l, 0xdf0d4164l, 0x19af70eel
};

private static long l_key[] = new long [40];
private  String buff[] = new String [64];
private int nc;
private long a, b, c, d, l, m, r;
public long rotr(long x,long n)
{  return x>>n;
  }
public long rotl(long x,long n)
{  return x<<n;
   }

//#define f_mix(a,b,c,d)
public void f_mix(long a,long b,long c,long d)
{
      r = rotr((int)a, 8);
      b ^= s_box[(int)a & 255];
      b += s_box[((int)r & 255) + 256];
      r = rotr((int)a, 16);
      a  = rotr((int)a, 24);
      c += s_box[(int)r & 255];
```

```
        d ^= s_box[((int)a & 255) + 256];
}
//#define b_mix(a,b,c,d)
public void b_mix(long a,long b,long c,long d)
{
        r = rotl((int)a, 8);
        b ^= s_box[((int)a & 255) + 256];
        c -= s_box[(int)r & 255];
        r = rotl((int)a, 16);
        a  = rotl((int)a, 24);
        d -= s_box[((int)r & 255) + 256];
        d ^= s_box[(int)a & 255];
}
//#define f_ktr(a,b,c,d,i)
public void f_ktr(long a,long b,long c,long d,int i)
{
    m = a + l_key[i];
    a = rotl((int)a, 13);
    r = a * l_key[i + 1];
    l = s_box[(int)m & 511];
    r = rotl((int)r, 5);
    c += rotl((int)m, r);
    l ^= r;
    r = rotl((int)r, 5);
    l ^= r;
    d ^= r;
    b += rotl((int)l, r);
}
//#define r_ktr(a,b,c,d,i)
public void r_ktr(long a,long b,long c,long d,int i)
{
    r = a * l_key[i + 1];
    a = rotr((int)a, 13);
    m = a + l_key[i];
    l = s_box[(int)m & 511];
    r = rotl((int)r, 5);
    l ^= r;
    c -= rotl((int)m, r);
    r = rotl((int)r, 5);
    l ^= r;
```

```
    d ^= r;
    b -= rotl((int)l, r);
            }

public long gen_mask(long x)
{   long m;
    m = ((~x ^ (x >> 1)) & 0x7fffffffl);
    m &= ((m >> 1) & (m >> 2));
   m &= ((m >> 3) & (m >> 6));
   if(m==0)
       return 0;
    m <<= 1;
    m |= (m << 1);
    m |= (m << 2);
    m |= (m << 4);
    m |= (m << 1) & ~x & 0x80000000l;

    return (m & 0xfffffffcl);
}

public long[] set_key(final long in_key[], final long key_len)
{   long i, j, m, w;
   m = key_len / 32 - 1;
    for(i = j = 0; i < 39; ++i)
    {
      vk[(int)i + 7] = rotl(vk[(int)i] ^ vk[(int)i + 5], 3) ^ in_key[(int)j] ^ i;
      j = (j == m ? 0 : j + 1);
    }
    vk[46] = key_len / 32;
    for(j = 0; j < 7; ++j)
    {
        for(i = 1; i < 40; ++i)
          vk[(int)i + 7] = rotl(vk[(int)i + 7] + s_box[(int)vk[(int)i + 6] & 511], 9);
       vk[7] = rotl(vk[7] + s_box[(int)vk[46] & 511], 9);
    }
    for(i = j = 0; i < 40; ++i)
    {
       l_key[(int)j] = vk[(int)i + 7];

       j = (j < 33 ? j + 7 : j - 33);
```

```
    }
    for(i = 5; i < 37; i += 2)
    {
        w = l_key[(int)i] | 3;
        if(m == gen_mask(w))
            w ^= (rotl(s_box[265 + (int)(l_key[(int)i] & 3)], l_key[(int)i + 3] & 31) & (int)m);
        l_key[(int)i] = w;
    }
    return l_key;
}

