FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PEDRO HENRIQUE BUGATTI

IMPLEMENTAÇÃO E AVALIAÇÃO DO ALGORITMO BLOWFISH EM C, JAVA E MICROCONTROLADORES PIC

MARÍLIA
2005

PEDRO HENRIQUE BUGATTI

IMPLEMENTAÇÃO E AVALIAÇÃO DO ALGORITMO BLOWFISH EM C, JAVA E MICROCONTROLADORES PIC

Monografia apresentada ao Curso de Ciência da Computação, do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Edward David Moreno

MARÍLIA
2005

PEDRO HENRIQUE BUGATTI

IMPLEMENTAÇÃO E AVALIAÇÃO DO ALGORITMO BLOWFISH C, JAVA E MICROCONTROLADORES PIC

Banca examinadora da monografia apresentada ao Programa de Graduação da UNIVEM,/F.E.E.S.R., para obtenção do Grau de Bacharel em Ciência da Computação. Área de Concentração: Criptografia

Resultado: _____

ORIENTADOR: Pr of. Dr. _____

1º EXAMINADOR: _____

2º EXAMINADOR: _____

Marília, _____de _____de 2005

Dedico a realização desse Trabalho de Conclusão de Curso aos meus queridos pais que nos momentos em que minha alma rondou o caminho do desânimo e da desilusão me guiaram novamente para o trajeto da luz da sabedoria e do amor.

"A Vida é Construída nos Sonhos e Concretizada no Amor"

Francisco Cândido Xavier

AGRADECIMENTOS

Agradeço a meus pais Ildeberto de Genova Bugatti e Ivani Aparecida da Silva Bugatti, pessoas éticas, dedicadas e que me deram a oportunidade de reconhecer que a vida não é feita apenas de perfeições. Sem a ajuda e compreensão deles não teria nem mesmo almejado a realização do presente trabalho. Obrigado por terem me dado a dádiva de ser seu filho.

Agradeço ao meu pai e amigo por todo o esforço, dedicação, respeito e ética com que sempre realizou tudo em sua vida, muitas vezes se esquecendo até de si próprio e por toda a ajuda concedida na realização do presente trabalho.

Agradeço à minha mãe e amiga por todo carinho, amor, compreensão, paciência e por me aturar ao longo desses 21 anos, virão mais pela frente.

Agradeço a meu irmão Fábio Henrique Bugatti pela paciência e companheirismo nas inúmeras vezes em que precisei utilizar o computador.

Agradeço ao meu orientador e amigo Edward David Moreno Ordonez, por ter me concedido a oportunidade de realizar o presente projeto, o qual acrescentou em muito para a minha formação, também por sua orientação e paciência e por ter acreditado em minha capacidade.

Agradeço à FAPESP pelo apoio concedido à pesquisa.

Agradeço a todos os bons amigos que tive oportunidade de conhecer nos laboratóios de pesquisa no decorrer do desenvolvimento deste e de outros trabalhos. Entre eles estão César Giacomini Penteado, Fábio Dacêncio Pereira e Rodolfo Barros Chiaramonte.

Agradeço aos amigos de graduação e da vida Hugo Wesley Kudo, Lúcio Felippe de Mello Neto, Marcelo José de Morais Filho e Wladimir Henrique Mangelardo Barbosa por dividirem comigo ao longo desses anos as preocupações e alegrias.

Agradeço à professora Maria Cristina A. Almeida e ao professor Luiz Fernandes Galante sempre prontos a esclarecer as possíveis dúvidas matemáticas no decorrer do desenvolvimento do trabalho.

Obrigado a todos que colaboraram direta ou indiretamente para a realização deste trabalho.

BUGATTI, Pedro Henrique. **Implementação e Avaliação do Algoritmo Blowfish em C, Java e Microcontroladores PIC**. 2005. 145f. Monografia (Graduação em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

As redes de computadores possibilitaram a troca de informações entre sistemas computacionais e seus usuários de forma generalizada e eficiente do ponto de vista de velocidade de acesso e troca dessas informações. No entanto, também gerou a necessidade de garantir um maior nível de segurança das informações que fluem nessas redes. A arte da criptografia assim como o uso de novas técnicas e novas metodologias como assinatura digital, certificados digitais e outras, possibilitam o aumento de integridade, confiabilidade e também a proteção para as informações e segurança para usuários de um sistema de comunicação de dados. Nesse projeto foram estudadas as características e eficiências de algoritmos criptográficos simétricos, em especial, o algoritmo Blowfish. O algoritmo foi implementado em software, utilizando linguagem C e Java e avaliado seu desempenho, posteriormente o mesmo procedimento foi realizado em hardware usando o *assembly* do microcontrolador PIC16F628.

Palavras Chave: Criptografia. Algoritmos Simétricos. Blowfish. Microcontroladores. Avaliação de Desempenho.

BUGATTI, Pedro Henrique. **Implementação e Avaliação do Algoritmo Blowfish em C, Java e Microcontroladores PIC**. 2005. 145f. Monografia (Graduação em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

The computer networks made possible the exchange of information among computational systems and its users in a general way, increasing the speed of access and exchange of information, improving the efficiency level. However, also it generated the necessity to guarantee an increase in the security level of the information that flow in these networks. The art of cryptograph as well as the use of new techniques and new methodologies like digital signature, digital certificates and others, make possible the increase of integrity, trustworthiness and also the protection for the information and security for users of a data communication system. In this project was studied the features and efficiencies of symmetrical cryptographic algorithms, in special, the Blowfish algorithm. The algorithm was implemented in C and Java language, evaluated its performance, and later, the same process was performed in hardware using the assembly of microcontroller PIC16F628.

Keywords: Cryptograph. Symmetrical Algorithms. Blowfish. Microcontrollers. Evaluation of Performance.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AES: *ADVANCED ENCRYPTION STANDARD*

DES: *DATA ENCRYPTION STANDARD*

NBS: *NATIONAL BUREAU OF STANDARDS*

NIST: *NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY*

NSA: *NATIONAL SECURITY AGENCY*

MDS: *MAXIMUM DISTANCE SEPARABLE*

PHT: *PSEUDO-HADAMARD TRANSFORM*

FIPS: *FEDERAL INFORMATION PROCESSING STANDARD*

SUMÁRIO

1 INTRODUÇÃO

Com a proliferação de computadores e sistemas de comunicações houve uma crescente utilização das redes de computadores e o conseqüente fluxo de informação, fez-se então necessário a utilização de técnicas para proporcionar confiabilidade para seus usuários e integridade dessas informações. Atualmente uma das técnicas mais utilizadas para esse fim é a criptografia (MENEZES; OORSCHOT; VANSTONE, 1997).

A criptografia através dos tempos tem sido uma arte praticada desde o antigo Egito até os dias atuais onde obteve grande importância na proteção de informações digitais. As técnicas de criptografia podem ser divididas em dois tipos: chave simétrica e chave assimétrica. Nos algoritmos de chave simétrica é utilizada uma única chave para cifrar e para decifrar a informação, a chave única é mais simples e rápida; no entanto, apresenta pouca eficiência no aspecto segurança.

Nos algoritmos de chave assimétrica são utilizadas duas chaves, a chave pública para cifrar a informação e a chave privada para decifrá-la. A chave pública pode ser amplamente conhecida mas a privada é conhecida apenas pelo receptor da mensagem aumentando a eficiência no aspecto segurança.

Vários algoritmos criptográficos vêm sendo propostos e pesquisados, os mais conhecidos são o DES (*Data Encryption Standard*) algoritmo criado pela IBM e que possui chave simétrica e o RSA conhecido pelas iniciais de seus três criadores (Rivest, Shamir, Adleman), algoritmo que possui chave assimétrica (TANENBAUM, 2003).

O algoritmo Blowfish estudado no presente trabalho tem grande aceitação no mercado sendo utilizado em diversas aplicações e não são conhecidos ataques sobre ele, além de ser mais rápido que o DES, não patenteado. A razão para sua implementação é devido a sua complexidade, pois após o entendimento de suas estruturas e funcionamento, outros

algoritmos simétricos que utilizam estruturas mais sofisticadas serão passíveis de melhor compreensão.

