FUNDAÇÃO DE ENSINO “EURIPIDES SOARES DA ROCHA”
CENTRO UNIVERSITARIO “EURIPIDES DE MARILIA” – UNIVEM
BACHARELADO EM CIENCIA DA COMPUTAÇÃO

FERNANDO LAZANHA

IMPLEMENTAÇÃO E DESEMPENHO DO ALGORITMO CRIPTOGRÁFICO “SERPENT”

MARILIA
2005

FERNANDO LAZANHA

IMPLEMENTAÇÃO E DESEMPENHO DO ALGORITMO CRIPTOGRÁFICO “SERPENT”.

Trabalho de conclusão de curso apresentado ao Centro Universitário Eurípides de Marilia, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do titulo de bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Edward David Moreno Ordonez

MARILIA
2005

LAZANHA, Fernando

Implementação em Software do Algoritmo Criptográfico “Serpent”/ Fernando Lazanha; orientador: Edward David Moreno Ordonez. Marilia, SP: [85], 2005

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

1.Criptografia 2.Algoritmos Simétricos 3.Serpent
4. Desempenho em Linguagens C e Java

CDD: 005.1

FERNANDO LAZANHA

IMPLEMENTAÇÃO E DESEMPENHO DO ALGORITMO CRIPTOGRÁFICO “SERPENT”

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

Resultado:_____

ORIENTADOR: Prof. Dr. Edward David

1º EXAMINADOR: _____

2º EXAMINADOR: _____

Marilia, 22 de Novembro de 2005

Dedico este trabalho ao Sr. Geraldo Lazanha, meu pai, que sempre acreditou em mim, e me apoiou em todas as minhas decisões. Obrigado....

AGRADECIMENTOS

Acima de tudo, a Deus, por me conceder o privilégio de ter saúde, alegria e inteligência para viver a vida, todo dia, com amor e paz.

Aos meus pais, que muitas vezes renunciaram a seus sonhos, para que nós realizássemos os nossos.

Aos meus irmãos, sempre muito atenciosos.

Aos companheiros de trabalho.

A todos, que direta ou indiretamente me ajudaram na conclusão de mais essa etapa da minha vida .

Meu orientador, Profº. Edward, pela paciência e confiança depositada em mim

Aos Professores da UNIVEM, que admiro a escolha profissional pelo ensino.

Aos meus amigos, que na alegria e na tristeza, sempre estiveram ao meu lado.

Ao meu grande amigo, Fernando Macedo Nogueira de Souza, grande companheiro.

E em especial, a Larissa Bilar que acima de tudo foi uma grande amiga e que sem ela, nada disso seria possível.

"Grandes realizações são possíveis quando se dá importância aos pequenos começos"

(Lao Tzu)

SUMARIO

LAZANHA, Fernando e. Implementação e Desempenho do Algoritmo Criptográfico “Serpent”. 2005. nº f. 85 Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marilia – Fundação de Ensino Eurípides Soares da Rocha

RESUMO

Neste trabalho foi feito um estudo sobre o mundo da criptologia, como história, algoritmos mais usados e implementação. Mais precisamente, dando ênfase ao algoritmo criptográfico “Serpent”.
Será utilizado um modelo do “Serpent”, que foi implementado na linguagem “C” e extraído detalhes da implementação desse algoritmo, logo após, usamos esses conceitos e o implementaremos nas linguagens de programação “C”, “JAVA. Será usado este algoritmo como base nas aplicações de cifragem e decifragem, e serão verificados seus desempenhos e velocidade através de gráficos”.

Palavras-Chaves: Criptografia. Algoritmos Simétricos. Serpent. Desempenho em Linguagens C e Java

LAZANHA, Fernando e. Implementação em Software do Algoritmo Criptográfico “Serpent”. 2005. nº f. 85 Trabalho de Conclusão de Curso (graduação em Ciência da Computação) – Centro Universitário Eurípides de Marilia – Fundação de Ensino Eurípides Soares da Rocha

ABSTRACT

In this final graduate labor, will be done a study about the world of cryptology like: history, usual algorithms and implementation. Most exactly, giving emphasis to the cryptographic algorithm “Serpent”.
For example, we´ll use a “Serpent” model that was implemented in the “C” language and we´ll extract some details from the implementation of this algorithm. After that, we´ll use this concepts and implement it in the “C”, “JAVA”. This algorithm will be used as base at the applications of cipherment and decipherment, and its performance and speed will be verified through the graphics.

Key-Words: Criptografy. Symetrics Algorithms. Serpent. Performace at language C and Java

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DES: Data Encryption Standard

NSA: National Security Agency

BCPL: Basic Combined Programming Language

MIT: Massachusetts Institute of Technology

NIST: National Institute of Standards and Technology

AES: Advanced Encryption Standard

APIs: Aplications Programming Interfaces

CAPITULO 1

HISTORIA DA CRIPTOLOGIA

1.1 - Introdução

No capitulo 1, apresentaremos um breve histórico sobre a criptografia, será feita uma introdução geral sobre os assuntos abordados, passando desde as suas primeiras utilizações até os dias de hoje.

1.2 – Um Breve Histórico Sobre a Criptografia

A história da criptologia[1] é um passeio no campo da criatividade humana [WWW2]. A criptologia foi usada por governantes e pelo povo, em épocas de guerra e em épocas de paz. A criptologia faz parte da história humana porque sempre houve fórmulas secretas, informações confidenciais e interesses dos mais diversos que não deveriam cair no domínio público ou na mão de inimigos [WWW1].

A criptologia é a arte ou ciência de escrever em cifra ou em códigos, de forma a permitir que somente o destinatário decifre e compreenda, ela é composta pela criptografia e pela criptoanálise conforme afirma [Pereira,2005].

Criptoanálise é a arte ou ciência de determinar a chave ou decifrar mensagens sem conhecer a chave. Uma tentativa de criptoanálise é chamada de ataque [Pereira,2005].

A historia da comunicação e a historia da criptologia estão intimamente ligadas desde o inicio, pois desde que o homem se comunica escrita e oralmente, existem muitas outras formas de transportar informações, como o telegrafo da selva, tambores, sinais de fumaça e outros. Esses métodos não são cifrados, mas precisam de uma pessoa que saiba enviá-los e outra para recebê-los[WWW1].

A criptografia surgiu há muito tempo, pois bastou o homem aprender a escrever para começar a “esconder” seus escritos não foram usados códigos ou sistemas matemáticos, mas sim a grande criatividade humana foi a base para a criação da criptologia.

A primeira forma de comunicação militar conhecida foi o bastão de Licurgo ou scytale (Figura 01), ela era uma cifra de transposição[2], que consistia em um bastão no qual era enrolada uma tira, como se mostra na figura 1.1, ao enrolar a tira, o remetente escrevia a mensagem de interesse e depois a desenrolava, deixando a mensagem toda embaralhada, o mensageiro usava a tira como cinto com as letras voltadas para dentro, ao receber a mensagem, o destinatário enrola novamente a tira em um bastão com a mesma espessura e tem-se a mensagem original.

---

[1] Tem origem no Grego (*kryptos* significa oculto, escondido, secreto; *graphos* significa escrever, grafar.)
[2] posição das letras do texto original é mudada, sem qualquer alteração no seu valor normal ou convencional

Figura 1.1 Bastão de Licurgo ou Scytale [WWW1]

A evolução foi lenta porém constante. Um primeiro exemplo de escrita cifrada acontece em 1900 a.C. em uma vila egípcia perto do Rio Nilo, chamada Menet Khufu [WWW1]. Um homem chamado Hhnumhotep II, que era arquiteto do faraó Amenemhet II, construiu alguns monumentos, só que esses tesouros precisavam ser documentados, escritos em tabletes de argila (Figura 1.2), mas não podiam cair em domínio publico, foi quando o escriba[3] de Khnumhotep II teve a idéia de substituir algumas palavras ou trechos destes tabletes, caso os escritos fossem roubados os ladrões se perderiam e ficariam presos nas catacumbas e morreriam de fome [WWW1].

Figura 1.2 Tabletes de Argila [WWW1]

[3] Escriba: pessoa da antiguidade que dominava a escrita

Com o passar do tempo, a humanidade foi incorporando a criptografia no seu dia a dia, pois com o domínio da escrita e da matemática, sistemas cada vez mais sofisticados de cifras iam surgindo e com isso estudos mais aprofundados na criptoanálise. Então, começaram a surgir máquinas e dispositivos mais elaborados e juntamente com os primeiros meios de comunicação a distância, o homem já demonstrou um interesse e uma necessidade maior pela criptologia[WWW1].

Com o surgimento dos primeiros sistemas de comunicação à distância, por serem sistemas abertos, deram um novo impulso à criptografia. Mas com o aparecimento de novas tecnologias começaram a surgir problemas. De um lado, as enormes vantagens de uma comunicação rápida e eficiente, do outro lado, as mensagens desprotegidas.

Mas a grande explosão da criptografia se deu no século XX, pois ele foi marcado por guerras, grandes evoluções na tecnologia e na ciência, movimentação de pessoas, um verdadeiro bombardeio de informações, até que a era digital tomou uma força ainda maior.

A criptologia foi um fator primordial e decisivo no desfecho da 1ª e 2ª Guerras Mundiais, pois também foram travadas verdadeiras batalhas nos “bastidores” dos “aliados” e também dos paises do “eixo”, pois tentavam decifrar de qualquer maneira as mensagens dos inimigos para tentar antecipá-los em emboscadas. Durante a guerra, os ingleses ficaram conhecidos por seus esforços processo de decifrar códigos [WWW1].

Mas na verdade a criptologia só se tornou o que é hoje devido à criação dos computadores. Eles são o maior indicativo da era digital. Depois da Segunda Guerra Mundial, com a invenção do computador, a área realmente floresceu incorporando complexos algoritmos matemáticos.

Vendo que a segurança das informações era de suma importância para o florescimento da informática, os EUA resolveram criar a NSA (National Security Agency), a

mais secreta das agências de inteligência, ela foi criada em 1952 com uma dupla missão: criar sistemas criptográficos para uso do governo dos EUA e quebrar aqueles utilizados pôr outras nações, para poder espionar suas comunicações [WWW1].

A partir dos anos 60, a criptoanálise vem evoluindo extraordinariamente, um projeto conduzido pela IBM, mudou totalmente o conceito sobre criptografia. Um projeto de um algoritmo chamado Lúcifer foi desenvolvido e apresentado a NSA (National Security Agency), no qual após a sua avaliação, introduz modificações (S-Boxes e chaves menores). Esse algoritmo depois de alguns anos serviu de base para a criação do DES (Data Encryption Standard) e muitos outros produtos cifrantes. A partir desse momento, vendo a importância do estudo da criptoanálise, começou uma verdadeira corrida nos EUA para construção de algoritmos cada vez mais eficientes e inquebráveis e ao mesmo tempo uma corrida de pesquisadores para conseguir quebrar os códigos, isso gerou estudo em cima de novos algoritmos, os quais são cada vez mais inquebráveis [WWW1].

1.2 - Justificativa

Este trabalho, de forma geral, engloba o estudo dos mais importantes algoritmos criptográficos, tais como: Serpent, Mars, DES, RC6, Rijndael, Twofish, AES. Mais especificamente destacamos o algoritmo “SERPENT”. Este trabalho mostra como foi importante a criptografia na história da humanidade, onde foram vencidas guerras com a ajuda da criptoanálise. E hoje em dia, na era digital ela se encontra mais importante ainda, pois há um grande fluxo de informações trafegando na rede e essas informações precisam de segurança, e essa segurança só é possível através da criptoanálise. Esse trabalho, vai poder mostrar como é importante a criptografia hoje em dia, pois a cada dia que passa aumentam ainda mais os ataques e invasões em redes de computadores.

1.3 - Objetivos

O presente trabalho tem como objetivo o estudo sobre a criptografia, seus algoritmos mais importantes e com um aprofundamento, sobre o algoritmo “SERPENT”. Este algoritmo será implementado nas linguagens C e Java, sendo feita análise de desempenho nessas duas linguagens de programação.

1.4 - Organização do trabalho

Esse trabalho esta organizado em 5 capítulos, a saber:

No primeiro capítulo é feita uma introdução sobre o assunto geral. Foi contada uma breve história sobre a criptologia.

Uma explicação mais profunda sobre a criptografia é desenvolvida no segundo capítulo. Nessa parte do trabalho também se apresenta alguns dos diversos algoritmos de criptografia existentes e uma breve explicação sobre cada um e os termos mais usados .

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

O quarto capítulo, são apresentados alguns conceitos sobre a linguagem “JAVA”, e se implementa o mesmo algoritmo “Serpent”, também foram feitos testes para verificar seu desempenho nas linguagens “C” e “JAVA”

Pôr fim, no quinto capítulo, é feita uma conclusão sobre tudo que foi estudado ao longo desse trabalho, destacando os principais resultados.

CAPITULO 2

CONCEITOS BÁSICOS DE CRIPTOGRAFIA

2.1 - Introdução

No 2º capitulo será apresentada uma explicação mais profunda sobre como a criptografia é desenvolvida. Nessa parte do trabalho também se apresenta alguns dos diversos algoritmos de criptografia existentes e uma breve explicação sobre cada um, alem dos termos mais usados .

2.2 - Conceitos

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 às informações ou dados, armazenados ou em trânsito, os serviços de segurança, da confidencialidade, integridade e autenticidade [BURNETT,2002].

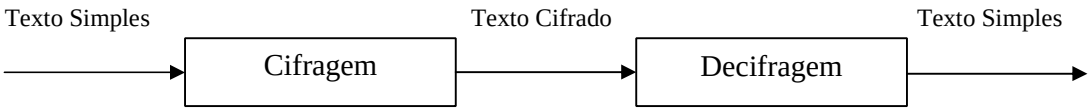


Figura 2.1 Os processos de cifragem e decifragem

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 cifragem. O processo contrário, ou seja, retornar uma mensagem em texto comum novamente é chamado de 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, 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.