public void encrypt(final byte in_blk[], byte out_blk[])
{   long  a, b, c, d, l, m, r;
    a = (long)in_blk[0] + l_key[0];
    b = (long)in_blk[1] + l_key[1];
    c = (long)in_blk[2] + l_key[2];
    d = (long)in_blk[3] + l_key[3];
   f_mix(a,b,c,d);
    a += d;
    f_mix(b,c,d,a);
    b += c;
    f_mix(c,d,a,b);
    f_mix(d,a,b,c);
    f_mix(a,b,c,d);
    a += d;
    f_mix(b,c,d,a);
    b += c;
    f_mix(c,d,a,b);
    f_mix(d,a,b,c);
    f_ktr(a,b,c,d, 4);
   f_ktr(b,c,d,a, 6);
   f_ktr(c,d,a,b, 8);
   f_ktr(d,a,b,c,10);
    f_ktr(a,b,c,d,12);
   f_ktr(b,c,d,a,14);
   f_ktr(c,d,a,b,16);
   f_ktr(d,a,b,c,18);
    f_ktr(a,d,c,b,20);
  f_ktr(b,a,d,c,22);
  f_ktr(c,b,a,d,24);
```

```
  f_ktr(d,c,b,a,26);
  f_ktr(a,d,c,b,28);
  f_ktr(b,a,d,c,30);
  f_ktr(c,b,a,d,32);
  f_ktr(d,c,b,a,34);
  b_mix(a,b,c,d);
  b_mix(b,c,d,a);
  c -= b;
   b_mix(c,d,a,b);
  d -= a;
   b_mix(d,a,b,c);
   b_mix(a,b,c,d);
   b_mix(b,c,d,a);
  c -= b;
   b_mix(c,d,a,b);
  d -= a;
   b_mix(d,a,b,c);
   out_blk[0] = (byte)(a - l_key[36]);
   out_blk[1] = (byte)(b - l_key[37]);
   out_blk[2] = (byte)(c - l_key[38]);
  out_blk[3] = (byte)(d - l_key[39]);
}

public void decrypt(final byte in_blk[], byte out_blk[])
{   long a, b, c, d, l, m, r;
    d = (long)in_blk[0] + l_key[36];
    c = (long)in_blk[1] + l_key[37];
    b = (long)in_blk[2] + l_key[38];
   a = (long)in_blk[3] + l_key[39];
   f_mix(a,b,c,d);
   a += d;
  f_mix(b,c,d,a);
  b += c;
  f_mix(c,d,a,b);
  f_mix(d,a,b,c);
  f_mix(a,b,c,d);
  a += d;
  f_mix(b,c,d,a);
  b += c;
  f_mix(c,d,a,b);
```

```
  f_mix(d,a,b,c);
  r_ktr(a,b,c,d,34);
  r_ktr(b,c,d,a,32);
  r_ktr(c,d,a,b,30);
  r_ktr(d,a,b,c,28);
  r_ktr(a,b,c,d,26);
  r_ktr(b,c,d,a,24);
  r_ktr(c,d,a,b,22);
  r_ktr(d,a,b,c,20);
  r_ktr(a,d,c,b,18);
  r_ktr(b,a,d,c,16);
  r_ktr(c,b,a,d,14);
  r_ktr(d,c,b,a,12);
  r_ktr(a,d,c,b,10);
  r_ktr(b,a,d,c, 8);
  r_ktr(c,b,a,d, 6);
  r_ktr(d,c,b,a, 4);
  b_mix(a,b,c,d);
  b_mix(b,c,d,a);
  c -= b;
  b_mix(c,d,a,b);
  d -= a;
  b_mix(d,a,b,c);
  b_mix(a,b,c,d);
  b_mix(b,c,d,a);
  c -= b;
  b_mix(c,d,a,b);
  d -= a;
  b_mix(d,a,b,c);
  out_blk[0] = (byte)(d - l_key[0]);
  out_blk[1] = (byte)(c - l_key[1]);
  out_blk[2] = (byte)(b - l_key[2]);
  out_blk[3] = (byte)(a - l_key[3]);
}

public static void main(String args[])
{
    Mars objeto = new Mars();

    // trocar no lugar do cap , colocar o arquivo de musica e video
```

```
        File inputFile = new File("cap.doc");
File encripFile = new File("encriptado.doc");
File decripFile = new File("decriptado.doc");
FileInputStream in = null;
FileOutputStream out = null;
        long chave[] = {0x01020304l,0x05060708l,0x090a0b0cl,0x0d0e0f10l};
        double inicio,fim;
        byte saida[] = new byte[4];
         objeto.set_key(chave, 128);
        try {
                in = new FileInputStream(inputFile);
        }
        catch (Exception e) {        }
   try {
                out = new FileOutputStream(encripFile);
        }
        catch (IOException e)
        {
                System.out.println("Erro: " + e);
        }
        int n;
        byte c[] = new byte[4];
        inicio = System.currentTimeMillis();
        try {
                while ((n = in.read(c)) != -1)
                {
                        byte lidos[] = new byte[n];
                        for (int contlidos = 0;contlidos <n; contlidos++)
                                lidos[contlidos] = c[contlidos];
                        objeto.encrypt(lidos,saida);
                        out.write(saida,0,saida.length);
                  }
                 in.close();
                 out.close();
          }
        catch (IOException e){
                }
        fim = System.currentTimeMillis();
        System.out.println("Tempo de Encriptação: " + ((fim - inicio)/1000)+" segundos");
try {
```