As vantagens da implementação de criptografia em hardware são a velocidade e segurança. Velocidade devido à existência de um hardware dedicado, ou seja, não acontece como na criptografia em software, onde o software deve executar em conjunto com outras aplicações e dados do sistema. Assim, em implementações de criptografia em hardware a segurança pode ser maior, uma vez que pode ser seguramente encapsulada para prover uma proteção física. Por outro lado, a criptografia em software é mais flexível e não apresenta problemas de memória, visto que as máquinas que implementam estes algoritmos em software dispõem de muita memória além do necessário para a sua implementação.

Assim neste projeto alguns algoritmos simétricos foram implementados tanto em software através de linguagens de alto nível como C e Java, como também em hardware através de microcontroladores.

1.1 Objetivos

O presente trabalho de pesquisa tem como objetivo principal o estudo, implementação e avaliação de desempenho do algoritmo criptográfico simétrico Blowfish, em linguagens de alto nível tais como C e Java e, posteriormente o estudo e proposta de implementação desse algoritmo em um microcontrolador da família PIC.

1.2 Trabalhos Correlatos

A seguir se apresenta uma rápida descrição de alguns trabalhos correlatos de importância relevante para a área de criptografia e para o desenvolvimento dessa proposta de trabalho.

O algoritmo Blowfish foi desenvolvido em 1993 por Bruce Schneier como uma alternativa mais rápida que o DES (*Data Encryption Standard*). O Blowfish é um algoritmo de cifra de bloco simétrica de 64 bits de chave variável entre 32 e 448 bits. Além disso, o algoritmo é não patenteado e livre de licença (SCHNEIER, 1993).

A criptografia realizada pelo Blowfish é feita através de 16 iterações utilizando o método *Feistel Network*. Objetivando aumentar a eficiência, foi escolhido usar na confecção do algoritmo funções simples para os microprocessadores, tais como XOR, adição e multiplicação modular (SCHNEIER, 1993).

O Twofish é um algoritmo de cifra de bloco de 128 bits, possui um comprimento de chave que pode variar de 128 bits a 256 bits. Foi desenvolvido por Bruce Schneier, John Kelseyy, Doug Whitingz, David Wagnerx, Chris Hall e Niels Ferguson em 1998. É um algoritmo muito forte e amplamente utilizado (SCHNEIER et al., 1998).

O esquema de cifragem do Twofish consiste em tratar a mensagem em blocos de 128 bits, utilizando chaves de tamanho variáveis. Assim como o Blowfish, a fim de aumentar a segurança, ele realiza 16 interações durante a criptografia utilizando o método *Feistel Network*. Possui ainda 4 *S-Boxes* de 8 por 8 bits, além disso o algoritmo usa a técnica de *whitenning*, que consiste em esconder a entrada e a saída da primeira e da décima sexta interação respectivamente, através de uma porta XOR. São utilizadas também matrizes MDS (*Maximum Distance Separable*) e PHT (*Pseudo-Hadamard Transform*) (SCHNEIER et al., 1998).

O CAST-128 é um algoritmo de cifra de bloco simétrica de 64 bits, possui um comprimento de chave que pode variar de 40 bits a 128 bits. Foi criado em 1996 por Carlisle Adams e Stafford Tavares usando o procedimento CAST (ADAMS, 1997).

O esquema de cifragem do CAST-128 consiste em tratar a mensagem em blocos de 64 bits. Assim como os outros dois algoritmos (Blowfish e Twofish), a fim de aumentar a segurança, ele realiza 16 iterações durante a criptografia utilizando o método *Feistel Network*.

O uso de algoritmos simétricos é devido ao bom desempenho, rapidez na cifragem e decifragem das informações. Funcionam muito bem com chaves pequenas, uma chave de criptografia de 128 bits torna um algoritmo simétrico quase impossível de ser quebrado.

Foram escolhidos os algoritmos simétricos Blowfish, Towfish e CAST-128 por conjugarem as características segurança e eficiência. Neste trabalho de TCC – Trabalho de Conclusão de Curso, dedicaremos atenção principalmente ao algoritmo Blowfish.

1.3 Material e Métodos

Para o desenvolvimento do projeto foram utilizados computadores e softwares disponibilizados no Laboratório de Arquitetura de Sistemas (LAS) do UNIVEM.

Para o desenvolvimento e avaliação dos algoritmos criptográficos foram utilizados compiladores das linguagens C Dev-C++ 4.0 e Java J2SDK SE v1.5.0.

Para a implementação do algoritmo avaliado em microcontroladores foram utilizados os equipamentos também disponibilizados no LAS, tais como: osciloscópio, geradores de sinais, ambiente de desenvolvimento de componentes eletrônicos (microcontroladores) utilizando tecnologia das empresas Xilinx com o uso de FPGAs (MORENO et al., 2003).

A implementação do algoritmo Blowfish foi validada através da utilização de vetores de teste.

O desempenho do algoritmo estudado e implementado foi avaliado através de estudos de casos. Numa segunda etapa os algoritmos foram implementados em nível de

hardware com a utilização de microcontroladores, onde foi mantida a flexibilidade e o aumento de velocidade.

1.4 Organização do Trabalho

Este trabalho de TCC está organizado em 6 capítulos, a saber:

Capítulo 1 - Introdução – principais objetivos do trabalho, contexto em que o trabalho está inserido, trabalhos correlatos, metodologia e organização do trabalho.

Capítulo 2 – Conceitos de Criptografia – breve histórico da área de criptografia, contextualização e importância da de criptografia, conceitos básicos e descrição de projetos relevantes para o desenvolvimento da área.

Capítulo 3 – Algoritmo Blowfish - apresenta o funcionamento do algoritmo Blowfish, mostrando em detalhes suas estruturas, processos e funcionamento.

Capítulo 4 – Implementação do Algoritmo Blowfish em C e Java – apresenta e enfatiza a implementação do algoritmo Blowfish nas linguagens C e Java, além de avaliação de desempenho da implementação do algoritmo em ambiente de “software” e a conseqüente análise dos resultados obtidos.

Capítulo 5 – O Blowfish em Microcontroladores – apresenta conceitos sobre arquitetura de microcontroladores e enfatiza a implementação e a eficiência do algoritmo Blowfish em ambiente de *hardware*, utilizando para tanto o microcontrolador PIC16F628.

Capítulo 6 – Conclusões – apresenta os principais resultados obtidos considerando a metodologia e o planejamento adotado, sugerindo atividades consideradas relevantes para a continuidade do projeto.

2 CONCEITOS DE CRIPTOGRAFIA

Neste capítulo são abordados conceitos básicos sobre a criptografia tais como, uma breve introdução sobre do que se trata a criptografia, sua história e evolução, seus aspectos relacionados à segurança da informação, a classificação dos algoritmos criptográficos quanto à utilização de cifras simétricas ou assimétricas e por fim uma breve descrição do projeto AES (*Advanced Encryption Standard)*.

2.1 Breve Introdução à Criptografia

A palavra criptografia vem das palavras gregas que significam “escrita secreta”.

A criptografia é uma ferramenta que pode ser usada para manter informações confidenciais e garantir sua integridade e autenticidade. Segundo Tanembaum (2003) todos os sistemas criptográficos modernos se baseiam no princípio de Kercknoff de um algoritmo publicamente conhecido e uma chave secreta. Muitos algoritmos criptográficos usam transformações complexas que envolvem substituições e permutações para transformar o texto simples (*plain text*) em texto cifrado (*cipher text*).

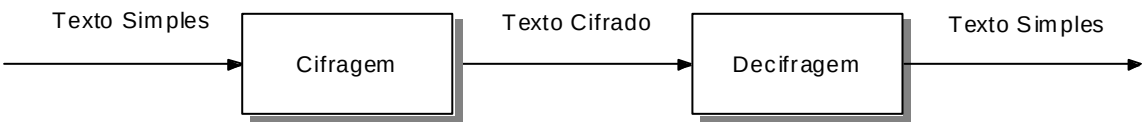

Figura 2.1 – Processos de Cifragem e Decifragem

A figura 2.1 ilustra os processos pelos quais a mensagem passa durante a criptografia. As mensagens a serem criptografadas (cifradas), conhecidas como texto simples, são transformadas por uma função que é parametrizada por uma chave. Em seguida, a saída

do processo de criptografia, conhecida como texto cifrado é transmitida normalmente por um meio de comunicação (TANEMBAUM, 2003).