2.3 - 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 do ciframento 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 segundo [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 cifragem e para decifragem do texto:

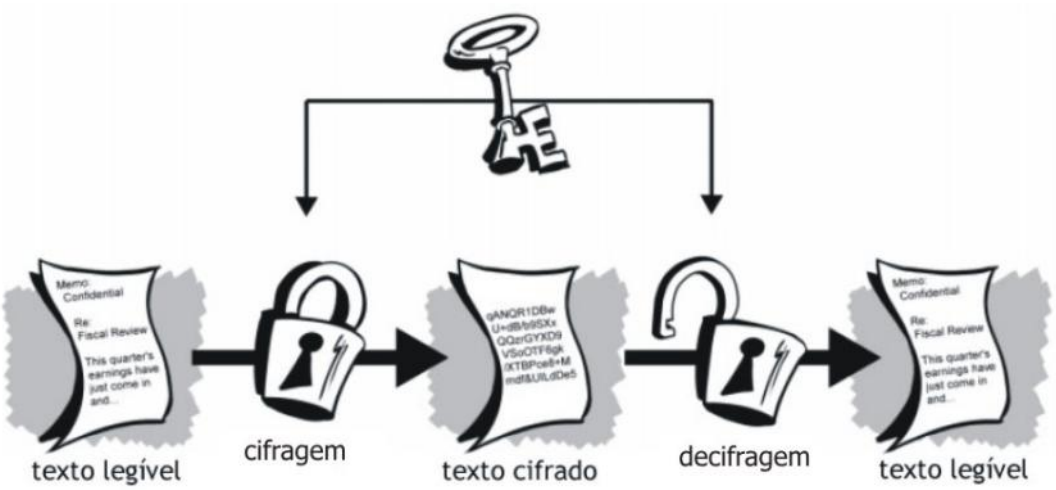


Figura 2.2 Processo de cifragem e decifragem usando chave simétrica [**Maia, L.P, 2005]**

2.4 - 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 um 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 0 e 100 bilhões, mas isso talvez não seja necessário. Uma vez que a chave correta for encontrada, não há necessidade continuar a pesquisa. Na média, o invasor tentará metade de todas as possíveis chaves — no exemplo anterior, 50 bilhões de chaves — antes de encontrar a correta. Às vezes leva mais tempo, às vezes menos, mas em média, aproximadamente metade de todas as possíveis chaves devem ser tentadas [Oliveira, E.E.L. ,2005].

Um invasor para tentar 50 bilhões de chaves utilizaria 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 0 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.2005].

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 utilizado. 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”.

2.4.1 - Exemplos de Algoritmos Simétricos

Nesta serão se discutem alguns dos vários algoritmos de criptografia, mas as chaves não são intercambiáveis entre algoritmos. Por exemplo, suponha que precisa-se criptografar os dados utilizando o algoritmo *Triple Digital Encryption Standard*, melhor conhecido como 3 DES. Se tentar decriptar 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 apresenta alguns algoritmos simétricos e suas características:

- *Digital Encryption Standard(DES*) é o algorítmo 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 propos 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*”.

- *Tripe-Des(3 DES),* é uma simples variação do DES, utilizando-o em três ciframento suscessivos, 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, algorítmo 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 de 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., 2005]. 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 IP, 32 voltas onde se aplicam as funções criptográficas e uma permutação FP.

2.5 - Criptografia Assimétrica

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

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 pode ser decifrada com a outra chave correspondente. A chave privada deve ser mantida secreta, enquanto a chave pública disponível livremente para qualquer interessado.

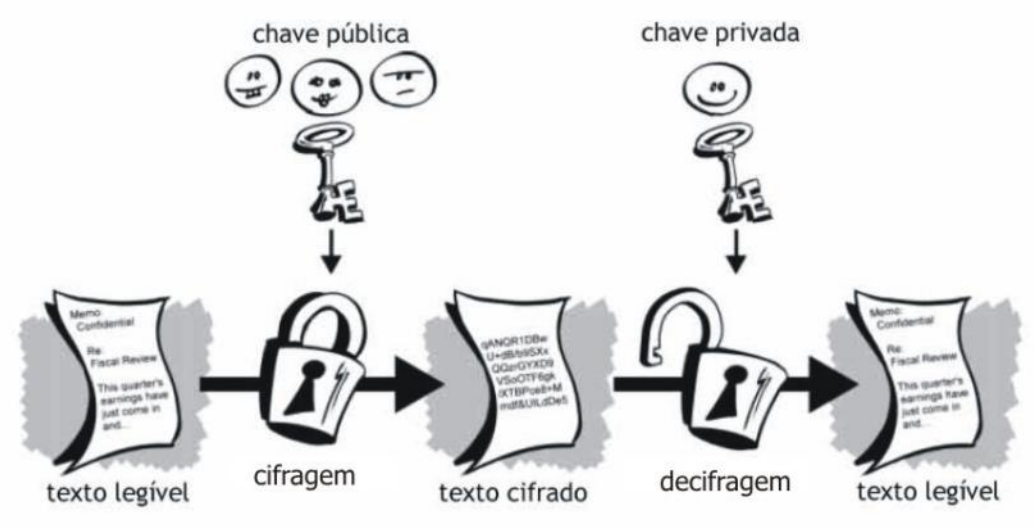


Figura 2.3 Processo de cifragem e decifragem usando chaves assimétricas [**Maia, L.P, 2005].**

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.

Esse sistema 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 propriedade 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*, olgoritmo que envolve a manipulação matemática de grandes quantidades numéricas. Sua segurança advém de algo denominado problema de logaritmo discreto. Sendo assim, esse olgoritmo 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 ciframento 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 publico.

- *Curvas Elípticas*, os sistemas criptográficos de curvas elípticas consistem em modificações de outros sistemas (o ElGamal, por exemplo), quem passam a trabalhar no domínio dos corpos finitos. Eles possuem o potencial de proverem sistemas criptográficos de chave pública mais seguros, 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 pública: 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.

2.6 - 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 pudesse ter que 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 criptografado 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 : 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)

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 pleo *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

2.6.1 - 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* e 8 em *backward mode*. A terceira e última fase é, essencialmente, o inverso da primeira fase.

O algoritmo RC6 [WWW7] 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*.

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 [WWW7] é 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 *IP*; 32 voltas onde se aplicam as funções criptográficas e uma permutação *FP*.

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*.

2.6.2 - 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 [WWW7].

2.6.2.1 - Complexidade Computacional

O NIST como já vimos anteriormente fazia algumas exigências para que um algoritmo pudesse ser submetido ao AES e devido ao fato de umas dessas exigências ser de os algoritmos serem aplicados tanto em software com em hardware, os algoritmos criptográficos não são muito complexos computacionalmente.

A complexidade computacional pode ser entendida como o conjunto de alguns fatores tais como: memória necessária, capacidade de ser aplicado em diversas e plataformas, maior portabilidade, etc [WWW7].

2.6.2.2 - Requisitos de Memória

Uma das características de grande importância para um algoritmo criptográfico é a capacidade dele criptografar ou decriptografar 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*[WWW7]

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

2.6.2.3 - Velocidade de Processamento

A velocidade de processamento é um fator importante, pois de nada adianta um algoritmo ser extremamente seguro, mas processar em uma velocidade muita baixa.

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 criptografando em Linguagem C, utilizando os tamanhos de chave exigidos pelo NIST – 128, 192 e 256 bits.

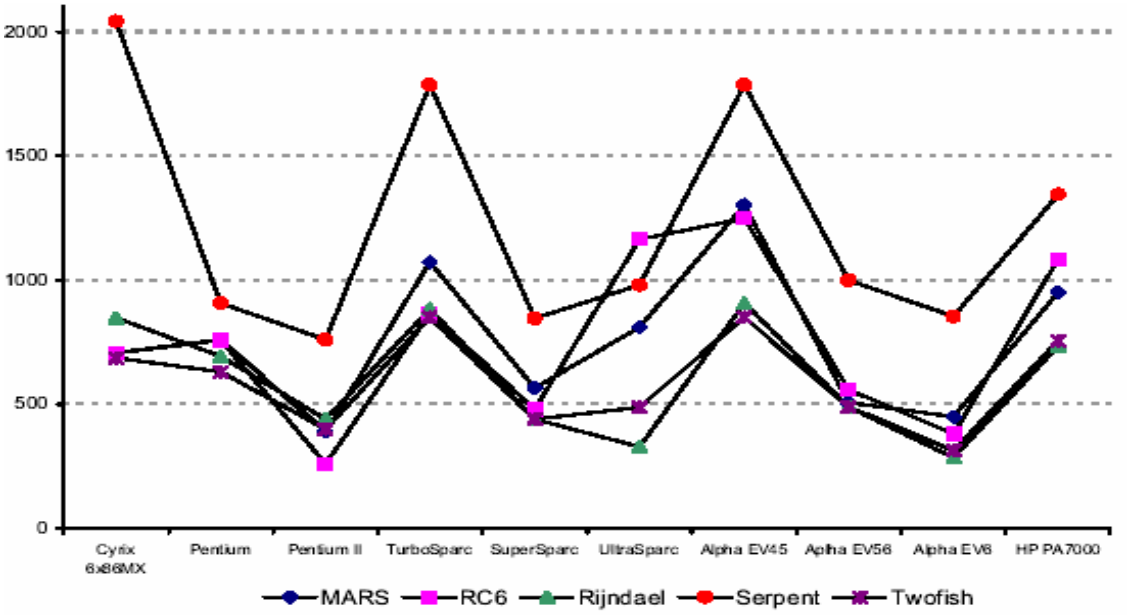


Figura 2.4 - Velocidade de Encriptação para uma chave de 128 bits em C[WWW7]

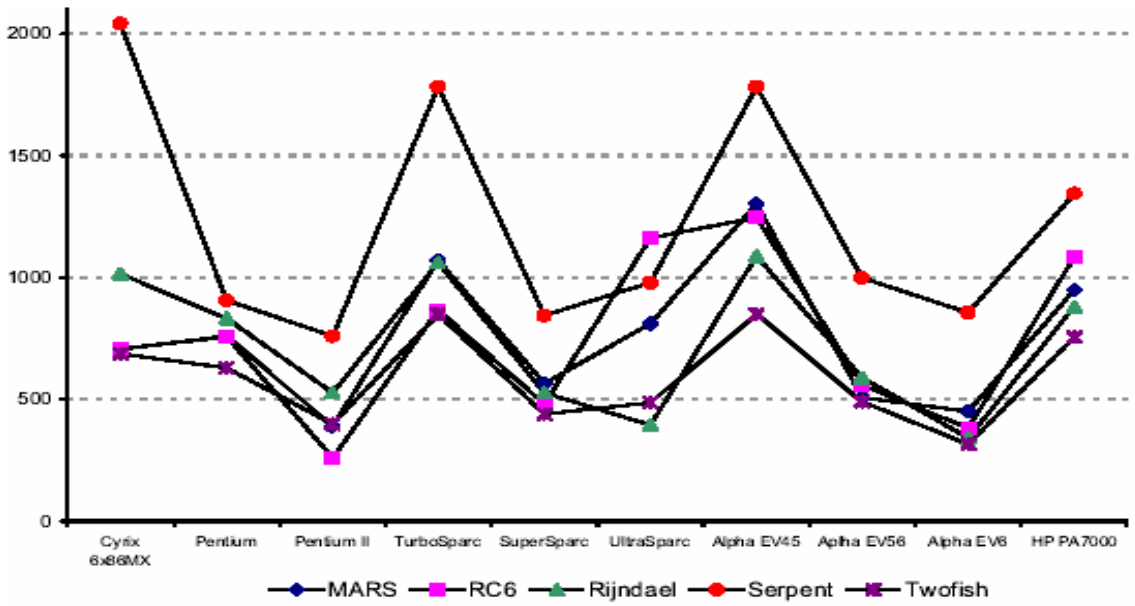


Figura 2.5 - Velocidade de Encriptação para uma chave de 192 bits em C[WWW7]

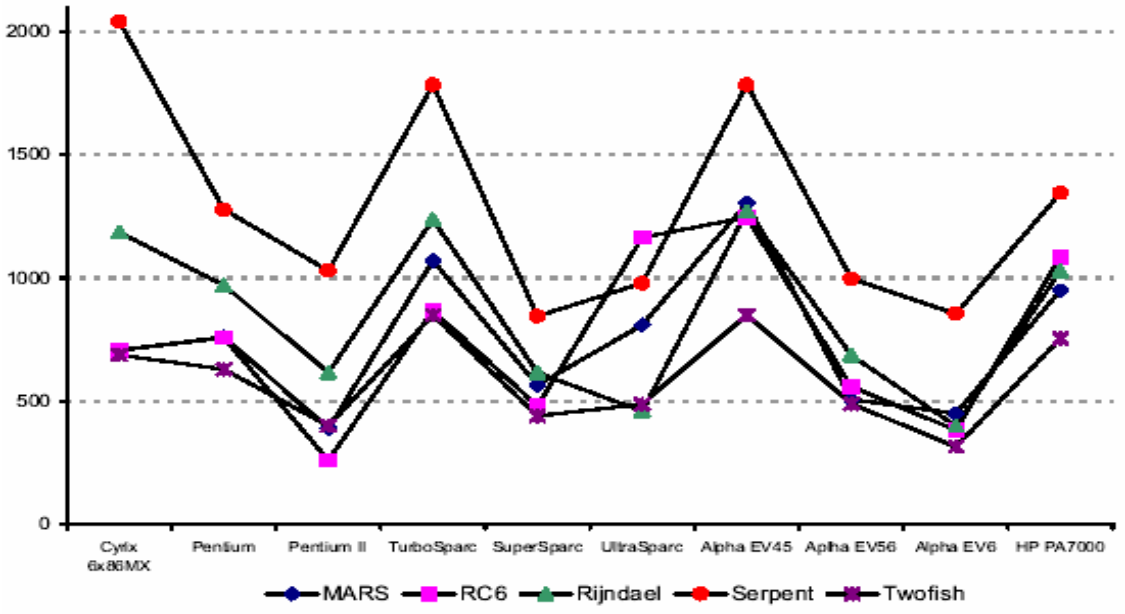


Figura 2.6 - Velocidade de Encriptação para uma chave de 256 bits em C[WWW7]

2.6.2.4 - Funções Criptográficas

A maioria dos algoritmos criptográficos geralmente se utilizam de 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 [WWW7]

	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 decifrark, maior 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.