```
                in = new FileInputStream(encripFile);
                }
        catch (IOException e)
        {
                System.out.println("Erro: " + e);
        }
        try {
                out = new FileOutputStream(decripFile);
        }
        catch (IOException e)
        {
                System.out.println("Erro: " + e);
        }
        inicio = System.currentTimeMillis();
        try {
                while ((n = in.read(c)) != -1)
                {
                        byte lidos[] = new byte[n];
                        for (int contlidos = 0;contlidos <n; contlidos++)
                                lidos[contlidos] = c[contlidos];
                        objeto.decrypt(lidos,saida);
                        out.write(saida,0,saida.length);
                }
                in.close();
                out.close();
        }
        catch (IOException e){     }
        fim = System.currentTimeMillis();
        System.out.println("Tempo de Decriptação: " + ((fim - inicio)/1000)+" segundos");
        }
}
```

Anexo

Algoritmo Mars implementado na linguagem C

```
#include "time.h"
#include "std_defs.h"
static u4byte s_box[] =
{
    0x09d0c479, 0x28c8ffe0, 0x84aa6c39, 0x9dad7287, /* 0x000    */
    0x7dff9be3, 0xd4268361, 0xc96da1d4, 0x7974cc93,
    0x85d0582e, 0x2a4b5705, 0x1ca16a62, 0xc3bd279d,
    0x0f1f25e5, 0x5160372f, 0xc695c1fb, 0x4d7ff1e4,
    0xae5f6bf4, 0x0d72ee46, 0xff23de8a, 0xb1cf8e83, /* 0x010    */
    0xf14902e2, 0x3e981e42, 0x8bf53eb6, 0x7f4bf8ac,
    0x83631f83, 0x25970205, 0x76afe784, 0x3a7931d4,
    0x4f846450, 0x5c64c3f6, 0x210a5f18, 0xc6986a26,
    0x28f4e826, 0x3a60a81c, 0xd340a664, 0x7ea820c4, /* 0x020    */
    0x526687c5, 0x7eddd12b, 0x32a11d1d, 0x9c9ef086,
    0x80f6e831, 0xab6f04ad, 0x56fb9b53, 0x8b2e095c,
    0xb68556ae, 0xd2250b0d, 0x294a7721, 0xe21fb253,
    0xae136749, 0xe82aae86, 0x93365104, 0x99404a66, /* 0x030    */
    0x78a784dc, 0xb69ba84b, 0x04046793, 0x23db5c1e,
    0x46cae1d6, 0x2fe28134, 0x5a223942, 0x1863cd5b,
    0xc190c6e3, 0x07dfb846, 0x6eb88816, 0x2d0dcc4a,
    0xa4ccae59, 0x3798670d, 0xcbfa9493, 0x4f481d45, /* 0x040    */
    0xeafc8ca8, 0xdb1129d6, 0xb0449e20, 0x0f5407fb,
    0x6167d9a8, 0xd1f45763, 0x4daa96c3, 0x3bec5958,
    0xababa014, 0xb6ccd201, 0x38d6279f, 0x02682215,
    0x8f376cd5, 0x092c237e, 0xbfc56593, 0x32889d2c, /* 0x050    */
    0x854b3e95, 0x05bb9b43, 0x7dcd5dcd, 0xa02e926c,
    0xfae527e5, 0x36a1c330, 0x3412e1ae, 0xf257f462,
    0x3c4f1d71, 0x30a2e809, 0x68e5f551, 0x9c61ba44,
    0x5ded0ab8, 0x75ce09c8, 0x9654f93e, 0x698c0cca, /* 0x060    */
    0x243cb3e4, 0x2b062b97, 0x0f3b8d9e, 0x00e050df,
    0xfc5d6166, 0xe35f9288, 0xc079550d, 0x0591aee8,
    0x8e531e74, 0x75fe3578, 0x2f6d829a, 0xf60b21ae,
    0x95e8eb8d, 0x6699486b, 0x901d7d9b, 0xfd6d6e31, /* 0x070    */
    0x1090acef, 0xe0670dd8, 0xdab2e692, 0xcd6d4365,
    0xe5393514, 0x3af345f0, 0x6241fc4d, 0x460da3a3,
    0x7bcf3729, 0x8bf1d1e0, 0x14aac070, 0x1587ed55,
    0x3afd7d3e, 0xd2f29e01, 0x29a9d1f6, 0xefb10c53, /* 0x080    */
    0xcf3b870f, 0xb414935c, 0x664465ed, 0x024acac7,
```

0x59a744c1, 0x1d2936a7, 0xdc580aa6, 0xcf574ca8,
0x040a7a10, 0x6cd81807, 0x8a98be4c, 0xaccea063,
0xc33e92b5, 0xd1e0e03d, 0xb322517e, 0x2092bd13, /* 0x090 */
0x386b2c4a, 0x52e8dd58, 0x58656dfb, 0x50820371,
0x41811896, 0xe337ef7e, 0xd39fb119, 0xc97f0df6,
0x68fea01b, 0xa150a6e5, 0x55258962, 0xeb6ff41b,
0xd7c9cd7a, 0xa619cd9e, 0xbcf09576, 0x2672c073, /* 0x0a0 */
0xf003fb3c, 0x4ab7a50b, 0x1484126a, 0x487ba9b1,
0xa64fc9c6, 0xf6957d49, 0x38b06a75, 0xdd805fcd,