2.2 História da Criptografia

A criptografia tem uma longa e fascinante história. É utilizada desde a época da escrita hieroglífica dos egípcios até os dias atuais, mas foi no século XX, onde desempenhou um papel crucial em ambas as guerras mundiais, onde houve seu grande desenvolvimento. A criptografia, portanto, sempre esteve associada com o meio militar, serviços diplomáticos e governamentais, sendo usada como uma ferramenta para proteção de segredos nacionais e estratégias. A partir do período da Segunda Guerra mundial com a invenção dos computadores foi possível a realização de centenas de cálculos em segundos proporcionando assim uma grande mudança nos métodos criptográficos. A criptografia sofreu a transação de arte para ciência (TANEMBAUM, 2003).

A proliferação dos computadores e dos sistemas de comunicação na década de 60 trouxe uma demanda do setor privado por meios de proteção da informação em seu formato digital e por serviços de segurança.

De acordo com Moreno; Pereira; Chiaramonte (2004) o algoritmo DES (*Data Encryption Standard* – Padrão de Criptografia de Dados) foi iniciado com o trabalho de Horst Feistel na IBM em meados dos anos 70, onde os algoritmos eram secretos, principalmente os utilizados pelas forças armadas e culminando em 1977 onde foi publicado no *National Bureau of Standards* (NBS). O DES foi o primeiro algoritmo criptográfico de conhecimento público e a partir de janeiro de 1977 o governo dos Estados Unidos o adotou como padrão oficial para informações não-confidenciais, se tornando assim o mais bem conhecido mecanismo criptográfico.

Segundo Stallings (2003) um grande impacto na história do desenvolvimento da criptografia ocorreu em 1976 quando Diffie e Hellman publicaram o artigo *New Directions in Cryptograph* que introduzia o conceito revolucionário de criptografia de chave pública e também um novo método ainda ingênuo para troca de chave, a segurança na qual se baseava era o difícil problema do logaritmo Discreto para se obter a chave secreta a partir da chave pública.

Embora os autores não tivessem realizado na prática o esquema de cifragem de chave pública, a idéia estava clara e gerou extensivo interesse e atividade na comunidade criptográfica.

A seguir outros acontecimentos relacionados à utilização da criptografia (TOKTZ, 2003):

Ano	Descrição
1943	Máquina Colossus projetada para quebrar códigos.
1969	James Ellis desenvolveu um sistema de chaves públicas e chaves privadas separadas.
1976	Diffie-Hellman é um algoritmo baseado no problema do logaritmo discreto, é o criptossistema de chave pública mais antigo ainda em uso.
1976	A IBM apresenta a cifra Lucifer ao NBS (*National Bureau of Standards*), o qual, após avaliar o algoritmo com a ajuda da NSA (*National Security Agency*), introduz algumas modificações (como as Caixas S e uma chave menor) e adota a cifra como padrão de criptografia de dados nos EUA (FIPS46-3, 1999), conhecido hoje como DES (*Data Encryption Standard*).

Ano	Descrição(cont.)
	Hoje o NBS é chamado de NIST (*National Institute of Standards and Technology*).
1977	Ronald L. Rivest, Adi Shamir e Leonard M. Adleman começaram a discutir como criar um sistema de chave pública prático. Ron Rivest acabou tendo uma grande idéia e a submeteu à apreciação dos amigos: era uma cifra de chave pública, tanto para confidenciabilidade quanto para assinaturas digitais, baseada na dificuldade da fatoração de números primos grandes. Foi batizada de RSA, de acordo com as primeiras letras dos sobrenomes dos autores.
1978	O algoritmo RSA é publicado na ACM (*Association for Computing Machinery*), uma dos melhores meios de divulgação de pesquisas científicas.
1990	Xuejia Lai e James Massey publicaram na Suíça "*A Proposal for a New Block Encryption Standard*" ("Uma Proposta para um Novo Padrão de Encriptação de Bloco" – LAI, 1990), o assim chamado IDEA (*International Data Encryption Algorithm*), para substituir o DES. O algoritmo IDEA utiliza uma chave de 128 bits e emprega operações adequadas para computadores de uso geral, tornando as implementações em software mais eficientes (SCHNEIER, 1996).
1991	Phil Zimmermann torna pública sua primeira versão de PGP (*Pretty Good Privacy*) como resposta ao FBI, o qual invoca o direito de acessar qualquer texto claro de comunicações entre usuários que se comunicam por meio de uma de rede de comunicação digital.

Ano	Descrição(cont.)
	PGP oferece alta segurança para o cidadão comum e, como tal, pode ser encarado como um concorrente de produtos comerciais como o *Mailsafe* da RSADSI.
1994	Novamente o professor Ronald L. Rivest, autor dos algoritmos RC2 e RC4 incluídos na biblioteca de criptografia BSAFE do RSADSI, publica a proposta do algoritmo RC5 na Internet. Esse algoritmo usa rotação dependente de dados como sua operação não linear e é parametrizado de modo que o usuário possa variar o tamanho do bloco, o número de estágios e o comprimento da chave.
1994	O algoritmo Blowfish, uma cifra de bloco de 64 bits com uma chave de até 448 bits de comprimento, é projetado por Bruce Schneier (SCHNEIER, 1993).
1997	O PGP 5.0 *Freeware* é amplamente distribuído para uso não comercial.
1997	O código DES de 56 bits é quebrado por uma rede de 14.000 computadores
1998	O código DES é quebrado em 56 horas por pesquisadores do Vale do Silício (DESKEY, 2001).
1999	O DES é quebrado em apenas 22 horas e 15 minutos, mediante a união da *Electronic Frontier Foundation* e a *Distributed.Net*, que reuniram em torno de 100.000 computadores pessoais ao DES Cracker pela Internet (MESERVE, 1999).

Ano	Descrição(cont.)
2000	O NIST (*National Institute of Standards and Technology*) anunciou um novo padrão de uma chave secreta de cifragem, escolhido entre 15 candidatos. Esse novo padrão foi criado para substituir o algoritmo DES, cujo tamanho das chaves tornou-se insuficiente para conter ataques de força bruta O algoritmo Rijndael, cujo nome é uma abreviação dos nomes dos autores Rijmen e Daemen, foi escolhido para se tornar o futuro AES (*Advanced Encryption Standard*) (FIPS197, 2001).
2000 - 2004	Muitos professores e profissionais da computação com vínculo em centros de pesquisa, universidades e empresas motivam-se e começam a pesquisar novas formas de implementar algoritmos e soluções de segurança. Surge, assim, uma "onda" de pesquisas e desenvolvimentos voltados a realizar otimizações dessas primeiras implementações e uma dessas tendências é a implementação em hardware.

2.3 Segurança da Informação e Criptografia

Devido ao advento da Internet e sua grande proliferação surgiram vários tipos de transações utilizando a informação digital, levando a uma necessidade da transmissão dessas informações de modo seguro. A solução para esse problema é a criptografia.

No entanto a criptografia não possui apenas o atributo de manter o sigilo de informação.

De acordo com Menezes; Oorschot; Vanstone (1997) criptografia é o estudo de técnicas matemáticas relacionadas a aspectos da segurança da informação tais como

confidenciabilidade, integridade de dados, autentificação de entidade e autentificação da origem dos dados, ou seja, criptografia não é apenas um meio de prover informação segura, mas também algumas outras técnicas.

De todos os tópicos para se obter a segurança da informação em uma rede, os listados a seguir formam a base a partir onde os outros derivaram (MENEZES; OORSCHOT; VANSTONE, 1997):

1 – Confidenciabilidade: é um serviço usado para manter o conteúdo da informação fora do alcance daqueles que não tem autorização para tê-la.

2 – Integridade dos Dados: é um serviço que identifica a alteração de dados não autorizada.

3 – Autentificação: é um serviço relacionado a autentificação de ambas as partes, entidade (pode-se entender entidade como usuário) e informação relacionadas numa comunicação.

4 – Não repúdio: o receptor deseja provar que foi realmente o transmissor correto quem envio os dados.