CAPITULO 3

O ALGORITMO SERPENT NA LINGUAGEM “C”

3.1 - Introdução

Neste capitulo, o trabalho dá mais ênfase no algoritmo “Serpent”, portanto, o terceiro capítulo, é reservado apenas para a explicação desse algoritmo, sua implementação na linguagem C e alguns testes em diversas máquinas diferentes para testar seu desempenho.

3.2 - Um Breve Histórico da Linguagem “C”

C é uma linguagem de programação que foi desenvolvida por Dennis Ritchie durante o começo dos anos 70 para ser usada na implementação de sistemas operacionais, como o UNIX e outras tarefas de programação de baixo nível [WWW4]. Derivada da linguagem sem tipos BCPL(A linguagem BCPL foi projetada por Martin Richards no meio dos anos 60

enquanto ele estava visitando o MIT, e foi usada no começo dos anos 70), ela evoluiu para um modelo estruturado; criada numa minúscula máquina como uma ferramenta para melhorar um ambiente de programação escasso, ela tornou-se uma das linguagens dominantes de hoje [KERNIGHAN, B e RITCHIE, D, 2005]

O desenvolvimento inicial da linguagem C ocorreu entre 1969 e 1973. Em 1973 ela tornou-se poderosa o suficiente para reimplementar o kernel do sistema operacional Unix. Em 1978, Brian Kernighan e Dennis Ritchie publicaram o agora bastante conhecido "C Programming Language" [KERNIGHAN, B e RITCHIE, D, 2005]. C tornou-se popular fora do Bell Labs depois de 1980 e foi por um tempo a linguagem dominante em programação de sistemas e de aplicações de micro-computadores [WWW4].

Ela se consagrou como a linguagem de programação de sistemas, e é a mais importante da comunidade Open Source. Bjarne Stroustrup e outros no Bell Labs trabalharam no final dos anos 80 para adicionar a orientação a objetos ao C, criando uma linguagem chamada C++ [KERNIGHAN, B e RITCHIE, D, 2005]. Implementações de C não checam erros tais como buffer overflow ou acesso a memória não alocada em tempo de execução. Ferramentas tais como o lint têm sido criadas para ajudar programadores a evitar esses erros. Da mesma forma a linguagem de programação Cyclone é uma versão modificada da linguagem de programação C destinada a reduzir esses problemas[WWW5].

3.3 - Princípios da Criptografia Moderna

Uma das grandes preocupações do mundo da ciência da computação nos dias de hoje é a maneira como os dados são protegidos, assim garantindo o sigilo total dos dados transmitidos eletronicamente através da rede. A partir dessa preocupação, os estudos em cima da criptografia partiram a ser de suma importância na construção de sistemas[WWW1].

Com o fim da segunda guerra mundial a criptografia passou a ser bem mais empregada. Uns dos pioneiros nesse emprego foi a IBM, com a criação do algoritmo “Lúcifer”, que passou a ser o primeiro algoritmo criptográfico a ser utilizado comercialmente em 1974. Esse algoritmo sofreu diversas modificações pela NSA, onde passou a ser conhecido como DES transformando-se no padrão da criptografia americana. Após um período de cerca de 20 anos do uso do DES sem maiores problemas, onde especialistas concluíram que o algoritmo era muito bem projetado e que para “quebrá-lo” seria necessário a construção de uma maquina especifica e custosa, estabeleceu-se que esse algoritmo simétrico continuaria como padrão até 1998, quando seria escolhido seu substituto[WWW1].

A substituição começou a ocorrer logo no começo de 1997, quando *o* NIST anunciou o inicio dos trabalhos de desenvolvimento do AES [WWW1], já em setembro daquele mesmo ano, o NIST, tinha definido que os concorrentes a substituição dos DES, deveriam passar por varias etapas, nas quais tinham que ser avaliados publicamente, seus direitos autorais seriam livres, o algoritmo deveria ser simétrico, suportar blocos de 128 bits e chaves de 128, 192 e 256 bits. Após um ano de análises, o NIST chegou aos 15 primeiros finalistas, que foram apresentados na primeira conferencia dos candidatos AES: CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOK197, MAGENTA, MARS, RC6, RIJNDAEL, SAFER+, SERPENT, TWOFISH.

Estes algoritmos foram submetidos a uma avaliação feita por membros da comunidade internacional de criptografia,. Em março de 1999, uma nova conferência foi realizada onde foram discutidos e analisados todos os algoritmos e através dessa decisão foram selecionados 5 novos algoritmos candidatos, que foram: MARS, RC6, RIJNDAEL, SERPENT E TWOFISH. Eles foram analisados novamente e o NIST patrocinou o 3º encontro entre os finalistas, dessa vez convidando seus criadores para falarem, discutirem e responderem a criticas sobre seus candidatos. Em maio de 2000, o NIST iniciou a análise de

todos os argumentos e em 2 de Outubro, o algoritmo Rijindael foi escolhido pela NIST transformando-se no mais novo AES/128 [WWW3].

Todos os algoritmos escolhidos eram muitos seguros, foram levadas em contas as seguintes características para a classificação final: segurança (existência de um ataque que não fosse utilizando a "força bruta" que conseguisse "quebrar" o algoritmo), desempenho (todos foram melhor que o DES, mas tiveram grandes diferenças de performance) e características intrínsecas inclusive propriedade intelectual.

O resultado obtido foi:

- 1-Rijndael (86 votos);
- 2-Serpent (59 votos);
- 3-Twofish (31 votos);
- 4-RC6 (23 votos);
- 5-MARS (13 votos)

Mas, o motivo pelo qual o Rijndael foi escolhido era pôr ser o mais apropriado pôr motivos de eficiência e facilidade de implementação em equipamentos de recursos limitados.

O novo algoritmo AES suporta tamanho de chave e de bloco de 128 a 256 bits e como já citado, assim como DES utiliza substituição, permutação e várias rotações circulares (uma forma de substituição) o número de rotações depende do tamanho da chave e do tamanho do bloco.

Uma curiosidade é que caso a Lei de Moore (a cada 18 meses a tecnologia dobra o poder computacional dos semicondutores) não seja contrariada e não haja nenhuma descoberta científica relevante (matemática, física, etc.) então o AES/128 bits durará até 2066 e o AES/256 alguns séculos[WWW8].

3.4 - O algoritmo Serpent

O Serpent é um algoritmo criptográfico desenvolvido pôr 3 pesquisadores da área de informática, são eles: Ross Anderson da Inglaterra, Eli Bihan de Israel e Lars Knudsen da Noruega. Eles desenvolveram esse algoritmo para ser candidato junto com outros algoritmos, e no final veria qual substituiria o DES e se tornaria o mais novo algoritmo de criptografia AES/128. Com a votação concluída, o algoritmo escolhido foi o Rijndael, mas o Serpent é muito semelhante a ele, onde a diferença principal é a velocidade do Rijndael, mais rápido (poucos círculos), mais o Serpent é relativamente mais seguro [WWW5].

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 e 256 [Antonio, A. M. H., 2000,].

Geralmente o algoritmo Serpent é formado pôr uma Permutação IP (Permutação Inicial), 32 voltas onde se aplicam as funções criptográficas e uma Permutação FP (Permutação Final).

3.4.1 – Descrição do Algoritmo

Segundo [Antonio, A. M. H., 2000] são executadas algumas funções criptográficas a cada uma das 32 iterações, essas funções são:

- uma inserção de chaves;
- uma passagem pelas S-Boxes;
- uma transformação linear que é descartada na última iteração.

3.4.1.1 Permutação Inicial (IP)

A permutação IP é de certa forma simples, onde os 128 bits do texto são trocados de lugar conforme se indica na tabela 3.1: o bit de número 1 do bloco de texto (o segundo bit) vai para a posição do bit número 4 (o quinto bit) e assim pôr diante [Antonio, A. M. H., 2000]

Tabela 3.1 A permutação IP [Antonio, A. M. H., 2000, pág 41]

0	32	64	96	1	33	65	97	2	34	66	98	3	35	67	99
4	36	68	100	5	37	69	101	6	38	70	102	7	39	71	103
8	40	72	104	9	41	73	105	10	42	74	106	11	43	75	107
12	44	76	108	13	45	77	109	14	46	78	110	15	47	79	11
16	48	80	112	17	49	81	113	18	50	82	114	19	51	83	115
20	52	84	116	21	53	85	117	22	54	86	118	23	55	87	119
24	56	88	120	25	57	89	121	26	58	90	122	27	59	91	123
28	60	92	124	29	61	93	125	30	62	94	126	31	63	95	127

O esquema de cifragem e decifragem do algoritmo “Serpent” indicados na Figura 3.2

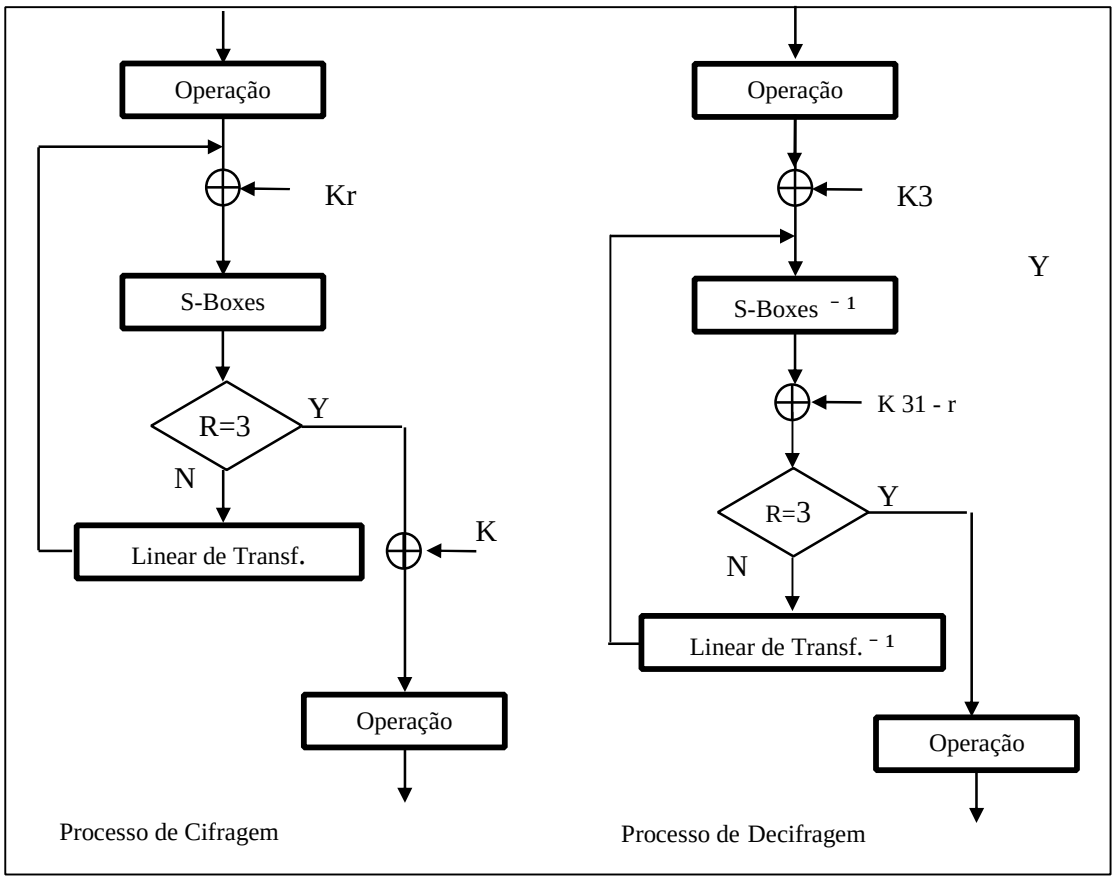


Figura 3.1 Esquemas de Cifragem e Decifragem [WWW6]

3.4.1.2 Inserção da Chave

A cada volta, é inserida no texto uma chave que faz parte de um conjunto de 33 sub-chaves de 128, 192 ou 256 bits produzidas a partir de uma chave de tamanho variável,

fornecida pelo usuário. Chaves menores do que o tamanho determinado são formatadas adicionando bit 1 ao final da chave mais tantos bits 0 quantos forem necessário [WWW7].

Uma sub-chave *Ki* é inserida no bloco de texto *Bi* através de uma operação XOR, como se observa na Figura 3.2

Esta versão do algoritmo "Serpent" visto na figura 3.2 foi cifrada com uma chave de 128 bits na chave. O Serpent necessita de 32 círculos e a necessidade de cada circulo gera uma sub-chave de 128 bits através de um EXOR no bloco de texto. As 32 sub-chaves são geradas pelo algoritmo chave da geração descrita na figura 3.2 [WWW6].