0x63d094cf, 0xf51c999e, 0x1aa4d343, 0xb8495294,
0xce9f8e99, 0xbffcd770, 0xc7c275cc, 0x378453a7, /* 0x0b0 */
0x7b21be33, 0x397f41bd, 0x4e94d131, 0x92cc1f98,
0x5915ea51, 0x99f861b7, 0xc9980a88, 0x1d74fd5f,
0xb0a495f8, 0x614deed0, 0xb5778eea, 0x5941792d,
0xfa90c1f8, 0x33f824b4, 0xc4965372, 0x3ff6d550, /* 0x0c0 */
0x4ca5fec0, 0x8630e964, 0x5b3fbbd6, 0x7da26a48,
0xb203231a, 0x04297514, 0x2d639306, 0x2eb13149,
0x16a45272, 0x532459a0, 0x8e5f4872, 0xf966c7d9,
0x07128dc0, 0x0d44db62, 0xafc8d52d, 0x06316131, /* 0x0d0 */
0xd838e7ce, 0x1bc41d00, 0x3a2e8c0f, 0xea83837e,
0xb984737d, 0x13ba4891, 0xc4f8b949, 0xa6d6acb3,
0xa215cdce, 0x8359838b, 0x6bd1aa31, 0xf579dd52,
0x21b93f93, 0xf5176781, 0x187dfdde, 0xe94aeb76, /* 0x0e0 */
0x2b38fd54, 0x431de1da, 0xab394825, 0x9ad3048f,
0xdfea32aa, 0x659473e3, 0x623f7863, 0xf3346c59,
0xab3ab685, 0x3346a90b, 0x6b56443e, 0xc6de01f8,
0x8d421fc0, 0x9b0ed10c, 0x88f1a1e9, 0x54c1f029, /* 0x0f0 */
0x7dead57b, 0x8d7ba426, 0x4cf5178a, 0x551a7cca,
0x1a9a5f08, 0xfcd651b9, 0x25605182, 0xe11fc6c3,
0xb6fd9676, 0x337b3027, 0xb7c8eb14, 0x9e5fd030,

0x6b57e354, 0xad913cf7, 0x7e16688d, 0x58872a69, /* 0x100 */
0x2c2fc7df, 0xe389ccc6, 0x30738df1, 0x0824a734,
0xe1797a8b, 0xa4a8d57b, 0x5b5d193b, 0xc8a8309b,
0x73f9a978, 0x73398d32, 0x0f59573e, 0xe9df2b03,
0xe8a5b6c8, 0x848d0704, 0x98df93c2, 0x720a1dc3, /* 0x110 */
0x684f259a, 0x943ba848, 0xa6370152, 0x863b5ea3,
0xd17b978b, 0x6d9b58ef, 0x0a700dd4, 0xa73d36bf,
0x8e6a0829, 0x8695bc14, 0xe35b3447, 0x933ac568,
0x8894b022, 0x2f511c27, 0xddfbcc3c, 0x006662b6, /* 0x120 */

0x117c83fe, 0x4e12b414, 0xc2bca766, 0x3a2fec10,
0xf4562420, 0x55792e2a, 0x46f5d857, 0xceda25ce,
0xc3601d3b, 0x6c00ab46, 0xefac9c28, 0xb3c35047,
0x611dfee3, 0x257c3207, 0xfdd58482, 0x3b14d84f, /* 0x130 */
0x23becb64, 0xa075f3a3, 0x088f8ead, 0x07adf158,
0x7796943c, 0xfacabf3d, 0xc09730cd, 0xf7679969,
0xda44e9ed, 0x2c854c12, 0x35935fa3, 0x2f057d9f,
0x690624f8, 0x1cb0bafd, 0x7b0dbdc6, 0x810f23bb, /* 0x140 */
0xfa929a1a, 0x6d969a17, 0x6742979b, 0x74ac7d05,
0x010e65c4, 0x86a3d963, 0xf907b5a0, 0xd0042bd3,
0x158d7d03, 0x287a8255, 0xbba8366f, 0x096edc33,
0x21916a7b, 0x77b56b86, 0x951622f9, 0xa6c5e650, /* 0x150 */
0x8cea17d1, 0xcd8c62bc, 0xa3d63433, 0x358a68fd,
0x0f9b9d3c, 0xd6aa295b, 0xfe33384a, 0xc000738e,
0xcd67eb2f, 0xe2eb6dc2, 0x97338b02, 0x06c9f246,
0x419cf1ad, 0x2b83c045, 0x3723f18a, 0xcb5b3089, /* 0x160 */
0x160bead7, 0x5d494656, 0x35f8a74b, 0x1e4e6c9e,
0x000399bd, 0x67466880, 0xb4174831, 0xacf423b2,
0xca815ab3, 0x5a6395e7, 0x302a67c5, 0x8bdb446b,
0x108f8fa4, 0x10223eda, 0x92b8b48b, 0x7f38d0ee, /* 0x170 */
0xab2701d4, 0x0262d415, 0xaf224a30, 0xb3d88aba,
0xf8b2c3af, 0xdaf7ef70, 0xcc97d3b7, 0xe9614b6c,
0x2baebff4, 0x70f687cf, 0x386c9156, 0xce092ee5,
0x01e87da6, 0x6ce91e6a, 0xbb7bcc84, 0xc7922c20, /* 0x180 */
0x9d3b71fd, 0x060e41c6, 0xd7590f15, 0x4e03bb47,
0x183c198e, 0x63eeb240, 0x2ddbf49a, 0x6d5cba54,
0x923750af, 0xf9e14236, 0x7838162b, 0x59726c72,
0x81b66760, 0xbb2926c1, 0x48a0ce0d, 0xa6c0496d, /* 0x190 */
0xad43507b, 0x718d496a, 0x9df057af, 0x44b1bde6,
0x054356dc, 0xde7ced35, 0xd51a138b, 0x62088cc9,
0x35830311, 0xc96efca2, 0x686f86ec, 0x8e77cb68,
0x63e1d6b8, 0xc80f9778, 0x79c491fd, 0x1b4c67f2, /* 0x1a0 */
0x72698d7d, 0x5e368c31, 0xf7d95e2e, 0xa1d3493f,
0xdcd9433e, 0x896f1552, 0x4bc4ca7a, 0xa6d1baf4,
0xa5a96dcc, 0x0bef8b46, 0xa169fda7, 0x74df40b7,
0x4e208804, 0x9a756607, 0x038e87c8, 0x20211e44, /* 0x1b0 */
0x8b7ad4bf, 0xc6403f35, 0x1848e36d, 0x80bdb038,
0x1e62891c, 0x643d2107, 0xbf04d6f8, 0x21092c8c,
0xf644f389, 0x0778404e, 0x7b78adb8, 0xa2c52d53,
0x42157abe, 0xa2253e2e, 0x7bf3f4ae, 0x80f594f9, /* 0x1c0 */