Com o objetivo de se obter a segurança da informação, “ferramentas” criptográficas (primitivas) são utilizadas para prover tal segurança. Na figura 2.2 é mostrada uma taxonomia dessas primitivas (MENEZES; OORSCHOT; VANSTONE, 1997). Em relação ao projeto abordado foram estudados especificamente, algoritmos criptográficos que utilizem primitivas de chave simétrica com cifras simétricas e de bloco, como o algoritmo Blowfish.

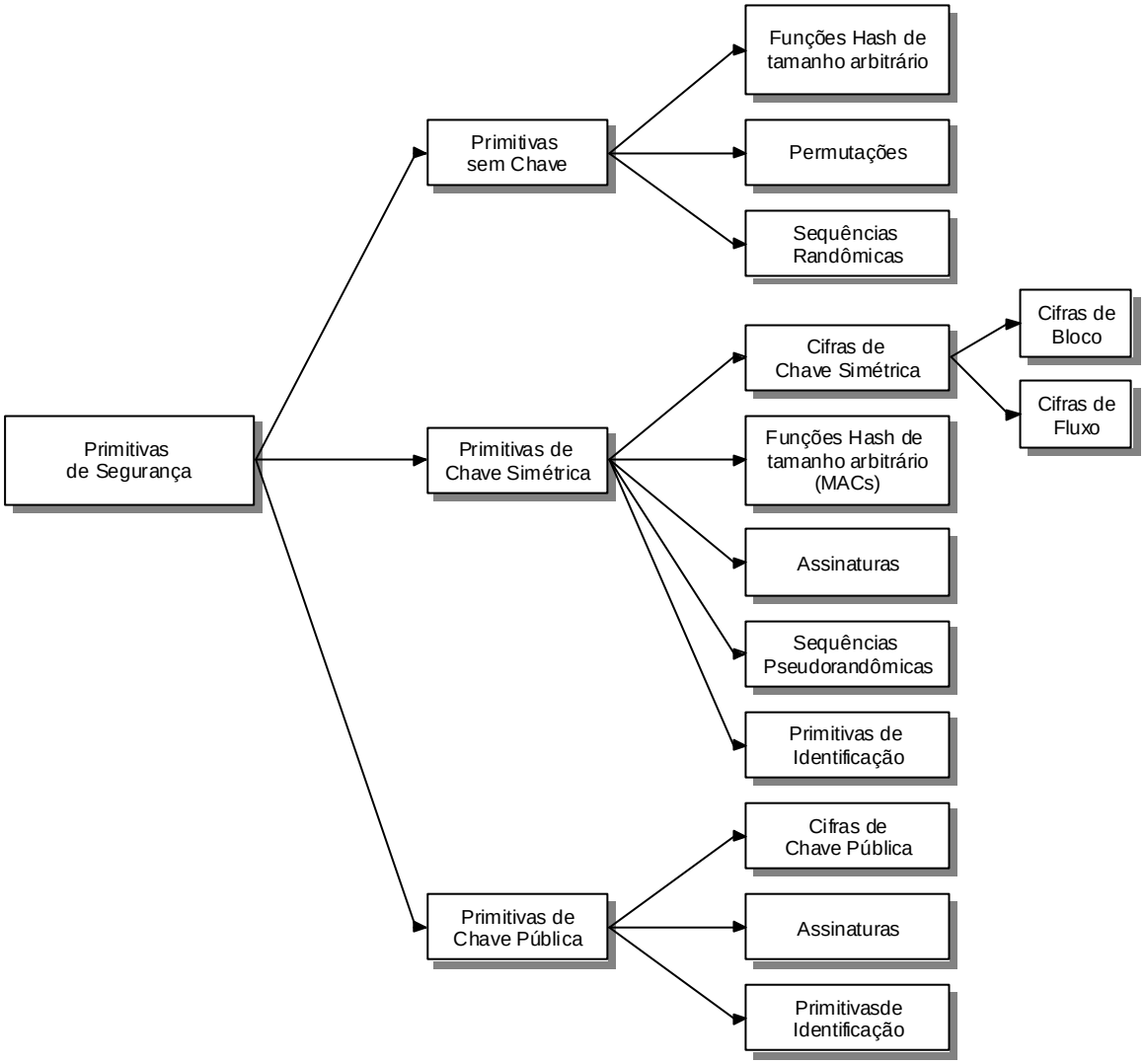

Figura 2.2 - Uma taxonomia de primitivas criptográficas (MENEZES; OORSCHOT; VANSTONE, 1997)

2.4 Classificação

Os algoritmos de criptografia podem ser classificados de acordo com a utilização das chaves de criptografia, assim, tais algoritmos podem ser divididos em dois tipos principais: Algoritmos Simétricos e Algoritmos Assimétricos.

2.4.1 Algoritmos Simétricos

Os algoritmos simétricos são assim denominados pois utilizam uma única chave para cifrar e decifrar os dados, constituindo, portanto, uma simetria. A criptografia simétrica é o método mais tradicional utilizado para se criptografar (cifrar) os dados.

A simetria resultante do uso de uma única chave para cifrar e decifrar a informação acarreta alguns problemas tais como: é necessário estabelecer um modo de informar ao destinatário a chave para decifrar de forma segura, e se essa forma segura de envio for alcançada seria melhor utilizá-la para transmitir a informação de uma vez ao invés de criptografá-la (cifrá-la).

Apesar de possíveis problemas abordados, a criptografia simétrica é eficiente em conexões seguras na Internet onde é necessária a troca de senhas temporárias para algumas transmissões críticas. Outro benefício da criptografia simétrica é o bom desempenho e a rapidez na cifragem e decifragem das informações.

Nas figuras 2.3 e 2.4 a seguir, pode-se observar o funcionamento da criptografia simétrica. Uma informação é cifrada através de um polinômio utilizando-se de uma chave que também serve para decifrar novamente a informação.

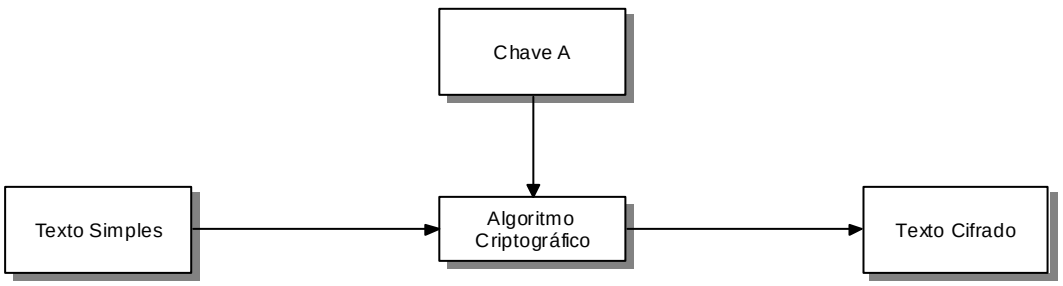

Figura 2.3 - Criptografia Simétrica – Processo de Cifragem

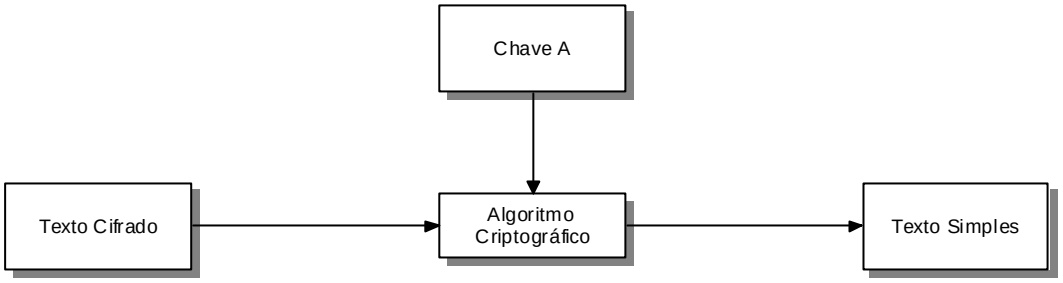

Figura 2.4 – Criptografia Simétrica – Processo de Decifragem

2.4.2 Algoritmos Assimétricos

Na criptografia de chave assimétrica (ou criptografia de chave pública) são usadas duas chaves ligadas matematicamente; o processo de cifragem utiliza uma das chaves para cifrar a informação, portanto a outra chave será utilizada pelo processo de decifragem para decifrar a informação.