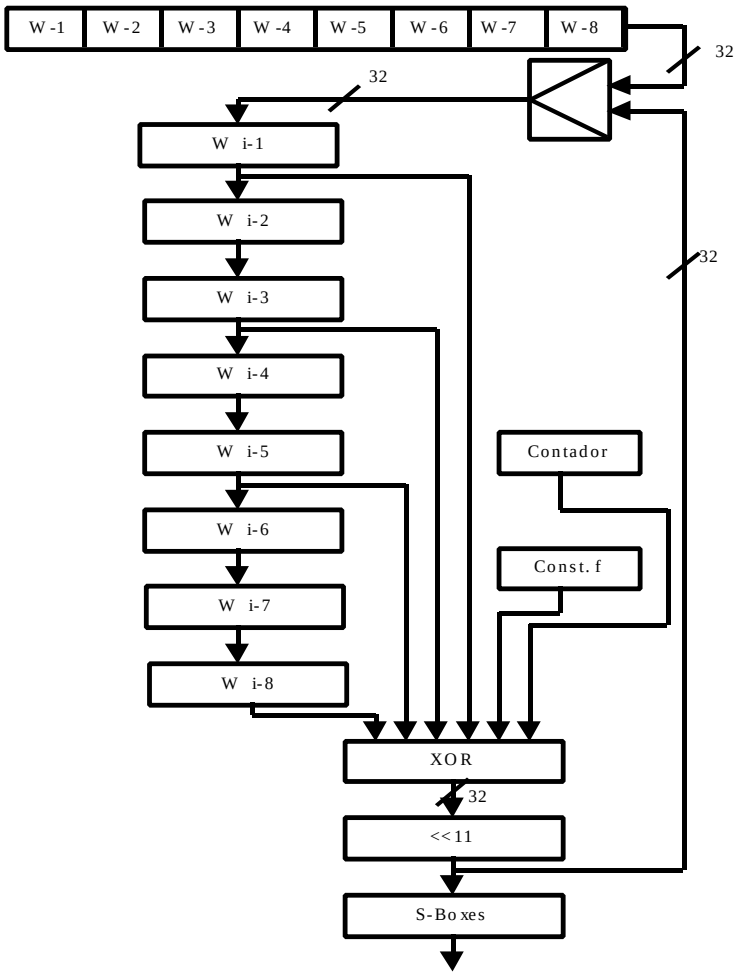


Figura 3.2 – Geração de chave no Serpent [WWW6]

Os processos do algoritmo que geram uma chave de 32 bits são pouco difíceis de executar, sendo que uma sub-chave é calculada em paralelo com a realização de círculos correspondentes. Os algoritmos são executados enquanto uma unidade separa do cifrador as sub-chaves e os armazena em um bloco de memória, conforme se observa na Figura 3.2 [WWW6].

3.4.1.3 - S-Boxes

O Serpent trabalha com 4 S-boxes cada um dele é utilizada 8 vezes e cada S-Box é utilizada uma de cada vez. Assim a S-Box S0 é utilizada na volta 0, S-Box S1 é utilizada na volta 1 e assim sucessivamente até que na volta 8 a S-Box S1 seja utilizada novamente [WWW7].

As S-Boxes do Serpent são caixas de substituição de 4 bits. Então para alcançar os 32 bits de cada bloco de texto, a cada volta são criadas 8 replicas da S-Box que está sendo usada. Os 32 bits do bloco de texto são distribuídos em ordem pelas 8 replicas da S-Box, ou seja, os primeiros 4 bits do bloco de texto vão para a primeira réplica da S-Box, os próximos 4 bits vão para a Segunda réplica e assim pôr diante [WWW7].

3.3.1.4 - Transformação Linear

Segundo [Antonio, A. M. H., 2000] os 32 bits de cada bloco de texto são transformados linearmente da seguinte maneira:

$$.x0, x1, x2, x3 := S1(Bi \oplus Ki)$$

$$x0 := x0 <<< 13$$

$$x2 := x2 <<< 3$$

$$x1 := x1 \oplus x0 \oplus x2$$

$$x3 := x3 \oplus x2 \oplus (x0 << 3)$$

x1:=x1<<<1

x3:=x3<<<7

x0:= x0⊕x1⊕x1

x2:= x2⊕x3⊕(x1<<7)

x0:=x0<<<5

x2:=x2<<<22

Bi+1:=x0,x1,x2,x3

Obs.: “<<<” é rotação e “<<” é deslocamento

3.4.1.5 - A Permutação Final (FP)

A permutação *FP* segue o mesmo principio da permutação *IP*, entretanto as permutações são realizadas seguindo a tabela 3.2

Tabela 3.2 A Permutação Final [WWW7]

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
65	69	73	77	81	85	89	93	97	101	105	109	113	117	121	125
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
66	70	74	78	82	86	90	94	98	102	106	110	114	118	122	126
3	7	11	15	19	23	37	31	35	39	43	47	51	55	59	127
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127

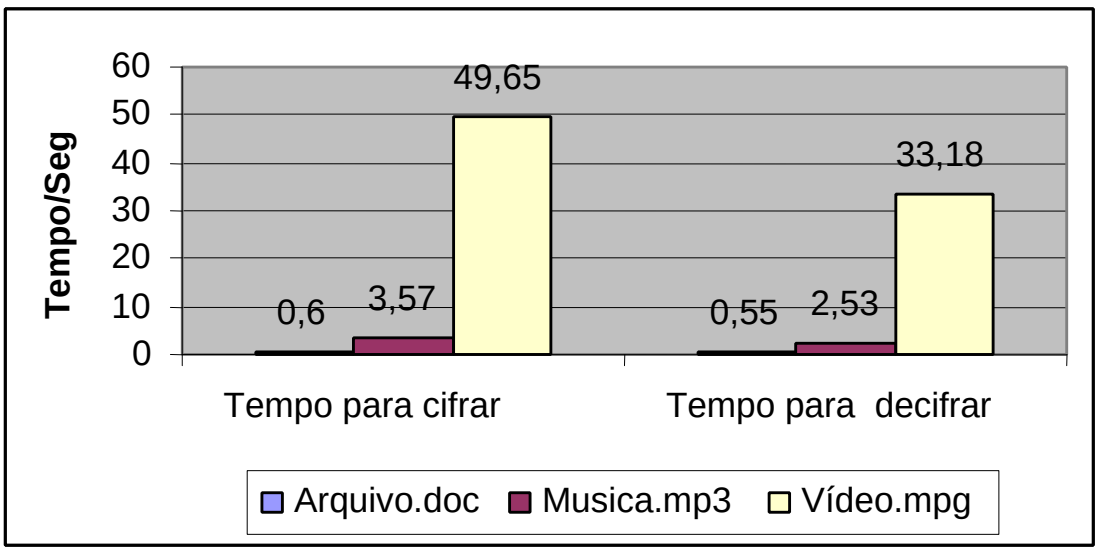
3.4 - Desempenho do Algoritmo “Serpent” na Linguagem “C”

Nesta fase do projeto analisamos como se comporta quanto ao desempenho o algoritmo Serpent, implementado na linguagem C. O código implementado, encontra-se em anexo.

Obs: Os computadores foram inicializados todas as vezes que foram testados os algoritmos e foram feitas 5 tomadas de tempo por arquivo, onde o tempo registrado nas tabelas é a média de todos os tempos.

Tabela 3.3 - Desempenho de um computador Celeron 466Mhz, 64MB Ram, Windows 98..

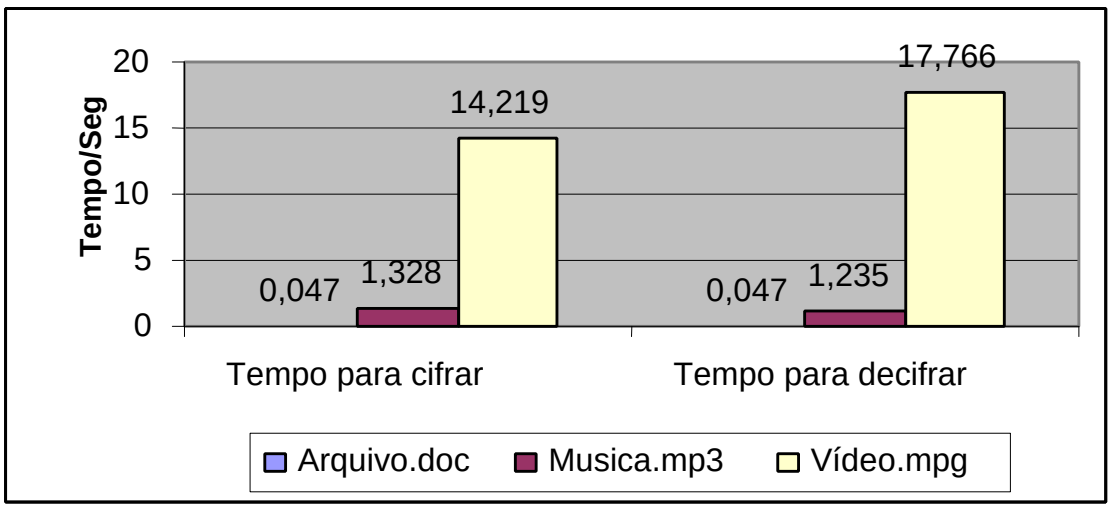
	Tempo para cifrar	*Tempo para decifrar*
Arquivo.doc - 132kb	0.60 seg	0.55 seg
Musica.mp3 - 2,83Mb	3.57 seg	2.53 seg
Vídeo.mpg - 37,9MB	49.65 seg	33.18 seg



Nesta comparação, vemos que o tempo de cifragem é maior que o tempo de decifragem em cada um dos diferentes tipos de arquivos.

Tabela 3.4 - Desempenho de um computador Pentium 4 - 1.60 GHz, Win. XP, 256 RAM

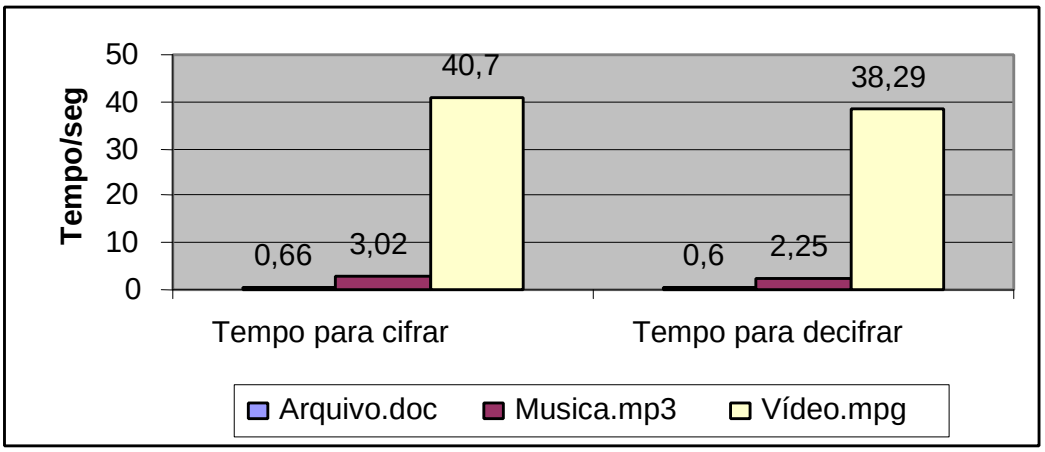
	Tempo para cifrar	*Tempo para decifrar*
Arquivo.doc - 132kb	0.047 seg	0.047 seg
Musica.mp3 - 2,83Mb	1.328 seg	1.235 seg
Vídeo.mpg - 37,9MB	14.219 seg	17.766 seg



Nesta comparação, vemos que o tempo de cifragem variou em relação ao tempo de decifragem em cada um dos diferentes tipos de arquivos.

Tabela 3.5 - Desempenho de um computador Celeron 633MHz , 240 MB RAM, windows 98

	Tempo para cifrar	*Tempo para decifrar*
Arquivo.doc - 132kb	0.66 segundos	0.60 segundos
Musica.mp3 - 2,83Mb	3.02 segundos	2.25 segundos
Vídeo.mpg - 37,9MB	40.70 segundos	38.29 segundos



Nesta comparação, vemos que o tempo de cifragem foi maior em relação ao tempo de decifragem em cada um dos diferentes tipos de arquivos.

Nesta primeira analise, foi verificado que o tempo levado para cifrar é na maioria das vezes maior do que para decifrar (levando em conta nenhum outro processo aberto no computador) em computadores de menor poder de processamento, em uma analise feita em um Pentium 4 os tempos foram praticamente iguais. Os arquivos de texto e musica, tiveram pouca diferença no tempo de cifragem e decifragem em relação ao arquivo de vídeo, por causa da diferença do tamanho dos arquivos.

CAPITULO 4

O ALGORITMO “SERPENT” NA LINGUAGEM JAVA

4.1 - Introdução

No capítulo 4, são dados alguns conceitos sobre a linguagem “JAVA”, juntamente com a implementação do algoritmo “Serpent”, também foram feito testes para verificar seu desempenho nas linguagens “C” e “JAVA”

4.2 - Um Breve Histórico da Linguagem JAVA

Java surgiu com uma pesquisa corporativa interna encomendada pela Sun Microsystens, o primeiro nome dado era Oak em homenagem a um tipo de madeira, mas como já existia essa linguagem, ela passou a se chamar Java. Mas foi com a explosão da Internet, que a Sun anuncio fortemente o Java [Deitel, H. M. e Deitel, P. J, 2002].

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 [WWW9].

A linguagem Java (orientada a objetos) é baseada nas duas linguagens mais usadas no mundo, que é o C e o C++, a Sun retirou alguns recursos que eram muitos confusos e que estavam mais propensos a erros como ponteiros, sobrecarga de operadores, etc. e incluiu recursos realmente mais interessantes para o usuário como interface gráfica, redes clientes/servidor, etc. [Deitel, H. M. e Deitel, P. J, 2002].

Os programas Java consistem em partes chamadas classes. Estas consistem em partes chamadas métodos que realizam tarefas e retornam as informações ao completarem suas tarefas. Um usuário pode programar cada “pedaço” que talvez outro usuário precise para formar um programa em Java. As bibliotecas das classes são também conhecidas como Java APIs. Com isso são duas coisas a aprender no mundo Java, a primeira é aprender a linguagem e a Segunda é usar as APIs. [Deitel, H. M. e Deitel, P. J, 2002].