```
    0x953194e7, 0x77eb92ed, 0xb3816930, 0xda8d9336,
    0xbf447469, 0xf26d9483, 0xee6faed5, 0x71371235,
    0xde425f73, 0xb4e59f43, 0x7dbe2d4e, 0x2d37b185,
    0x49dc9a63, 0x98c39d98, 0x1301c9a2, 0x389b1bbf, /* 0x1d0    */
    0x0c18588d, 0xa421c1ba, 0x7aa3865c, 0x71e08558,
    0x3c5cfcaa, 0x7d239ca4, 0x0297d9dd, 0xd7dc2830,
    0x4b37802b, 0x7428ab54, 0xaeee0347, 0x4b3fbb85,
    0x692f2f08, 0x134e578e, 0x36d9e0bf, 0xae8b5fcf, /* 0x1e0    */
    0xedb93ecf, 0x2b27248e, 0x170eb1ef, 0x7dc57fd6,
    0x1e760f16, 0xb1136601, 0x864e1b9b, 0xd7ea7319,
    0x3ab871bd, 0xcfa4d76f, 0xe31bd782, 0x0dbeb469,
    0xabb96061, 0x5370f85d, 0xffb07e37, 0xda30d0fb, /* 0x1f0    */
    0xebc977b6, 0x0b98b40f, 0x3a4d0fe6, 0xdf4fc26b,
    0x159cf22a, 0xc298d6e2, 0x2b78ef6a, 0x61a94ac0,
    0xab561187, 0x14eea0f0, 0xdf0d4164, 0x19af70ee
};