Uma das chaves é mantida em segredo e é denominada chave privada. A outra chave denominada de chave pública é disponibilizada a todos, sendo assim, quando uma pessoa quiser enviar uma informação ela irá cifrá-la usando a chave pública, e a informação só poderá ser decifrada com a chave privada do destinatário.

A técnica de criptografia assimétrica pode ser utilizada para assinatura digital e autentificação.

É possível combinar a criptografia simétrica com a assimétrica, somando a segurança com a rapidez, pois os sistemas criptográficos assimétricos geralmente não são tão computacionalmente eficientes quanto os simétricos, portanto eles são utilizados em conjunto para prover facilidades de distribuição da chave e capacidades de assinatura digital.

Um ponto forte da criptografia assimétrica é o de não ser necessário o envio da chave para decifrar a informação como ocorre na criptografia simétrica.

Com a chave pública pode-se cifrar informações que só poderão ser decifradas pelo proprietário da chave privada, num processo unidirecional como pode ser observado nas figuras 2.5 e 2.6.

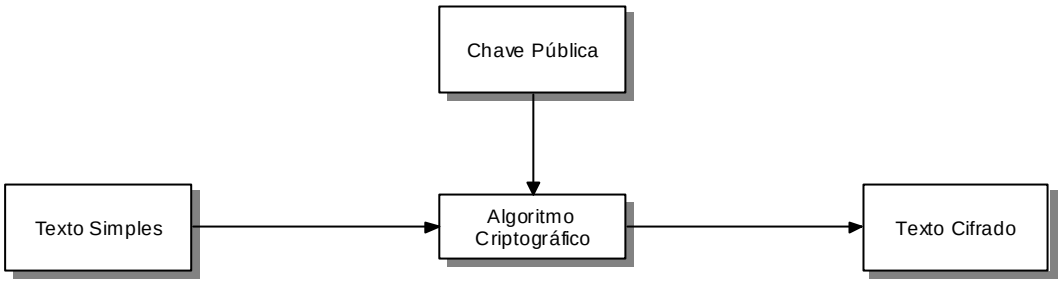

Figura 2.5 – Criptografia Assimétrica – Processo de Cifragem

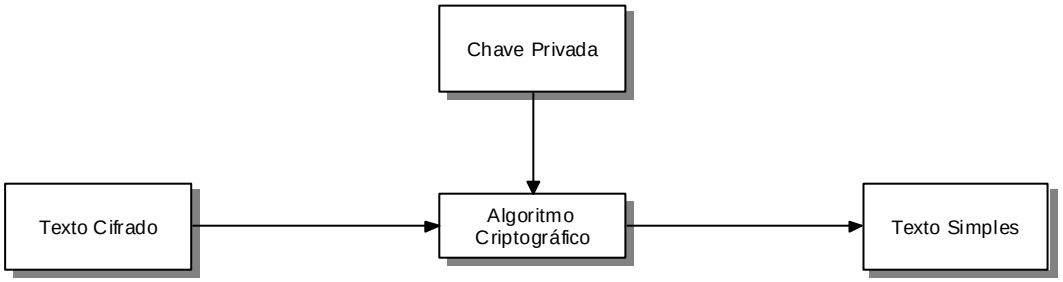

Figura 2.6 – Criptografia Assimétrica – Processo de Decifragem

2.4.3 Diferenças entre Algoritmos Simétricos e Assimétricos

O método de cifragem que utiliza chave simétrica tanto quanto o que utiliza chave assimétrica (ou pública) possuem várias vantagens e desvantagens, algumas delas comuns aos dois esquemas. Segundo Menezes; Oorschot; Vanstone (1997) algumas vantagens e desvantagens dos dois métodos são apresentadas respectivamente nas tabelas 2.1 e 2.2.

Tabela 2.1 – Vantagens das criptografias de chave simétrica e assimétrica

Simétrica	Cifras de chave simétrica podem ser desenvolvidas para ter altas taxas de saídas de dados. Algumas implementações em hardware alcançam taxas de cifragem de centenas de me gabytes por segundo (mbytes/s), enquanto que implementações em software alcançam taxas de megabytes por segundo.
	Chaves para cifras de chave simétrica são relativamente curtas.
	Cifras de chave simétrica podem ser empregadas como primitivas para construir vários mecanismos criptográficos incluindo geradores de números pseudo-randômicos.
	Cifras de chave simétrica podem ser compostas para produzir cifras mais fortes. Simples transformações que são fáceis de se analisar, mas se sozinhas são fracas, podem ser usadas para construir cifras fortes.
Assimétrica	Apenas a chave privada deve permanecer secreta.
	Dependendo do modo de uso o par de chaves pública e privada permanece o mesmo por um tempo considerável
	Muitos esquemas de chave pública têm desempenho relativamente eficiente em mecanismos de assinatura digital.
	Em uma rede ampla, o número de chaves é considerado muito menor do que no esquema de chave simétrica.

Tabela 2.2 – Desvantagens das criptografias de chave simétrica e assimétrica

Simétrica	A chave deve permanecer secreta na comunicação
	Em uma rede ampla existem muitas chaves, conseqüentemente, a manipulação de tantas chaves se torna uma desvantagem.

Tabela 2.2 – Desvantagens das criptografias de chave simétrica e assimétrica

Assimétrica	O desempenho em relação à rapidez de cifragem e decifragem do esquema de chave assimétrica se comparada com o de chave simétrica é bem inferior.
	O tamanho das chaves requeridas pela criptografia de chave assimétrica é maior do que os requeridos pela criptografia de chave simétrica.
	A criptografia de chave assimétrica não tem uma história tão extensa quanto a criptografia de chave simétrica.

Portanto, a criptografia simétrica e a criptografia assimétrica têm vantagens complementares, sendo assim, cada esquema pode explorar os pontos fortes do outro como, por exemplo (MENEZES; OORSCHOT; VANSTONE, 1997):

1. a criptografia de chave assimétrica facilita assinaturas eficientes (particularmente o não repúdio) e também a manipulação de chaves.

2. a criptografia de chave simétrica é eficiente para cifrar e para algumas aplicações de integridade de dados.

2.5 Projeto AES

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

Devido a controvérsias existentes no DES sobre suspeitas do NSA (*National Security Agency),* o órgão do governo americano especializado em decifrar códigos, ter ocultado uma

“porta dos fundos” (*backdoor*) que pudesse facilitar ainda mais a decifração do DES por parte dela, se o NIST anunciasse um novo padrão os especialistas em criptografia concluiriam automaticamente que a NSA havia mesmo criado uma “porta dos fundos” no DES, e assim ela poderia ler tudo que fosse 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 de implementações em software e hardware. O algoritmo teria de ser público ou licenciado em termos não-discriminatórios.

Foram selecionadas 15 propostas sérias na fase inicial, os algoritmos foram (NIST02, 1998): 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 meados de 1998, a decisão tomada foi baseada nos requisitos de segurança, eficiência, simplicidade, flexibilidade e memória (importante para sistemas embarcados) (NIST03, 1997).

Os cinco finalistas foram (NIST01, 1999):

1. Rijndael (de Joan Daemen e Vincent Rijmen, 86 votos)
2. Serpent (de Ross Anderson, Eli Biham e Lars Knudsen, 59 votos)

3. Twofish (de uma equipe liderada por Bruce Schneier, 31 votos)
4. RC6 (da RSA Laboratories, 23 votos)
5. MARS (da IBM, 13 votos)

No ano 2000, o NIST anunciou que o Rijndael finalmente seria o escolhido e em 2001, ele se tornou o novo padrão tão desejado, conhecido como AES (*Advanced Encryption Standard*) e foi publicado pelo *Federal Information Processing Standard* FIPS197 (FIPS197, 2001).

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

A tabela 2.3 apresenta os algoritmos selecionados na fase inicial e seus respectivos países e autores:

Tabela 2.3 – 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 2.4 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 2.4 – Características Gerais dos concorrentes do AES (BIHAM, 1999)

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

2.5.1 Breve descrição dos 5 finalistas

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

A implementação do MARS se dá basicamente em três fases. A primeira fase implementa uma rápida mistura dos dados do texto fonte, a inserção da chave e 8 voltas da transformação *Type-3 Feistel*. A segunda fase é a fase mais importante do processo, ela

consiste de 16 voltas da transformação *Type-3 Feistel*, destas 16 voltas, 8 foram implementadas em *forward mode* e 8 em *backward mode*. A terceira e última fase são, essencialmente, o inverso da primeira fase.