Os sistemas Java geralmente consistem em várias partes: linguagem, a interface de programas e varias bibliotecas de classes. Segundo [Deitel, H. M. e Deitel, P. J, 2002] os programas Java, normalmente passam pôr 5 fases:

Fase 1: O programa é criado no editor e armazenado no disco

Fase 2: O compilador cria bytecodes e os armazena em disco

Fase 3: O carregador da classe coloca bytecodes na memória

Fase 4: O verificador de bytecodes confirma que todos os bytecodes são válidos e não violam restrições de segurança de Java

Fase 5: O interpretador lê os bytecodes e os traduz para uma linguagem que o computador pode entender, possivelmente armazenando valores dos dados enquanto executa o programa

4.3 - APIs Usadas na Criptografia

A linguagem Java 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 decifrado. Esses códigos são usados para segurança de senhas, não podem ser decifrados, o código hash precisa ser re-gerado e comparado com a seqüência disponível anteriormente. Se ambos se igualarem, o acesso é liberado [Cunha, R. F,2005].

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 primária. 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 pôr um remetente confiável, possuidor da chave privada [Cunha, R. F,2005].

4.4 – Implementação do Algoritmo “Serpent” na Linguagem Java

Nesta seção, estudamos a implementação do algoritmo “Serpent”, mas a implementação deste algoritmo foi totalmente baseada no código já estudado na linguagem C, descrita no capitulo 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 perfeita do algoritmo na linguagem Java.

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

```
public long i,lk,a,b,c,d,e,f,g,h;
public long t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16;
public static long l_key[] = new long [140];
```

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

```
public void k_set(long r,long a,long b,long c,long d)
{
        this.a = l_key[4 * (int)r +  8];
        this.b = l_key[4 * (int)r +  9];
        this.c = l_key[4 * (int)r + 10];
        this.d = l_key[4 * (int)r + 11];
}
public void k_get(long r,long a,long b,long c,long d)
{
        l_key[4 * (int)r +  8] = this.a;
        l_key[4 * (int)r +  9] = this.b;
        l_key[4 * (int)r + 10] = this.c;
        l_key[4 * (int)r + 11] = this.d;
}
```

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

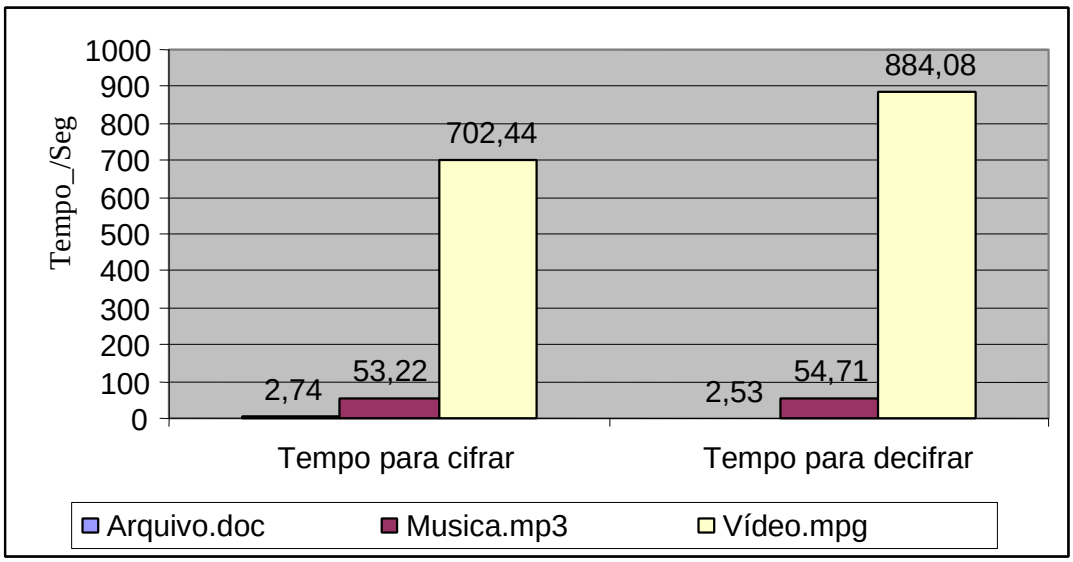
4.5 Teste de Desempenho na linguagem Java e Comparação com Desempenho do algoritmo na linguagem C

Aqui apresentamos os resultados dos testes de cifrar e decifrar dos arquivos de texto, música e vídeo, para obter o desempenho do algoritmo “Serpent” implementado na linguagem Java, assim como foi feito no capíulo anterior utilizando a linguagem C. Também será feita comparações dos gráficos obtidos para verificação de velocidade na duas linguagens.

Tabela - 4.1 Desempenho de um computador Celeron 466Mhz, 64MB Ram, Windows 98.

	Tempo para cifrar	*Tempo para decifrar*
Arquivo.doc - 132kb	2,74	2,53
Musica.mp3 - 2,83Mb	53,22	54,71
Vídeo.mpg - 37,9MB	702,44	884,08

Implementação em Java



Implementação em "C"

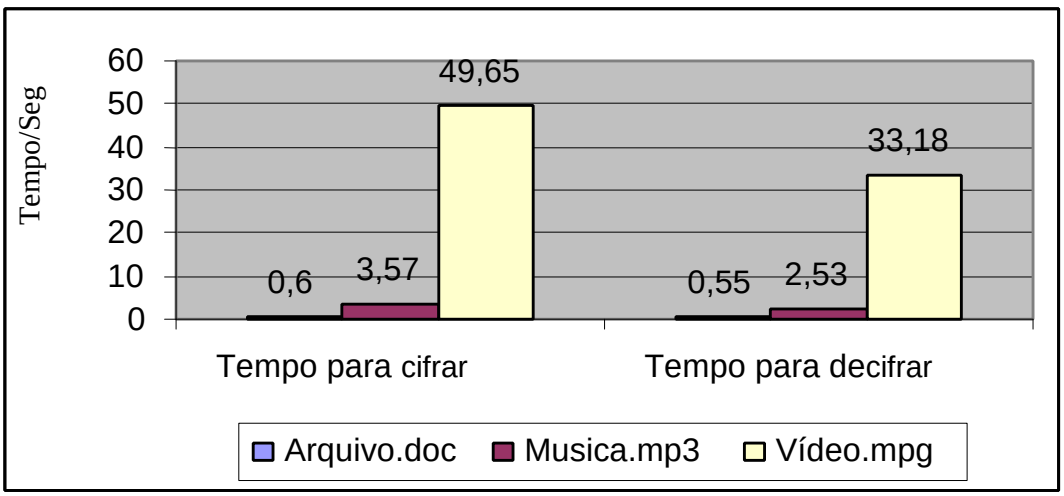
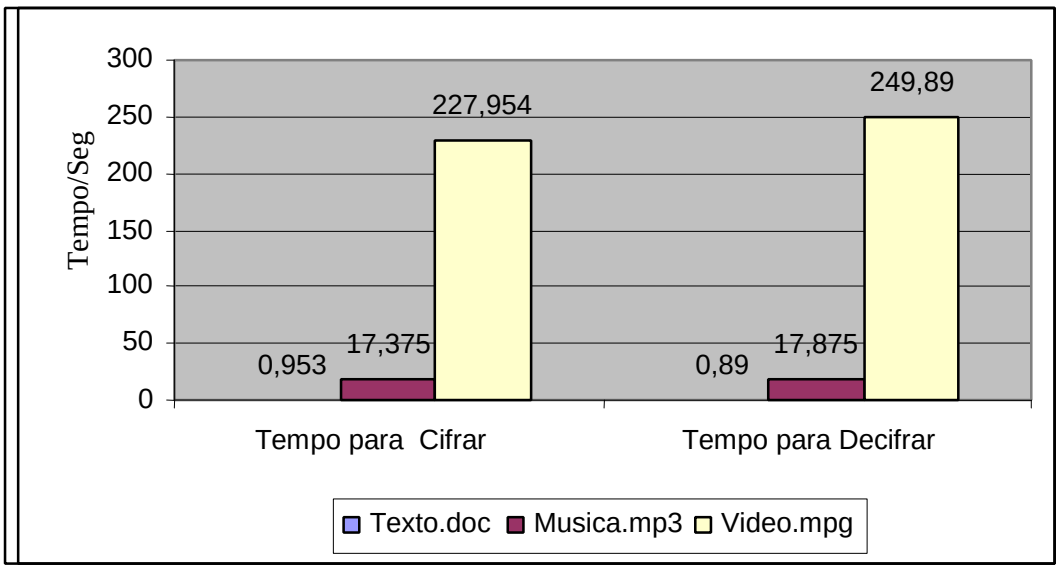


Tabela 4.2 - Desempenho de um computador Pentium 4 1.60 GHz, windows XP, 256 RAM.

	Tempo para cifrar	*Tempo para decifrar*
Arquivo.doc - 132kb	0,953 seg.	0,89 seg.
Musica.mp3 - 2,83Mb	17,375 seg.	17,875 seg.
Vídeo.mpg - 37,9MB	227,954 seg.	249,89 seg.

Implementação em Java



Implementação em C

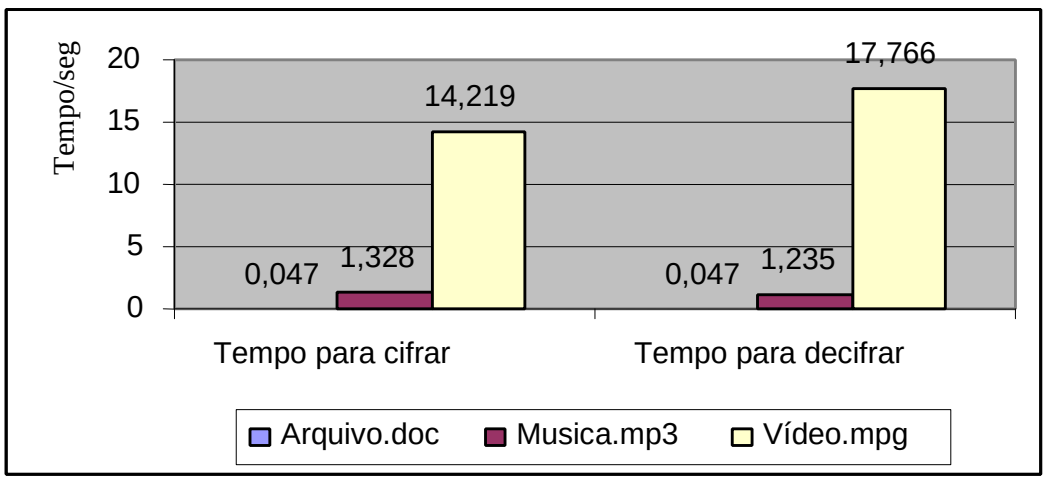
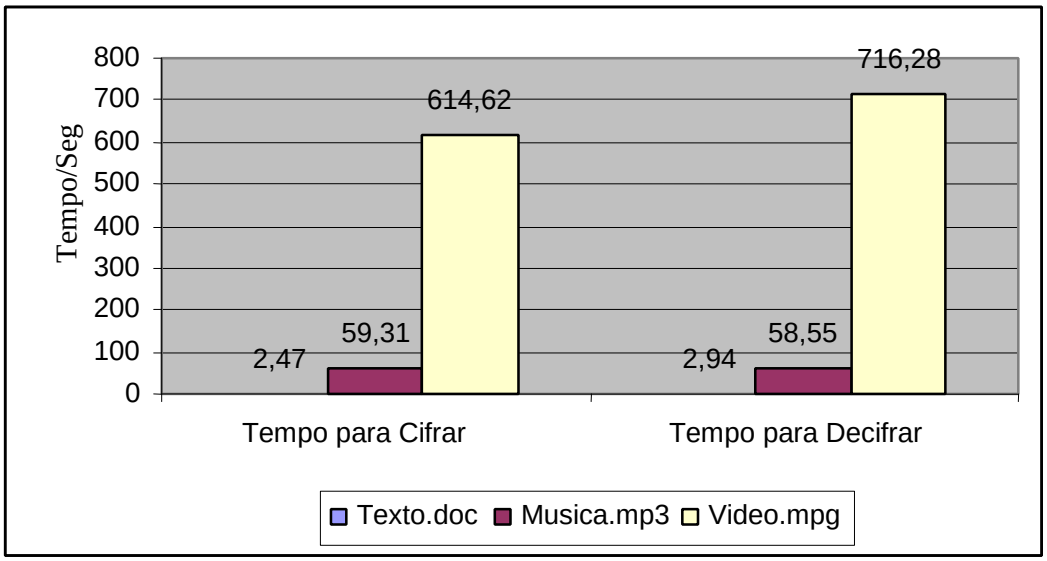


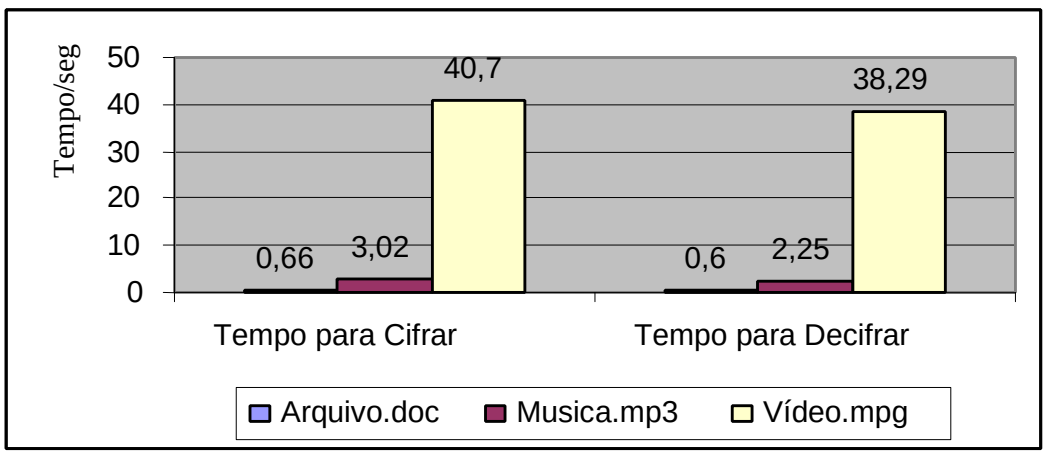
Tabela 4.3 - Desempenho de um computador Celeron 633MHz , 240 MB RAM, windows 98.