static u4byte vk[47] =
{
    0x09d0c479, 0x28c8ffe0, 0x84aa6c39, 0x9dad7287, 0x7dff9be3, 0xd4268361,
    0xc96da1d4
};
static u4byte   l_key[40];
#define f_mix(a,b,c,d)              \
        r = rotr(a, 8);             \
        b ^= s_box[a & 255];        \
        b += s_box[(r & 255) + 256];    \
        r = rotr(a, 16);            \
        a  = rotr(a, 24);           \
        c += s_box[r & 255];        \
        d ^= s_box[(a & 255) + 256]
#define b_mix(a,b,c,d)              \
        r = rotl(a, 8);             \
        b ^= s_box[(a & 255) + 256];    \
        c -= s_box[r & 255];        \
        r = rotl(a, 16);            \
        a  = rotl(a, 24);           \
        d -= s_box[(r & 255) + 256];    \
        d ^= s_box[a & 255]
#define f_ktr(a,b,c,d,i)    \
```

```
    m = a + l_key[i];       \
    a = rotl(a, 13);        \
    r = a * l_key[i + 1];   \
    l = s_box[m & 511];     \
    r = rotl(r, 5);         \
    c += rotl(m, r);        \
    l ^= r;                 \
    r = rotl(r, 5);         \
    l ^= r;                 \
    d ^= r;                 \
    b += rotl(l, r)
#define r_ktr(a,b,c,d,i)    \
    r = a * l_key[i + 1];   \
    a = rotr(a, 13);        \
    m = a + l_key[i];       \
    l = s_box[m & 511];     \
    r = rotl(r, 5);         \
    l ^= r;                 \
    c -= rotl(m, r);        \
    r = rotl(r, 5);         \
    l ^= r;                 \
    d ^= r;                 \
    b -= rotl(l, r)
u4byte gen_mask(u4byte x)
{   u4byte  m;
    m = (~x ^ (x >> 1)) & 0x7fffffff;
    m &= (m >> 1) & (m >> 2); m &= (m >> 3) & (m >> 6);
    if(!m)  /* return if mask is empty - no key fixing needed   */
            /* is this early return worthwhile?                 */
        return 0;
    m <<= 1; m |= (m << 1); m |= (m << 2); m |= (m << 4);
    m |= (m << 1) & ~x & 0x80000000;
    return m & 0xfffffffc;
};
u4byte *set_key(const u4byte in_key[], const u4byte key_len)
{   u4byte  i, j, m, w;
    m = key_len / 32 - 1;
    for(i = j = 0; i < 39; ++i)
    {
      vk[i + 7] = rotl(vk[i] ^ vk[i + 5], 3) ^ in_key[j] ^ i;
```