O algoritmo RC6 (RIVEST, et al. 1998) foi desenvolvido por 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 são 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; RIJMEN, 1999) foi desenvolvido por dois pesquisadores belgas. O Rijndael é um algoritmo de blocos iterativos com tamanho de bloco e de chave variáveis, podendo ser especificados independentemente para 128, 192 ou 256 bits.

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

O Serpent (ANDERSON; BIHAM; KNUDSEN, 1999) 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 et al., 1998) possui um bloco de 128 bits e chaves que podem ter um tamanho de 128, 192 ou 256 bits.

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

2.5.2 Comparação dos 5 finalistas

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

2.5.2.1 Complexidade Computacional

O NIST como já visto 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 plataformas, entre outros.

2.5.2.2 Requisitos de Memória

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

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

A tabela 2.5, segundo Schneier e Whiting (2000), especifica a memória utilizada pelos algoritmos implementados em *smart cards*.

Tabela 2.5 – Memória Mínima Requerida para Implementação em *smart cards*

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

2.5.2.3 Velocidade de Processamento

A velocidade de processamento é um fator importante, pois 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 e Whiting (2000), mostram em ciclos de *clock*, o desempenho na cifragem dos algoritmos em *Assembler* e na Linguagem C, utilizando os tamanhos de chave exigidos pelo NIST – 128, 192 e 256 bits.

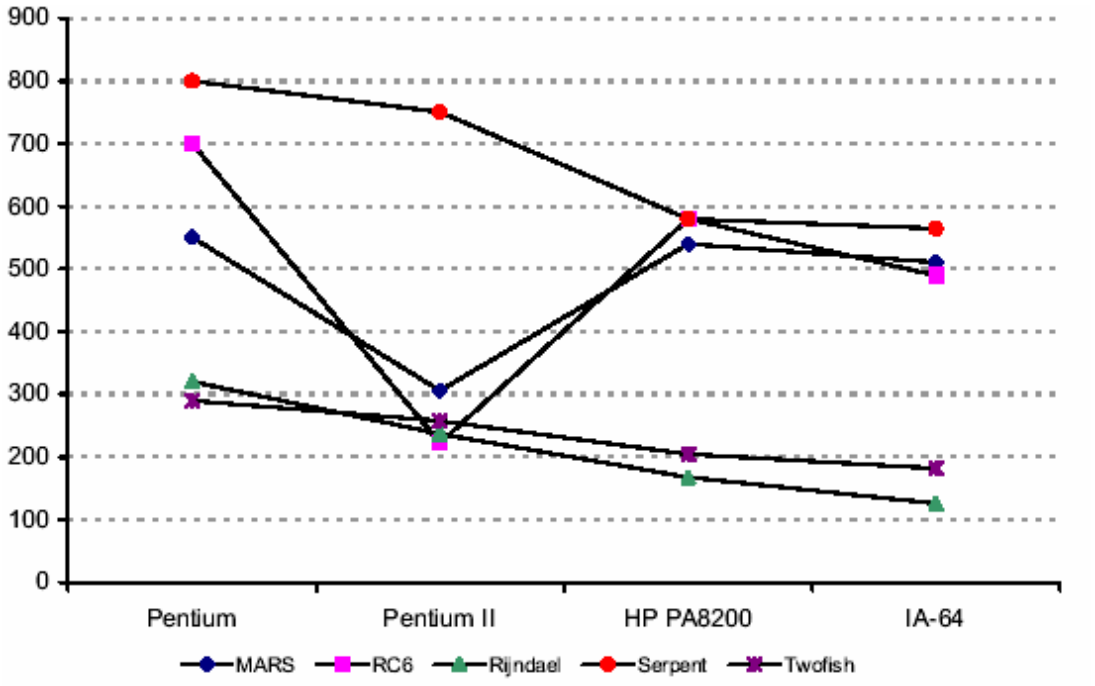

Figura 2.7 – Velocidade de Cifragem para uma chave de 128 bits em Assembler (SCHNEIER; WHITING, 2000)

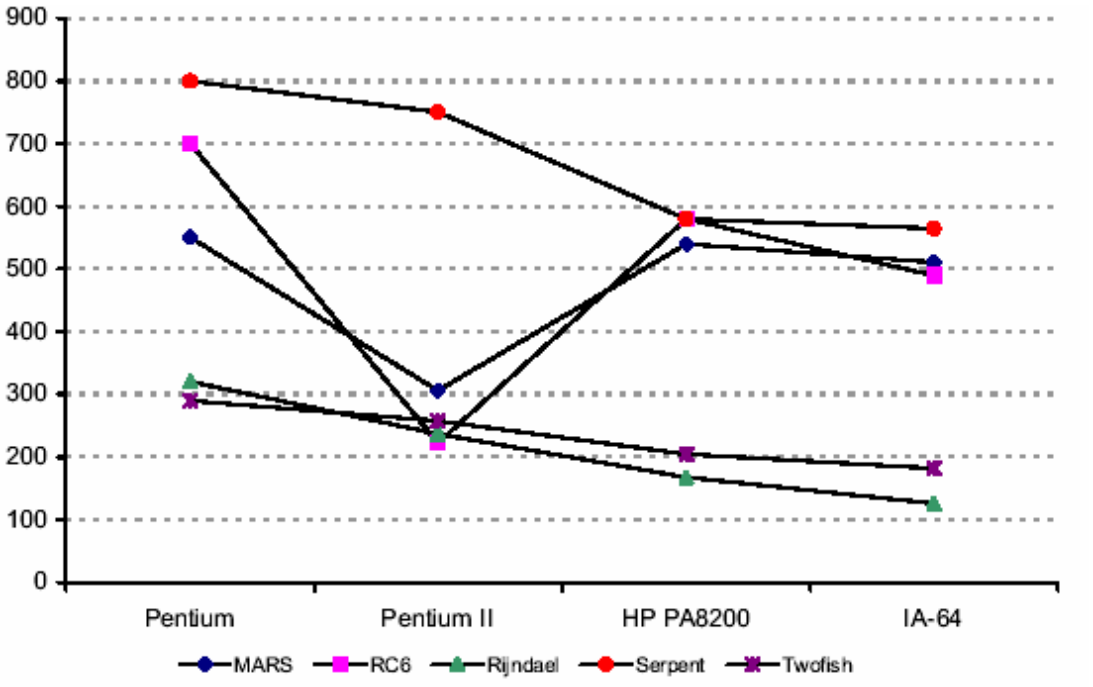

Figura 2.8 - Velocidade de Cifragem para uma chave de 192 bits em Assembler (SCHNEIER; WHITING, 2000)