	Tempo para cifrar	*Tempo para decifrar*
Arquivo.doc - 132kb	2,47	2,94
Musica.mp3 - 2,83Mb	59,31	58,55
Vídeo.mpg - 37,9MB	614,62	716,28

Implementação em Java



Implementação em C



4.6 - Conclusão dos Testes

De acordo com os testes realizados, usando chave de 128 bits, os arquivos de texto, áudio e vídeo, com arquivos de tamanhos iguais, implementados com o algoritmo Serpent, usando a linguagem Java, possuem um tempo muito maior de cifragem e decifragem, pelo motivo da linguagem Java ser interpretada, chegando a até 10 vezes mais lento do que na linguagem C de acordo com as tabelas 4.1, 4.2 e 4.3 pois C é uma linguagem executável e Java é interpretada.

Conclusão

Podemos verificar o esforço que o NIST promoveu para que houvesse uma substituição do DES. Com isso o NIST trouxe um grande desenvolvimento para o mundo da criptografia, proporcionando muito mais aplicações na área de segurança computacional.

Foram usados três tipos de máquinas para verificação do desempenho do algoritmo SERPENT, cada uma com um tipo de hardware e configuração. Foram utilizados os compiladores C e Java, testando o desempenho de cada algoritmo nas máquinas e compiladores. Levando em conta, sempre a reinicialização das máquinas a cada tomada de tempo para que a memória estivesse sempre limpa.

De acordo com os testes realizados, usando chave de 128 bits, os arquivos de texto, áudio e vídeo, com arquivos de tamanhos iguais, usando a linguagem Java, possuem um tempo muito maior de cifragem e decifragem do os na linguagem C. Com o resultado das comparações, foi verificado que não importa as especificações da máquina, a velocidade de cifragem e decifragem do algoritmo em maquinas com compiladores C é muito menor do que em máquinas com compiladores Java, de fato que, os compiladores C são compilados e os Java são interpretados. Também é descartada a hipótese do tipo do arquivo influenciar no tempo de cifragem e decifragem, pois os cifradores trabalham na ordem dos bits e não em tipos de arquivos.

A proposta inicial deste trabalho previa uma implementação também em hardware deste algoritmo. Contudo, essa implementação não foi realizada devido a falta de tempo, uma vez que a implementação deste algoritmo, em C e em Java, foi utilizado mais tempo que o previsto. Sugere-se que a implementação em hardware fique para outros acadêmicos que possuírem interesse na área de segurança de dados.

Bibliografia

[Antonio, A. M. H., 2000] Antonio, A. M. H., " Um Estudo Descritivo de Novos Algoritmos de Criptografia" TCC apresentado no curso de Informática, na UFP (universidade federal de Pelotas), 2000

[BIHAM. E, 1999] BIHAM, E.. Serpent: A Proposal for the Advanced Encryption Standard. 1999.

[Cunha, R. F, 2005] Cunha, R. F; "Assinatura Digital". Instituto Tecnológico de Aeronáutica. Loiola, M. A ., http://www.comp.pucpcaldas.br/~al550252544/java.htm, acesso em outubro 2005.

[Deitel, H.M., Deitel, P.J, 2003] Deitel, H.M. 2003 Deitel, P.J.; " Java Como Programar ", 4ª edição, Bookman 2003.

[Hinz, M.A.M., 2000] Hinz, M.A.M., "Um esutdo descritivo de novos algoritmos de criptografia", Universidade de Pelotas Instituto de Física e Matemática , Pelotas, 2000.

[IBM, 99] IBM Corporation. MARS – A Candidate Cipher for AES. 1999

[Kernighan, B. W. e Ritchie, D. M., 1988] Kernighan, B. W. e Ritchie, D. M., "C, a Linguagem de Programação" Prentice Hall, Inc., 1988.

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

[Pereira, F. D,2003] Pereira, F. D., "Um Criptoprocessador VLIW para Algoritmos Criptográficos Simétricos" Tese de Mestrado: Centro Universitário "Euripides de Marilia" – UNIVEM, 200

[WWW1] TKOTZ, Viktoria, "Site Aldeia Numaboa", disponível em <http://www.numaboa. com.br /criptologia/historia>, acesso dia 20 de março de 2005

[WWW2] "Site Verdade Absoluta", disponível em <http://www.absoluta.org/cripty/cripty _h.htm>, acesso em março de 2005

[WWW3] REGINA, D. BUENO, R. RIBEIRO,R. disponível em <http://www.security. unicamp.br/docs/conceitos/o4.html>, acesso dia 07 de novembro de 2005

[WWW4] Copyright © 1999,2005 Fundão Inc. disponível em <http://www2.fundao .pro.br/articles.asp?cod=16>, acesso dia 08 setembro de 2005

[WWW5] DENNIS, M. “Bell Labs/Lucent Technologies” disponível em <http://cm.bell-labs.com/cm/cs/who/dmr/chistPT.html>, acesso dia 08 de setembro de 2005

[WWW6] BORA, Piotr “Military University os Technology” disponível em <http://csrc. nist.gov/CryptoToolkit/aes/round2/comments/ 20000513 -pbora.pdf>, acesso em 08 de Outubro de 2005

[WWW7] Antonio, A. M. H. disponível em <http://www.ufpel.tche.br/prg/sisbi/bibct/acervo /info/2000/Mono-MarcoAnto.pdf> acesso em 09 de Outubro de 2005

[WWW8] JERONIMO, V. disponível em <http://www.lockabit.coppe.ufrj.br/rlab/rlab_textos ?id=77>, com acesso em 9 de novembro de 2005

[WWW9] Loiola, M. A disponível em <http://www.comp. pucpcaldas.br/~al550252544/ java.htm>, acesso em 9 de outubro 2005.

Apendice

Implementação do algoritmo Serpent na linguagem Java

```
import javax.swing.*;
import java.io.*;
public class Serpent2{
public long i,lk,a,b,c,d,e,f,g,h;
public long t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16;
public static long l_key[] = new long [140];
public Serpent2()
{ }
public long rotr(long x,long n)
{
        return x>>n;
}
public long rotl(long x,long n)
{
        return x<<n;
}
public void sb0(long a,long b,long c,long d,long e,long f,long g,long h)
{
   t1 = a ^ d;        t2 = a & d;          t3 = c ^ t1;
   t6 = b & t1;       t4 = b ^ t3;     t10 = ~t3;
   h = t2 ^ t4;       t7 = a ^ t6;     t14 = ~t7;
   t8 = c | t7;       t11 = t3 ^ t7;   g = t4 ^ t8;
   t12 = h & t11;     f = t10 ^ t12;   e = t12 ^ t14;
}
public void ib0(long a,long b,long c,long d,long e,long f,long g,long h)
{
   this.t1 = ~a;        this.t2 = a ^ b;       this.t3 = t1 | t2;
   this.t4 = d ^ t3;    this.t7 = d & t2;      this.t5 = c ^ t4;
   this.t8 = t1 ^ t7;   this.g = t2 ^ t5;      this.t11 = a & t4;
   this.t9 = g & t8;    this.t14 = t5 ^ t8;    this.f = t4 ^ t9;
   this.t12 = t5 | f;   this.h = t11 ^ t12;    this.e = h ^ t14;
}
public void sb1(long a,long b,long c,long d,long e,long f,long g,long h)
{
   t1 = ~a;             t2 = b ^ t1;        t3 = a | t2;
   t4 = d | t2;         t5 = c ^ t3;        g = d ^ t5;
   t7 = b ^ t4;         t8 = t2 ^ g;        t9 = t5 & t7;
   h = t8 ^ t9;         t11 = t5 ^ t7;      f = h ^ t11;
   t13 = t8 & t11;      e = t5 ^ t13;
}
public void ib1(long a,long b,long c,long d,long e,long f,long g,long h)
{
```

```
    this.t1 = a ^ d;      this.t2 = a & b;      this.t3 = b ^ c;
    this.t4 = a ^ t3;     this.t5 = b | d;      this.t7 = c | t1;
    this.h = t4 ^ t5;     this.t8 = b ^ t7;     this.t11 = ~t2;
    this.t9 = t4 & t8;    this.f = t1 ^ t9;     this.t13 = t9 ^ t11;
    this.t12 = h & f;     this.g = t12 ^ t13;   this.t15 = a & d;
    this.t16 = c ^ t13;   this.e = t15 ^ t16;
}
public void sb2(long a,long b,long c,long d,long e,long f,long g,long h)
{
    t1 = ~a;          t2 = b ^ d;          t3 = c & t1;
    t13 = d | t1;     e = t2 ^ t3;         t5 = c ^ t1;
    t6 = c ^ e;       t7 = b & t6;         t10 = e | t5;
    h = t5 ^ t7;      t9 = d | t7;         t11 = t9 & t10;
    t14 = t2 ^ h;     g = a ^ t11;         t15 = g ^ t13;
    f = t14 ^ t15;
}
public void ib2(long a,long b,long c,long d,long e,long f,long g,long h)
{
    this.t1 = b ^ d;      this.t2 = ~t1;          this.t3 = a ^ c;
    this.t4 = c ^ t1;     this.t7 = a | t2;       this.t5 = b & t4;
    this.t8 = d ^ t7;     this.t11 = ~t4;         this.e = t3 ^ t5;
    this.t9 = t3 | t8;    this.t14 = d & t11;     this.h = t1 ^ t9;
    this.t12 = e | h;     this.f = t11 ^ t12;     this.t15 = t3 ^ t12;
    this.g = t14 ^ t15;
}
public void sb3(long a,long b,long c,long d,long e,long f,long g,long h)
{
    t1 = a ^ c;       t2 = d ^ t1;         t3 = a & t2;
    t4 = d ^ t3;      t5 = b & t4;         g = t2 ^ t5;
    t7 = a | g;       t8 = b | d;          t11 = a | d;
    t9 = t4 & t7;     f = t8 ^ t9;         t12 = b ^ t11;
    t13 = g ^ t9;     t15 = t3 ^ t8;       h = t12 ^ t13;
    t16 = c & t15;    e = t12 ^ t16;
}
public void ib3(long a,long b,long c,long d,long e,long f,long g,long h)
{
    this.t1 = b ^ c;      this.t2 = b | c;      this.t3 = a ^ c;
    this.t7 = a ^ d;      this.t4 = t2 ^ t3;    this.t5 = d | t4;
    this.t9 = t2 ^ t7;    this.e = t1 ^ t5;     this.t8 = t1 | t5;
    this.t11 = a & t4;    this.g = t8 ^ t9;     this.t12 = e | t9;
    this.f = t11 ^ t12;   this.t14 = a & g;     this.t15 = t2 ^ t14;
    this.t16 = e & t15;   this.h = t4 ^ t16;
}
public void sb4(long a,long b,long c,long d,long e,long f,long g,long h)
   {
```

```
    t1 = a ^ d;        t2 = d & t1;        t3 = c ^ t2;
    t4 = b | t3;       h = t1 ^ t4;        t6 = ~b;
    t7 = t1 | t6;      e = t3 ^ t7;        t9 = a & e;
    t10 = t1 ^ t6;     t11 = t4 & t10;     g = t9 ^ t11;
    t13 = a ^ t3;      t14 = t10 & g;      f = t13 ^ t14;
}
public void ib4(long a,long b,long c,long d,long e,long f,long g,long h)
{
    this.t1 = c ^ d;       this.t2 = c | d;       this.t3 = b ^ t2;
    this.t4 = a & t3;      this.f = t1 ^ t4;      this.t6 = a ^ d;
    this.t7 = b | d;       this.t8 = t6 & t7;     this.h = t3 ^ t8;
    this.t10 = ~a;         this.t11 = c ^ h;      this.t12 = t10 | t11;
    this.e = t3 ^ t12;     this.t14 = c | t4;     this.t15 = t7 ^ t14;
    this.t16 = h | t10;    this.g = t15 ^ t16;
}
public void sb5(long a,long b,long c,long d,long e,long f,long g,long h)
{
    t1 = ~a;           t2 = a ^ b;         t3 = a ^ d;
    t4 = c ^ t1;       t5 = t2 | t3;       e = t4 ^ t5;
    t7 = d & e;        t8 = t2 ^ e;        t10 = t1 | e;
    f = t7 ^ t8;       t11 = t2 | t7;      t12 = t3 ^ t10;
    t14 = b ^ t7;      g = t11 ^ t12;      t15 = f & t12;
    h = t14 ^ t15;
}
public void ib5(long a,long b,long c,long d,long e,long f,long g,long h)
{
    this.t1 = ~c;          this.t2 = b & t1;      this.t3 = d ^ t2;
    this.t4 = a & t3;      this.t5 = b ^ t1;      this.h = t4 ^ t5;
    this.t7 = b | h;       this.t8 = a & t7;      this.f = t3 ^ t8;
    this.t10 = a | d;      this.t11 = t1 ^ t7;    this.e = t10 ^ t11;
    this.t13 = a ^ c;      this.t14 = b & t10;    this.t15 = t4 | t13;
    this.g = t14 ^ t15;
}
public void sb6(long a,long b,long c,long d,long e,long f,long g,long h)
{
    t1 = ~a;           t2 = a ^ d;         t3 = b ^ t2;
    t4 = t1 | t2;      t5 = c ^ t4;        f = b ^ t5;
    t13 = ~t5;         t7 = t2 | f;        t8 = d ^ t7;
    t9 = t5 & t8;      g = t3 ^ t9;        t11 = t5 ^ t8;
    e = g ^ t11;       t14 = t3 & t11;     h = t13 ^ t14;
}
public void ib6(long a,long b,long c,long d,long e,long f,long g,long h)
{
    this.t1 = ~a;          this.t2 = a ^ b;       this.t3 = c ^ t2;
    this.t4 = c | t1;      this.t5 = d ^ t4;      this.t13 = d & t1;
```

```
    this.f = t3 ^ t5;      this.t7 = t3 & t5;      this.t8 = t2 ^ t7;