```
    j = (j == m ? 0 : j + 1);
  }
  vk[46] = key_len / 32;
  for(j = 0; j < 7; ++j)
  {
     for(i = 1; i < 40; ++i)
       vk[i + 7] = rotl(vk[i + 7] + s_box[vk[i + 6] & 511], 9);
    vk[7] = rotl(vk[7] + s_box[vk[46] & 511], 9);
  }
  for(i = j = 0; i < 40; ++i)
  {
    l_key[j] = vk[i + 7];
    j = (j < 33 ? j + 7 : j - 33);
  }
  for(i = 5; i < 37; i += 2)
  {
    w = l_key[i] | 3;
    if(m = gen_mask(w))
      w ^= (rotl(s_box[265 + (l_key[i] & 3)], l_key[i + 3] & 31) & m);
    l_key[i] = w;
  }
  return l_key;
};
void encrypt(const u4byte in_blk[4], u4byte out_blk[4])
{  u4byte  a, b, c, d, l, m, r;
  a = in_blk[0] + l_key[0]; b = in_blk[1] + l_key[1];
  c = in_blk[2] + l_key[2]; d = in_blk[3] + l_key[3];
  f_mix(a,b,c,d); a += d;
  f_mix(b,c,d,a); b += c;
  f_mix(c,d,a,b);
  f_mix(d,a,b,c);
  f_mix(a,b,c,d); a += d;
  f_mix(b,c,d,a); b += c;
  f_mix(c,d,a,b);
  f_mix(d,a,b,c);
  f_ktr(a,b,c,d, 4); f_ktr(b,c,d,a, 6); f_ktr(c,d,a,b, 8); f_ktr(d,a,b,c,10);
  f_ktr(a,b,c,d,12); f_ktr(b,c,d,a,14); f_ktr(c,d,a,b,16); f_ktr(d,a,b,c,18);
  f_ktr(a,d,c,b,20); f_ktr(b,a,d,c,22); f_ktr(c,b,a,d,24); f_ktr(d,c,b,a,26);
  f_ktr(a,d,c,b,28); f_ktr(b,a,d,c,30); f_ktr(c,b,a,d,32); f_ktr(d,c,b,a,34);
  b_mix(a,b,c,d);
```

```
    b_mix(b,c,d,a); c -= b;
    b_mix(c,d,a,b); d -= a;
    b_mix(d,a,b,c);
    b_mix(a,b,c,d);
    b_mix(b,c,d,a); c -= b;
    b_mix(c,d,a,b); d -= a;
    b_mix(d,a,b,c);
    out_blk[0] = a - l_key[36]; out_blk[1] = b - l_key[37];
    out_blk[2] = c - l_key[38]; out_blk[3] = d - l_key[39];
};

void decrypt(const u4byte in_blk[4], u4byte out_blk[4])
{   u4byte  a, b, c, d, l, m, r;
    d = in_blk[0] + l_key[36]; c = in_blk[1] + l_key[37];
    b = in_blk[2] + l_key[38]; a = in_blk[3] + l_key[39];
    f_mix(a,b,c,d); a += d;
    f_mix(b,c,d,a); b += c;
    f_mix(c,d,a,b);
    f_mix(d,a,b,c);
    f_mix(a,b,c,d); a += d;
    f_mix(b,c,d,a); b += c;
    f_mix(c,d,a,b);
    f_mix(d,a,b,c);
    r_ktr(a,b,c,d,34); r_ktr(b,c,d,a,32); r_ktr(c,d,a,b,30); r_ktr(d,a,b,c,28);
    r_ktr(a,b,c,d,26); r_ktr(b,c,d,a,24); r_ktr(c,d,a,b,22); r_ktr(d,a,b,c,20);
    r_ktr(a,d,c,b,18); r_ktr(b,a,d,c,16); r_ktr(c,b,a,d,14); r_ktr(d,c,b,a,12);
    r_ktr(a,d,c,b,10); r_ktr(b,a,d,c, 8); r_ktr(c,b,a,d, 6); r_ktr(d,c,b,a, 4);
    b_mix(a,b,c,d);
    b_mix(b,c,d,a); c -= b;
    b_mix(c,d,a,b); d -= a;
    b_mix(d,a,b,c);
    b_mix(a,b,c,d);
    b_mix(b,c,d,a); c -= b;
    b_mix(c,d,a,b); d -= a;
    b_mix(d,a,b,c);
    out_blk[0] = d - l_key[0]; out_blk[1] = c - l_key[1];
    out_blk[2] = b - l_key[2]; out_blk[3] = a - l_key[3];
}
#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>
unsigned char buff[64];                 //buffer onde vou guardando o que leio
int nc;                                         //numero corrente de posições preenchidas no buffer
int modo;  //variavel que indica se vou cifrar ou decifrar
void adiciona(unsigned char * origem, unsigned char * destino, int pos, int n);
//nbytesretornado  = update (origem, nbytes, destino)
int update(unsigned char * bloco, int n, unsigned char * destino) //
{
            int i;
   unsigned char saida[16];
   int nretorno=0;
            for(i=0;i<n;i++)
            {
                        buff[nc] = bloco[i];
                        nc++;
       //vou lendo o texto ate preencher mei buffer,quando nc =16 entao buffer cheio
                        if(nc == 16) //blocos de 128 bits (16 * 8)
                        {
          if(modo == 0)
            encrypt((u4byte *)buff, (u4byte *)saida); // chamar funcao para criptografia
              // o que criptografo   //pra onde vai
          else
            decrypt((u4byte *)buff, (u4byte *)saida); // chamar funcao para decriptografia
          nc = 0; //agora volto pra posição zero do buffer
          adiciona(saida, destino, nretorno, 16);
          nretorno += 16;
                        }
            }
   return nretorno;
}
void adiciona(unsigned char * origem, unsigned char * destino, int pos, int n)
{
    int i;
    for (i=0;i<n;i++)
    {
       destino[pos+i] = origem[i];
    }
}
/* ------------------- void doFinal(bloco) ------------------- **
** Finaliza a operação, acrescentando o padding               */
```