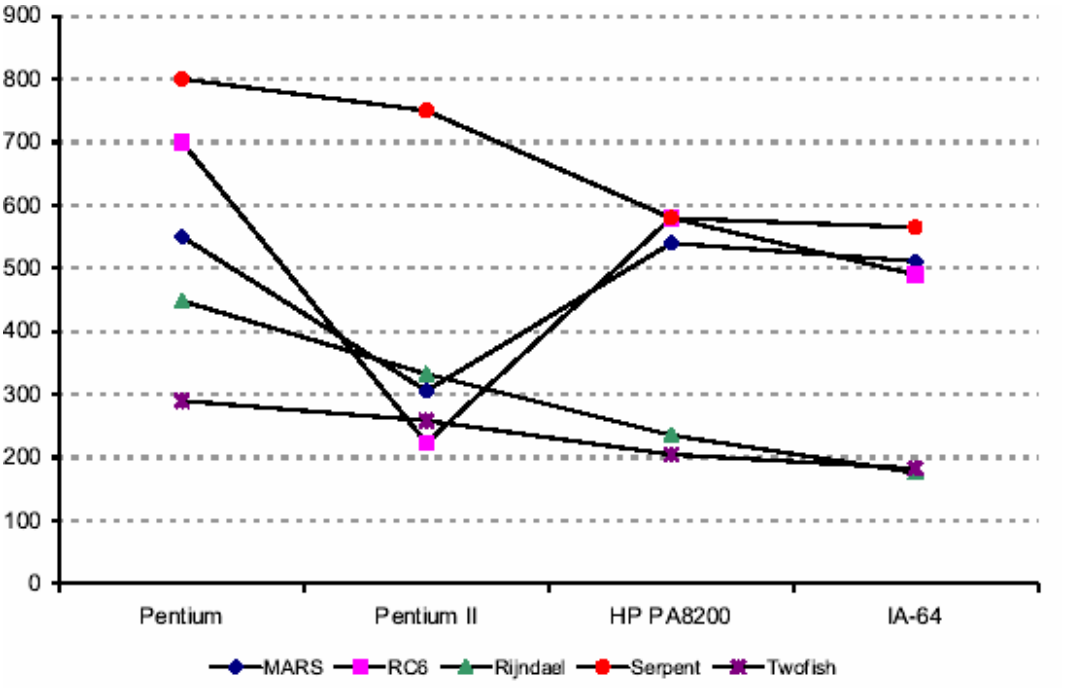

Figura 2.9 - Velocidade de Cifragem para uma chave de 256 bits em Assembler (SCHNEIER; WHITING, 2000)

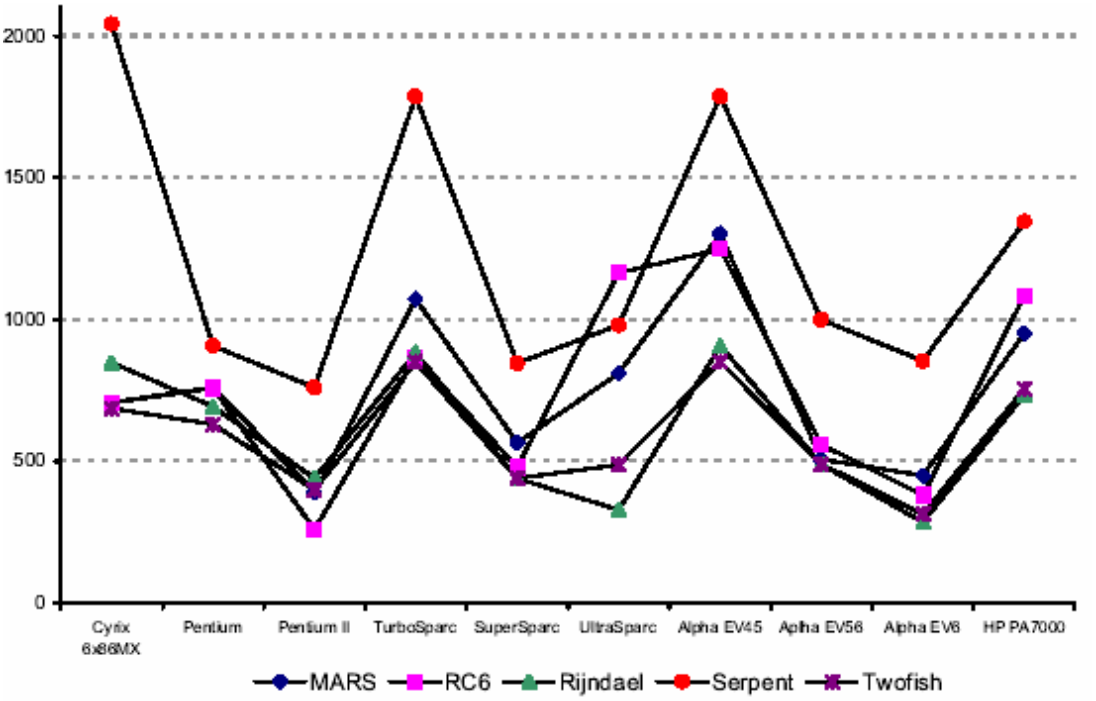

Figura 2.10 - Velocidade de Cifragem para uma chave de 128 bits em C (SCHNEIER; WHITING, 2000)

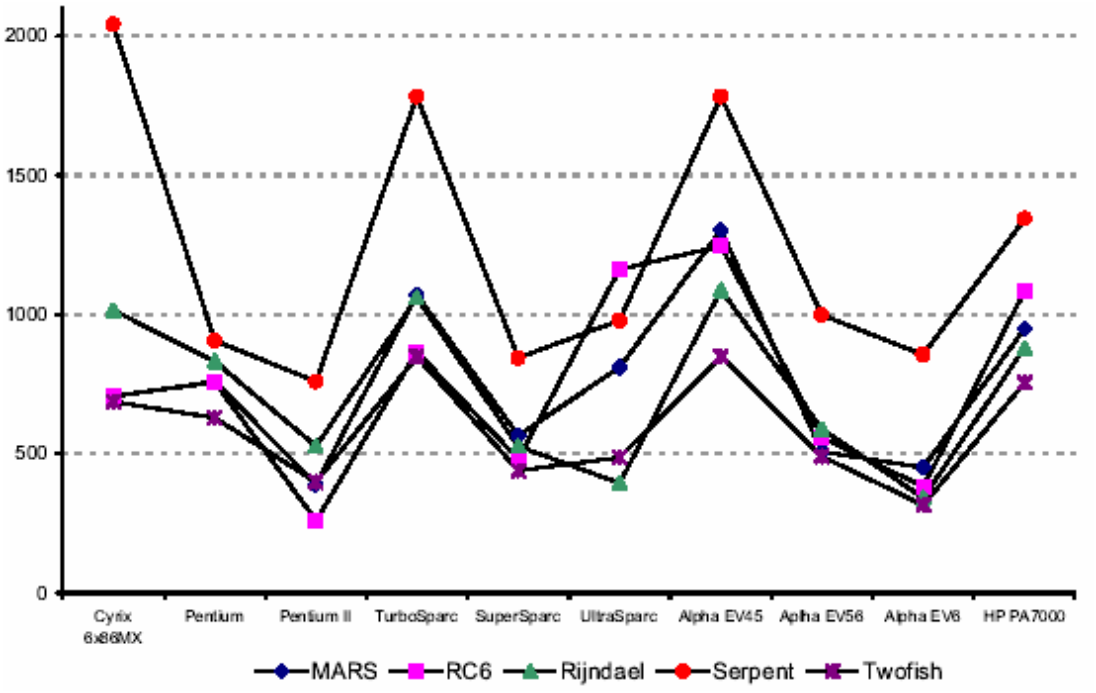

Figura 2.11 - Velocidade de Cifragem para uma chave de 192 bits em C (SCHNEIER; WHITING, 2000)

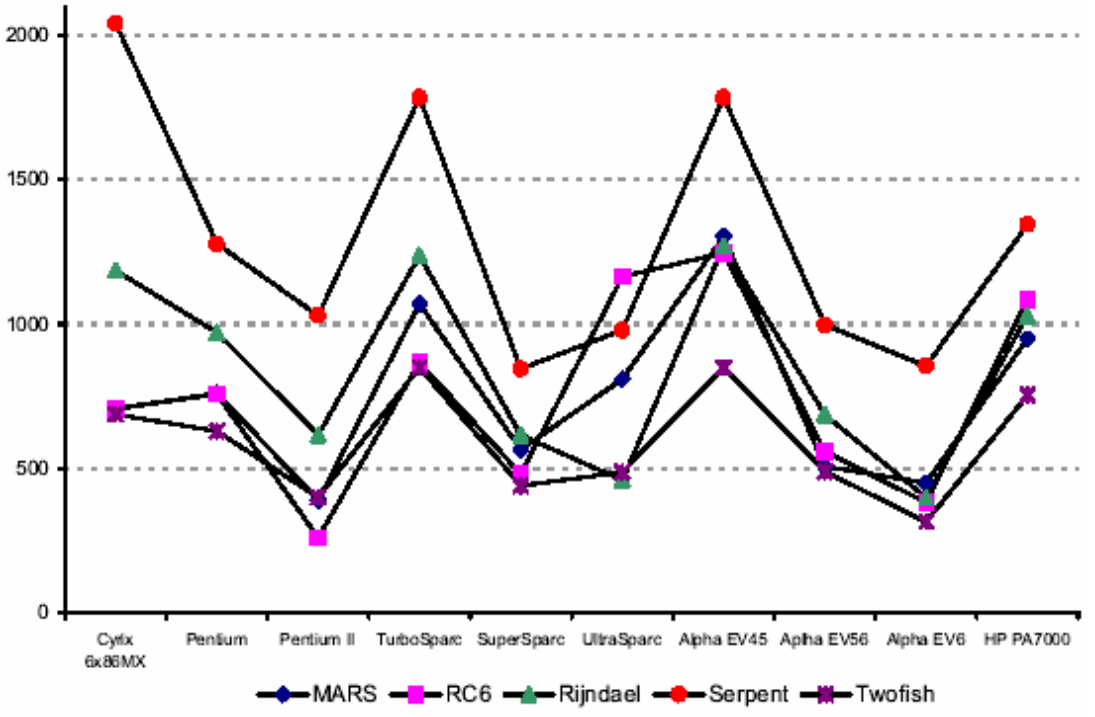

Figura 2.12 - Velocidade de Cifragem para uma chave de 256 bits em C (SCHNEIER; WHITING, 2000)