    this.t9 = b | t8;      this.h = t5 ^ t9;       this.t11 = b | h;
    this.e = t8 ^ t11;     this.t14 = t3 ^ t11;    this.g = t13 ^ t14;
}
public void sb7(long a,long b,long c,long d,long e,long f,long g,long h)
{
    t1 = ~c;               t2 = b ^ c;             t3 = b | t1;
    t4 = d ^ t3;           t5 = a & t4;            t7 = a ^ d;
    h = t2 ^ t5;           t8 = b ^ t5;            t9 = t2 | t8;
    t11 = d & t3;          f = t7 ^ t9;            t12 = t5 ^ f;
    t15 = t1 | t4;         t13 = h & t12;          g = t11 ^ t13;
    t16 = t12 ^ g;         e = t15 ^ t16;
}
public void ib7(long a,long b,long c,long d,long e,long f,long g,long h)
{
    this.t1 = a & b;       this.t2 = a | b;        this.t3 = c | t1;
    this.t4 = d & t2;      this.h = t3 ^ t4;       this.t6 = ~d;
    this.t7 = b ^ t4;      this.t8 = h ^ t6;       this.t11 = c ^ t7;
    this.t9 = t7 | t8;     this.f = a ^ t9;        this.t12 = d | f;
    this.e = t11 ^ t12;    this.t14 = a & h;       this.t15 = t3 ^ f;
    this.t16 = e ^ t14;    this.g = t15 ^ t16;
}
public void k_xor(long r,long a,long b,long c,long d)
{
    this.a ^= l_key[4 * (int)r +  8];
    this.b ^= l_key[4 * (int)r +  9];
    this.c ^= l_key[4 * (int)r + 10];
    this.d ^= l_key[4 * (int)r + 11];
}
public void k_set(long r,long a,long b,long c,long d)
{
    this.a = l_key[4 * (int)r +  8];
    this.b = l_key[4 * (int)r +  9];
    this.c = l_key[4 * (int)r + 10];
    this.d = l_key[4 * (int)r + 11];
}
public void k_get(long r,long a,long b,long c,long d)
{
    l_key[4 * (int)r +  8] = this.a;
    l_key[4 * (int)r +  9] = this.b;
    l_key[4 * (int)r + 10] = this.c;
    l_key[4 * (int)r + 11] = this.d;
}
public void rot(long a,long b,long c,long d)
{
```

```
    a = rotl(a, 13);          c = rotl(c, 3);
    d ^= c ^ (a << 3);        b ^= a ^ c;
    d = rotl(d, 7);           b = rotl(b, 1);
    a ^= b ^ d;               c ^= d ^ (b << 7);
    a = rotl(a, 5);           c = rotl(c, 22);
}
public void irot(long a,long b,long c,long d)
{
    c = rotr(c, 22);          a = rotr(a, 5);
    c ^= d ^ (b << 7);        a ^= b ^ d;
    d = rotr(d, 7);           b = rotr(b, 1);
    d ^= c ^ (a << 3);        b ^= a ^ c;
    c = rotr(c, 3);           a = rotr(a, 13);
}
public long[] set_key(final long in_key[], final long key_len)
{
    if(key_len < 0 || key_len > 256)
            return null;
    i = 0; lk = (key_len + 31) / 32;
    while(i < lk)
    {
                    l_key[(int)i] = in_key[(int)i];
                    i++;
    }
     if(key_len < 256)
    {
                    while(i < 8)
               l_key[(int)i++] = 0;
            i = key_len / 32; lk = 1 << key_len % 32;
    l_key[(int)i] = l_key[(int)i] & (lk - 1) | lk;
            }
    for(i = 0; i < 132; ++i)
    {
            lk = l_key[(int)i] ^ l_key[(int)i + 3] ^ l_key[(int)i + 5]
                                    ^ l_key[(int)i + 7] ^ 0x9e3779b9l ^ i;
            l_key[(int)i + 8] = (lk << 11) | (lk >> 21);
            }
    k_set( 0,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get( 0,e,f,g,h);
    k_set( 1,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get( 1,e,f,g,h);
    k_set( 2,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get( 2,e,f,g,h);
    k_set( 3,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get( 3,e,f,g,h);
    k_set( 4,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get( 4,e,f,g,h);
    k_set( 5,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get( 5,e,f,g,h);
    k_set( 6,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get( 6,e,f,g,h);
    k_set( 7,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get( 7,e,f,g,h);
```

```
    k_set( 8,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get( 8,e,f,g,h);
    k_set( 9,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get( 9,e,f,g,h);
    k_set(10,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get(10,e,f,g,h);
    k_set(11,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get(11,e,f,g,h);
    k_set(12,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get(12,e,f,g,h);
    k_set(13,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get(13,e,f,g,h);
    k_set(14,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get(14,e,f,g,h);
    k_set(15,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get(15,e,f,g,h);
    k_set(16,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get(16,e,f,g,h);
    k_set(17,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get(17,e,f,g,h);
    k_set(18,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get(18,e,f,g,h);
    k_set(19,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get(19,e,f,g,h);
    k_set(20,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get(20,e,f,g,h);
    k_set(21,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get(21,e,f,g,h);
    k_set(22,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get(22,e,f,g,h);
    k_set(23,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get(23,e,f,g,h);
    k_set(24,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get(24,e,f,g,h);
    k_set(25,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get(25,e,f,g,h);
    k_set(26,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get(26,e,f,g,h);
    k_set(27,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get(27,e,f,g,h);
    k_set(28,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get(28,e,f,g,h);
    k_set(29,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get(29,e,f,g,h);
    k_set(30,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get(30,e,f,g,h);
    k_set(31,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get(31,e,f,g,h);
    k_set(32,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get(32,e,f,g,h);
            return l_key;
}
public void encrypt(final byte in_blk[], byte out_blk[])
{
            int i;
            a = in_blk[0]; b = in_blk[1]; c = in_blk[2]; d = in_blk[3];

    k_xor( 0,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 1,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 2,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 3,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 4,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 5,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 6,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 7,e,f,g,h); sb7(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 8,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 9,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(10,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(11,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(12,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
```

```
    k_xor(13,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(14,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(15,e,f,g,h); sb7(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(16,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(17,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(18,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(19,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(20,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(21,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(22,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(23,e,f,g,h); sb7(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(24,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(25,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(26,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(27,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(28,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(29,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(30,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(31,e,f,g,h); sb7(e,f,g,h,a,b,c,d); k_xor(32,a,b,c,d);

    out_blk[0] = (byte)a; out_blk[1] = (byte)b; out_blk[2] = (byte)c; out_blk[3] = (byte)d;
}
public void decrypt(final byte in_blk[], byte out_blk[])
{
    this.a = in_blk[0]; this.b = in_blk[1]; this.c = in_blk[2]; this.d = in_blk[3];

    k_xor(32,a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor(31,e,f,g,h);
    irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor(30,a,b,c,d);
    irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor(29,e,f,g,h);
    irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor(28,a,b,c,d);
    irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor(27,e,f,g,h);
    irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor(26,a,b,c,d);
    irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor(25,e,f,g,h);
    irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor(24,a,b,c,d);
    irot(a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor(23,e,f,g,h);
    irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor(22,a,b,c,d);
    irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor(21,e,f,g,h);
    irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor(20,a,b,c,d);
    irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor(19,e,f,g,h);
    irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor(18,a,b,c,d);
    irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor(17,e,f,g,h);
    irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor(16,a,b,c,d);
    irot(a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor(15,e,f,g,h);
    irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor(14,a,b,c,d);
    irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor(13,e,f,g,h);
```

```
    irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor(12,a,b,c,d);
    irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor(11,e,f,g,h);
    irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor(10,a,b,c,d);
    irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor( 9,e,f,g,h);
    irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor( 8,a,b,c,d);
    irot(a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor( 7,e,f,g,h);
    irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor( 6,a,b,c,d);
    irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor( 5,e,f,g,h);
    irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor( 4,a,b,c,d);
    irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor( 3,e,f,g,h);
    irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor( 2,a,b,c,d);
    irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor( 1,e,f,g,h);
    irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor( 0,a,b,c,d);

    out_blk[0] = (byte)a; out_blk[1] = (byte)b; out_blk[2] = (byte)c; out_blk[3] = (byte)d;
}
public static void main(String args[])
{
    Serpent2 objeto = new Serpent2();

            File inputFile = new File("texto.doc");
    File encripFile = new File("Bejo.doc");
    File decripFile = new File("Levoufora.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));
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));
            }
}
```