```
//Essa função acrescenta zero ao buffer quando ele ainda não estiver cheio
//ele so pode criptografar ou decriptografar estando cheio
void doFinal( unsigned char * retorno)
{
	int idebug;
	int i;
	int npad;
 unsigned char pad[64] = {
	  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
	  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
	  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
	};
	npad = 16 - nc;
	if (npad == 0) npad = 16;
	update(pad,npad,retorno);
}
void cifraArquivo(char * origem, char * destino)
{
	FILE * fp;
	FILE * fd;
	unsigned char vet[1024], retorno[1024];
	int aux;
	int nbr;
	float inicio,fim; //variáveis para medir o tempo
	fp = fopen(origem, "rb"); //abre o arquivo para leitura
	if (fp == NULL) // se for nulo não foi possivel abrir o arquivo
	{
		printf("Erro ao abrir o arquivo para leitura");return;
	}
	fd = fopen(destino, "wb"); //abre o arquivo para escrita
	if (fd == NULL) // se for nulo não foi possivel abrir o arquivo
	{
		printf("Erro ao abrir o arquivo para escrita");return;
	}
	inicio = clock(); //inicio da contagem do tempo
	while(!feof(fp))
	{
		aux = fread(vet,sizeof(char),1024,fp); // lê do arquivo. Formato:
      // nbyteslidos = fread(enderecodedestino, tamanhodaestrutura, nvezes, ponteirodoarquivo);
		nbr = update(vet,aux,retorno); // chama a função update. Formato:
```

```
        //nbytesretornado  = update (origem, nbytes, destino)
        fwrite(retorno,sizeof(char),nbr,fd); // escreve no arquivo destino. Formato:
        //fwrite(enderecoorigem, tamanhoestrutura, nvezes, ponteirodoarquivo);
            }
            doFinal(retorno); //se sobrar alguma coisa no buffer,então uso doFinal
    fwrite(retorno,sizeof(char),16,fd); //grava os ultimos bytes processados
            fim = clock(); //fim da contagem do tempo
            printf("\nTempo: %.5f segundos",  (fim - inicio)/CLOCKS_PER_SEC); //mostra o tempo
    fclose(fp); // fecha o arquivo
    fclose(fd); // fecha o arquivo
}
int main(int argc, char ** argv)
{
char nomeorig[15];
char nomedestino[15];
   u4byte chave[] = {0x01020304,0x05060708,0x090a0b0c,0x0d0e0f10};
   set_key(chave, 128);
   modo = 0; //criptografa
  printf("digite nome do original: ");
  scanf("%s", &nomeorig);
  printf("digite nome do destino: ");
  scanf("%s", &nomedestino);
  printf("diigite o modo (0-criptografar, 1- decriptografar)");
  scanf("%d", &modo);
   if (modo==0)
      printf("encriptando...");
   else
      printf("decriptando...");
   cifraArquivo(nomeorig,nomedestino);
   system("pause");
}
```