Anexo

Implementação do Algoritmo Serpent na Linguagem C

```
#include "time.h"
#define BLOCK_SWAP
#ifdef CORE_TIME
#  undef    BLOCK_SWAP
#endif
#include "std_defs.h"

static char *alg_name[] = { "serpent", "serpent.c", "serpent" };
char **cipher_name()
{
   return alg_name;
}

#define sb0(a,b,c,d,e,f,g,h)    \
   t1 = a ^ d;     \
   t2 = a & d;     \
   t3 = c ^ t1;    \
   t6 = b & t1;    \
   t4 = b ^ t3;    \
   t10 = ~t3;      \
   h = t2 ^ t4;    \
   t7 = a ^ t6;    \
   t14 = ~t7;      \
   t8 = c | t7;    \
   t11 = t3 ^ t7;  \
   g = t4 ^ t8;    \
   t12 = h & t11;  \
   f = t10 ^ t12;  \
   e = t12 ^ t14

#define ib0(a,b,c,d,e,f,g,h)    \
   t1 = ~a;        \
   t2 = a ^ b;     \
   t3 = t1 | t2;   \
   t4 = d ^ t3;    \
   t7 = d & t2;    \
   t5 = c ^ t4;    \
   t8 = t1 ^ t7;   \
   g = t2 ^ t5;    \
   t11 = a & t4;   \
   t9 = g & t8;    \
   t14 = t5 ^ t8;  \
   f = t4 ^ t9;    \
   t12 = t5 | f;   \
   h = t11 ^ t12;  \
   e = h ^ t14


#define sb1(a,b,c,d,e,f,g,h)    \
   t1 = ~a;        \
   t2 = b ^ t1;    \
   t3 = a | t2;    \
   t4 = d | t2;    \
   t5 = c ^ t3;    \
   g = d ^ t5;     \
   t7 = b ^ t4;    \
   t8 = t2 ^ g;    \
   t9 = t5 & t7;   \
   h = t8 ^ t9;    \
   t11 = t5 ^ t7;  \
   f = h ^ t11;    \
   t13 = t8 & t11; \
   e = t5 ^ t13
```

```
#define ib1(a,b,c,d,e,f,g,h)    \
   t1 = a ^ d;     \
   t2 = a & b;     \
   t3 = b ^ c;     \
   t4 = a ^ t3;    \
   t5 = b | d;     \
   t7 = c | t1;    \
   h = t4 ^ t5;    \
   t8 = b ^ t7;    \
   t11 = ~t2;      \
   t9 = t4 & t8;   \
   f = t1 ^ t9;    \
   t13 = t9 ^ t11; \
   t12 = h & f;    \
   g = t12 ^ t13;  \
   t15 = a & d;    \
   t16 = c ^ t13;  \
   e = t15 ^ t16

#define sb2(a,b,c,d,e,f,g,h)    \
   t1 = ~a;        \
   t2 = b ^ d;     \
   t3 = c & t1;    \
   t13 = d | t1;   \
   e = t2 ^ t3;    \
   t5 = c ^ t1;    \
   t6 = c ^ e;     \
   t7 = b & t6;    \
   t10 = e | t5;   \
   h = t5 ^ t7;    \
   t9 = d | t7;    \
   t11 = t9 & t10; \
   t14 = t2 ^ h;   \
   g = a ^ t11;    \
   t15 = g ^ t13;  \
   f = t14 ^ t15

#define ib2(a,b,c,d,e,f,g,h)    \
   t1 = b ^ d;     \
   t2 = ~t1;       \
   t3 = a ^ c;     \
   t4 = c ^ t1;    \
   t7 = a | t2;    \
   t5 = b & t4;    \
   t8 = d ^ t7;    \
   t11 = ~t4;      \
   e = t3 ^ t5;    \
   t9 = t3 | t8;   \
   t14 = d & t11;  \
   h = t1 ^ t9;    \
   t12 = e | h;    \
   f = t11 ^ t12;  \
   t15 = t3 ^ t12; \
   g = t14 ^ t15

#define sb3(a,b,c,d,e,f,g,h)    \
   t1 = a ^ c;     \
   t2 = d ^ t1;    \
   t3 = a & t2;    \
   t4 = d ^ t3;    \
   t5 = b & t4;    \
   g = t2 ^ t5;    \
   t7 = a | g;     \
   t8 = b | d;     \
   t11 = a | d;    \
   t9 = t4 & t7;   \
```

```
    f = t8 ^ t9;    \
    t12 = b ^ t11;  \
    t13 = g ^ t9;   \
    t15 = t3 ^ t8;  \
    h = t12 ^ t13;  \
    t16 = c & t15;  \
    e = t12 ^ t16


#define ib3(a,b,c,d,e,f,g,h)    \
    t1 = b ^ c;     \
    t2 = b | c;     \
    t3 = a ^ c;     \
    t7 = a ^ d;     \
    t4 = t2 ^ t3;   \
    t5 = d | t4;    \
    t9 = t2 ^ t7;   \
    e = t1 ^ t5;    \
    t8 = t1 | t5;   \
    t11 = a & t4;   \
    g = t8 ^ t9;    \
    t12 = e | t9;   \
    f = t11 ^ t12;  \
    t14 = a & g;    \
    t15 = t2 ^ t14; \
    t16 = e & t15;  \
    h = t4 ^ t16

#define sb4(a,b,c,d,e,f,g,h)    \
    t1 = a ^ d;     \
    t2 = d & t1;    \
    t3 = c ^ t2;    \
    t4 = b | t3;    \
    h = t1 ^ t4;    \
    t6 = ~b;        \
    t7 = t1 | t6;   \
    e = t3 ^ t7;    \
    t9 = a & e;     \
    t10 = t1 ^ t6;  \
    t11 = t4 & t10; \
    g = t9 ^ t11;   \
    t13 = a ^ t3;   \
    t14 = t10 & g;  \
    f = t13 ^ t14

#define ib4(a,b,c,d,e,f,g,h)    \
    t1 = c ^ d;     \
    t2 = c | d;     \
    t3 = b ^ t2;    \
    t4 = a & t3;    \
    f = t1 ^ t4;    \
    t6 = a ^ d;     \
    t7 = b | d;     \
    t8 = t6 & t7;   \
    h = t3 ^ t8;    \
    t10 = ~a;       \
    t11 = c ^ h;    \
    t12 = t10 | t11;\
    e = t3 ^ t12;   \
    t14 = c | t4;   \
    t15 = t7 ^ t14; \
    t16 = h | t10;  \
    g = t15 ^ t16

#define sb5(a,b,c,d,e,f,g,h)    \
    t1 = ~a;        \
    t2 = a ^ b;     \
```

```
    t3 = a ^ d;     \
    t4 = c ^ t1;    \
    t5 = t2 | t3;   \
    e = t4 ^ t5;    \
    t7 = d & e;     \
    t8 = t2 ^ e;    \
    t10 = t1 | e;   \
    f = t7 ^ t8;    \
    t11 = t2 | t7;  \
    t12 = t3 ^ t10; \
    t14 = b ^ t7;   \
    g = t11 ^ t12;  \
    t15 = f & t12;  \
    h = t14 ^ t15

#define ib5(a,b,c,d,e,f,g,h)    \
    t1 = ~c;        \
    t2 = b & t1;    \
    t3 = d ^ t2;    \
    t4 = a & t3;    \
    t5 = b ^ t1;    \
    h = t4 ^ t5;    \
    t7 = b | h;     \
    t8 = a & t7;    \
    f = t3 ^ t8;    \
    t10 = a | d;    \
    t11 = t1 ^ t7;  \
    e = t10 ^ t11;  \
    t13 = a ^ c;    \
    t14 = b & t10;  \
    t15 = t4 | t13; \
    g = t14 ^ t15

#define sb6(a,b,c,d,e,f,g,h)    \
    t1 = ~a;        \
    t2 = a ^ d;     \
    t3 = b ^ t2;    \
    t4 = t1 | t2;   \
    t5 = c ^ t4;    \
    f = b ^ t5;     \
    t13 = ~t5;      \
    t7 = t2 | f;    \
    t8 = d ^ t7;    \
    t9 = t5 & t8;   \
    g = t3 ^ t9;    \
    t11 = t5 ^ t8;  \
    e = g ^ t11;    \
    t14 = t3 & t11; \
    h = t13 ^ t14

#define ib6(a,b,c,d,e,f,g,h)    \
    t1 = ~a;        \
    t2 = a ^ b;     \
    t3 = c ^ t2;    \
    t4 = c | t1;    \
    t5 = d ^ t4;    \
    t13 = d & t1;   \
    f = t3 ^ t5;    \
    t7 = t3 & t5;   \
    t8 = t2 ^ t7;   \
    t9 = b | t8;    \
    h = t5 ^ t9;    \
    t11 = b | h;    \
    e = t8 ^ t11;   \
    t14 = t3 ^ t11; \
    g = t13 ^ t14
```

```
#define sb7(a,b,c,d,e,f,g,h)    \
    t1 = ~c;        \
    t2 = b ^ c;     \
    t3 = b | t1;    \
    t4 = d ^ t3;    \
    t5 = a & t4;    \
    t7 = a ^ d;     \
    h = t2 ^ t5;    \
    t8 = b ^ t5;    \
    t9 = t2 | t8;   \
    t11 = d & t3;   \
    f = t7 ^ t9;    \
    t12 = t5 ^ f;   \
    t15 = t1 | t4;  \
    t13 = h & t12;  \
    g = t11 ^ t13;  \
    t16 = t12 ^ g;  \
    e = t15 ^ t16

#define ib7(a,b,c,d,e,f,g,h)    \
    t1 = a & b;     \
    t2 = a | b;     \
    t3 = c | t1;    \
    t4 = d & t2;    \
    h = t3 ^ t4;    \
    t6 = ~d;        \
    t7 = b ^ t4;    \
    t8 = h ^ t6;    \
    t11 = c ^ t7;   \
    t9 = t7 | t8;   \
    f = a ^ t9;     \
    t12 = d | f;    \
    e = t11 ^ t12;  \
    t14 = a & h;    \
    t15 = t3 ^ f;   \
    t16 = e ^ t14;  \
    g = t15 ^ t16

#define k_xor(r,a,b,c,d)    \
    a ^= l_key[4 * r +  8]; \
    b ^= l_key[4 * r +  9]; \
    c ^= l_key[4 * r + 10]; \
    d ^= l_key[4 * r + 11]

#define k_set(r,a,b,c,d)    \
    a = l_key[4 * r +  8];  \
    b = l_key[4 * r +  9];  \
    c = l_key[4 * r + 10];  \
    d = l_key[4 * r + 11]

#define k_get(r,a,b,c,d)    \
    l_key[4 * r +  8] = a;  \
    l_key[4 * r +  9] = b;  \
    l_key[4 * r + 10] = c;  \
    l_key[4 * r + 11] = d

#define rot(a,b,c,d)    \
    a = rotl(a, 13);    \
    c = rotl(c, 3);     \
    d ^= c ^ (a << 3);  \
    b ^= a ^ c;         \
    d = rotl(d, 7);     \
    b = rotl(b, 1);     \
    a ^= b ^ d;         \
    c ^= d ^ (b << 7);  \
    a = rotl(a, 5);     \
    c = rotl(c, 22)
```

```
#define irot(a,b,c,d)   \
    c = rotr(c, 22);    \
    a = rotr(a, 5);     \
    c ^= d ^ (b << 7);  \
    a ^= b ^ d;         \
    d = rotr(d, 7);     \
    b = rotr(b, 1);     \
    d ^= c ^ (a << 3);  \
    b ^= a ^ c;         \
    c = rotr(c, 3);     \
    a = rotr(a, 13)

u4byte  l_key[140];

u4byte *set_key(const u4byte in_key[], const u4byte key_len)
{   u4byte  i,lk,a,b,c,d,e,f,g,h;
    u4byte  t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16;

    if(key_len < 0 || key_len > 256)

        return (u4byte*)0;

    i = 0; lk = (key_len + 31) / 32;

    while(i < lk)
    {
    #ifdef  BLOCK_SWAP
    l_key[i] = io_swap(in_key[lk - i - 1]);
    #else
     l_key[i] = in_key[i];
#endif
        i++;
    }

    if(key_len < 256)
    {
        while(i < 8)

            l_key[i++] = 0;

        i = key_len / 32; lk = 1 << key_len % 32;

        l_key[i] = l_key[i] & (lk - 1) | lk;
    }

    for(i = 0; i < 132; ++i)
    {
        lk = l_key[i] ^ l_key[i + 3] ^ l_key[i + 5]
                        ^ l_key[i + 7] ^ 0x9e3779b9 ^ i;

        l_key[i + 8] = (lk << 11) | (lk >> 21);
    }

    k_set( 0,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get( 0,e,f,g,h);
    k_set( 1,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get( 1,e,f,g,h);
    k_set( 2,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get( 2,e,f,g,h);
    k_set( 3,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get( 3,e,f,g,h);
    k_set( 4,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get( 4,e,f,g,h);
    k_set( 5,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get( 5,e,f,g,h);
    k_set( 6,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get( 6,e,f,g,h);
    k_set( 7,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get( 7,e,f,g,h);
    k_set( 8,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get( 8,e,f,g,h);
    k_set( 9,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get( 9,e,f,g,h);
    k_set(10,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get(10,e,f,g,h);
    k_set(11,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get(11,e,f,g,h);
    k_set(12,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get(12,e,f,g,h);
```

```
    k_set(13,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get(13,e,f,g,h);
    k_set(14,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get(14,e,f,g,h);
    k_set(15,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get(15,e,f,g,h);
    k_set(16,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get(16,e,f,g,h);
    k_set(17,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get(17,e,f,g,h);
    k_set(18,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get(18,e,f,g,h);
    k_set(19,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get(19,e,f,g,h);
    k_set(20,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get(20,e,f,g,h);
    k_set(21,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get(21,e,f,g,h);
    k_set(22,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get(22,e,f,g,h);
    k_set(23,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get(23,e,f,g,h);
    k_set(24,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get(24,e,f,g,h);
    k_set(25,a,b,c,d);sb2(a,b,c,d,e,f,g,h);k_get(25,e,f,g,h);
    k_set(26,a,b,c,d);sb1(a,b,c,d,e,f,g,h);k_get(26,e,f,g,h);
    k_set(27,a,b,c,d);sb0(a,b,c,d,e,f,g,h);k_get(27,e,f,g,h);
    k_set(28,a,b,c,d);sb7(a,b,c,d,e,f,g,h);k_get(28,e,f,g,h);
    k_set(29,a,b,c,d);sb6(a,b,c,d,e,f,g,h);k_get(29,e,f,g,h);
    k_set(30,a,b,c,d);sb5(a,b,c,d,e,f,g,h);k_get(30,e,f,g,h);
    k_set(31,a,b,c,d);sb4(a,b,c,d,e,f,g,h);k_get(31,e,f,g,h);
    k_set(32,a,b,c,d);sb3(a,b,c,d,e,f,g,h);k_get(32,e,f,g,h);

    return l_key;
};

void encrypt(const u4byte in_blk[4], u4byte out_blk[])
{   u4byte  a,b,c,d,e,f,g,h;
    u4byte  t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16;

#ifdef  BLOCK_SWAP
    a = io_swap(in_blk[3]); b = io_swap(in_blk[2]);
    c = io_swap(in_blk[1]); d = io_swap(in_blk[0]);
#else
    a = in_blk[0]; b = in_blk[1]; c = in_blk[2]; d = in_blk[3];
#endif

    k_xor( 0,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 1,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 2,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 3,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 4,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 5,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 6,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 7,e,f,g,h); sb7(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor( 8,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor( 9,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(10,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(11,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(12,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(13,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(14,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(15,e,f,g,h); sb7(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(16,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(17,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(18,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(19,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(20,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(21,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(22,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(23,e,f,g,h); sb7(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(24,a,b,c,d); sb0(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(25,e,f,g,h); sb1(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(26,a,b,c,d); sb2(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(27,e,f,g,h); sb3(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(28,a,b,c,d); sb4(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(29,e,f,g,h); sb5(e,f,g,h,a,b,c,d); rot(a,b,c,d);
    k_xor(30,a,b,c,d); sb6(a,b,c,d,e,f,g,h); rot(e,f,g,h);
    k_xor(31,e,f,g,h); sb7(e,f,g,h,a,b,c,d); k_xor(32,a,b,c,d);
```

```
#ifdef  BLOCK_SWAP
   out_blk[3] = io_swap(a); out_blk[2] = io_swap(b);
   out_blk[1] = io_swap(c); out_blk[0] = io_swap(d);
#else
   out_blk[0] = a; out_blk[1] = b; out_blk[2] = c; out_blk[3] = d;
#endif
};

void decrypt(const u4byte in_blk[4], u4byte out_blk[4])
{   u4byte  a,b,c,d,e,f,g,h;
   u4byte  t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16;

#ifdef  BLOCK_SWAP
   a = io_swap(in_blk[3]); b = io_swap(in_blk[2]);
   c = io_swap(in_blk[1]); d = io_swap(in_blk[0]);
#else
   a = in_blk[0]; b = in_blk[1]; c = in_blk[2]; d = in_blk[3];
#endif

   k_xor(32,a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor(31,e,f,g,h);
   irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor(30,a,b,c,d);
   irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor(29,e,f,g,h);
   irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor(28,a,b,c,d);
   irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor(27,e,f,g,h);
   irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor(26,a,b,c,d);
   irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor(25,e,f,g,h);
   irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor(24,a,b,c,d);
   irot(a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor(23,e,f,g,h);
   irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor(22,a,b,c,d);
   irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor(21,e,f,g,h);
   irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor(20,a,b,c,d);
   irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor(19,e,f,g,h);
   irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor(18,a,b,c,d);
   irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor(17,e,f,g,h);
   irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor(16,a,b,c,d);
   irot(a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor(15,e,f,g,h);
   irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor(14,a,b,c,d);
   irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor(13,e,f,g,h);
   irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor(12,a,b,c,d);
   irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor(11,e,f,g,h);
   irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor(10,a,b,c,d);
   irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor( 9,e,f,g,h);
   irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor( 8,a,b,c,d);
   irot(a,b,c,d); ib7(a,b,c,d,e,f,g,h); k_xor( 7,e,f,g,h);
   irot(e,f,g,h); ib6(e,f,g,h,a,b,c,d); k_xor( 6,a,b,c,d);
   irot(a,b,c,d); ib5(a,b,c,d,e,f,g,h); k_xor( 5,e,f,g,h);
   irot(e,f,g,h); ib4(e,f,g,h,a,b,c,d); k_xor( 4,a,b,c,d);
   irot(a,b,c,d); ib3(a,b,c,d,e,f,g,h); k_xor( 3,e,f,g,h);
   irot(e,f,g,h); ib2(e,f,g,h,a,b,c,d); k_xor( 2,a,b,c,d);
   irot(a,b,c,d); ib1(a,b,c,d,e,f,g,h); k_xor( 1,e,f,g,h);
   irot(e,f,g,h); ib0(e,f,g,h,a,b,c,d); k_xor( 0,a,b,c,d);

#ifdef  BLOCK_SWAP
   out_blk[3] = io_swap(a); out_blk[2] = io_swap(b);
   out_blk[1] = io_swap(c); out_blk[0] = io_swap(d);
#else
   out_blk[0] = a; out_blk[1] = b; out_blk[2] = c; out_blk[3] = d;
#endif
};

#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 criptografar ou decriptografar

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
            };

            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: %.3f 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");
}
```