Observando as figuras 2.7, 2.8 e 2.9 é possível verificar que os algoritmos mais rápidos, em *assembly* são o Rijndael e o Twofish. Observa-se também que a velocidade não varia muito com o aumento da chave.

Observando o desempenho em C, se verifica que o tempo para cifrar é maior quando comparado ao tempo nas respectivas implementações em *assembly*, em torno de 2 a 3 vezes. Em C os algoritmos mais rápidos são Rijndael, Twofish e RC6. Observa-se também que há uma maior variação em função do processador utilizado. Novamente se observa que aumentando o tamanho da chave, não há muita variação no tempo, exceto o algoritmo Rijndael que apresentou uma variação mais considerável.

2.5.2.4 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 2.6 especifica as funções criptográficas utilizadas pelos 5 algoritmos finalistas do projeto AES.

Tabela 2.6 – Utilização das funções criptográficas

	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 realizar a cifragem e a decifragem maior segurança esse proporcionará, 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.

2.6 Considerações Finais

Neste capítulo foi dada ênfase principalmente aos conceitos básicos da criptografia e ao projeto AES.

A atenção dada ao projeto AES foi de extrema importância em virtude de um dos principais tópicos abordados ser a comparação de desempenho entre as diversas implementações de algoritmos criptográficos, fato que será abordado no projeto através da análise de desempenho do algoritmo Blowfish tanto em hardware quanto em software, criado por Bruce Scheneier o mesmo criador do algoritmo Twofish, participante do projeto AES.

3 ALGORITMO BLOWFISH

Este capítulo apresenta o funcionamento do algoritmo Blowfish, mostra em detalhes sua estrutura lógica e estruturas de dados dos principais parâmetros utilizados tanto no processo de cifragem de dados, quanto no processo de decifragem de dados. O objetivo principal desse estudo é o de subsidiar a implementação do algoritmo de forma eficiente nas linguagens C e JAVA.

3.1 Descrição do Algoritmo

O Blowfish é um algoritmo de chave simétrica que utiliza uma cifra de bloco de 64 bits e possui diversos tamanhos de chave que variam entre 32 e 448 bits.

O algoritmo consiste de duas etapas, uma primeira etapa denominada expansão de chave e uma segunda etapa denominada cifragem de dados. A expansão de chave converte uma chave de até no máximo 448 bits em um conjunto de subchaves totalizando 4168 bytes.

A cifragem dos dados ocorre através de uma rede de Feistel com 16 iterações. Essa etapa utiliza um método de cifragem fraco através de várias iterações, de modo a tornar-se um processo complexo. Cada iteração consiste de uma permutação dependente somente da chave e de uma substituição dependente da chave e dos dados envolvidos. Todas as operações realizadas são XORs (Ou Exclusivos) e adições sobre palavras de 32 bits. As funções utilizadas nas iterações, XORS e adições, são funções simples e eficientes quando executadas pelos microprocessadores.

Por se tratar de um algoritmo de cifra de bloco, a cifragem do texto é realizada em blocos e, no caso do Blowfish, os blocos possuem 64 bits tratados em dois grupos de 32 bits.

3.2 Funções Criptográficas utilizadas pelo algoritmo

Além das funções básicas como XOR, ADD e Deslocamentos; o algoritmo Blowfish também utiliza as seguintes estruturas criptográficas denominadas de “S-boxes” e “Rede Feistel”.

3.2.1 A Estrutura de S-Boxes

Uma S-box nada mais é do que uma caixa de substituição na qual é introduzido um texto inicial, sobre o qual são realizadas substituições com o intuito de emaranhá-lo para dificultar sua decifragem (interpretação).

As S-boxes podem variar de tamanho (comprimento) tanto quanto na entrada quanto a saída, podendo também serem fixas ou dependentes de chave. No caso do algoritmo Blowfish são utilizadas 4 S-boxes de 32 bits com 256 entradas cada uma.

3.2.2 Estrutura Rede de Feistel

Segundo Schneier et al. (1998), uma rede de Feistel é um método que transforma qualquer função em uma permutação. Criada por Horst Feistel, um dos idealizadores do algoritmo Lúcifer, se tornou popular e atualmente é utilizada amplamente nos algoritmos criptográficos.

O objetivo de uma rede de Feistel é a obtenção de um método complexo de cifragem a partir de inúmeras repetições de um método simples de cifragem. A rede de Feistel tem como característica tornar uma função não inversível em uma função passível de inversão.

3.3 Funcionamento do Algoritmo

O Blowfish utiliza um vetor P formado por 18 subchaves de 32 bits cada uma e por quatro “S-Boxes”. As subchaves (elemento) do vetor P são denominadas de “P1”, “P2”, “P3”,..., “P18”. Cada S-box contém 256 elementos de 32 bits, totalizando 8192 bits.

3.3.1 Expansão de Chave

A expansão de chave possibilita a criação de subchaves a partir de uma chave primária escolhida aleatoriamente. Essas subchaves são calculadas utilizando o próprio algoritmo Blowfish.

Em um primeiro passo, o vetor P e as quatro S-boxes são preenchidas, seqüencialmente, com uma “string” fixa. Essa “string” consiste de dígitos hexadecimais do número PI (3,1415926535...) excluindo o dígito da parte inteira de (dígito 3). O número na base dez é representado pela seqüência 3,1415926535.... Retirando a parte inteira (digito) obtém-se a seguinte seqüência na base dez 1415926535..... Para transformar essa seqüência para a base hexadecimal é necessário determinar a quantidade de dígitos da parte fracionária do número a ser utilizada. A quantidade de dígitos utilizados gera o comprimento da *string* obtida.

Num segundo passo é realizada uma operação XOR entre os 32 primeiros bits da chave com o primeiro elemento do vetor P (P1), posteriormente é realizada a operação XOR entre os próximos 32 bits da chave com o segundo elemento do vetor P (P2) e assim por diante até completar todos os bits da chave.

Caso os bits da chave se esgotem antes que todos os elementos do vetor P tenham sido utilizados, deve-se recomeçar o processo com os primeiros bits da chave e continuar a operação até que se esgotem todos os elementos do vetor P.

Por exemplo, se a chave aleatória escolhida tivesse um tamanho de 64 bits, seria realizada uma operação XOR entre o valor da primeira posição do vetor P (P1) com os 32 bits mais significativos dessa chave, e uma operação XOR entre o valor da segunda posição do vetor P (P2) com os 32 bits menos significativos da chave. Como pode ser observado, caso os bits que constituem a chave se esgotarem quando utilizado o elemento P2 de P; portanto o processo deve ser reiniciado e as operações XOR retomadas. Assim, serão realizadas: uma operação XOR do elemento P3 com os 32 bits mais significativos da chave, e uma operação XOR entre o elemento P4 com os 32 bits menos significativos da chave, e assim por diante até que seja realizada a operação XOR do elemento P17 do vetor P com os 32 bits mais significativos dessa chave e uma operação XOR entre o elemento P18 do vetor P com os 32 bits menos significativos da chave, esgotando, e dessa forma esgotando todos os elementos do vetor P.

A Figura 3.1 mostra o processo acima descrito de forma gráfica.

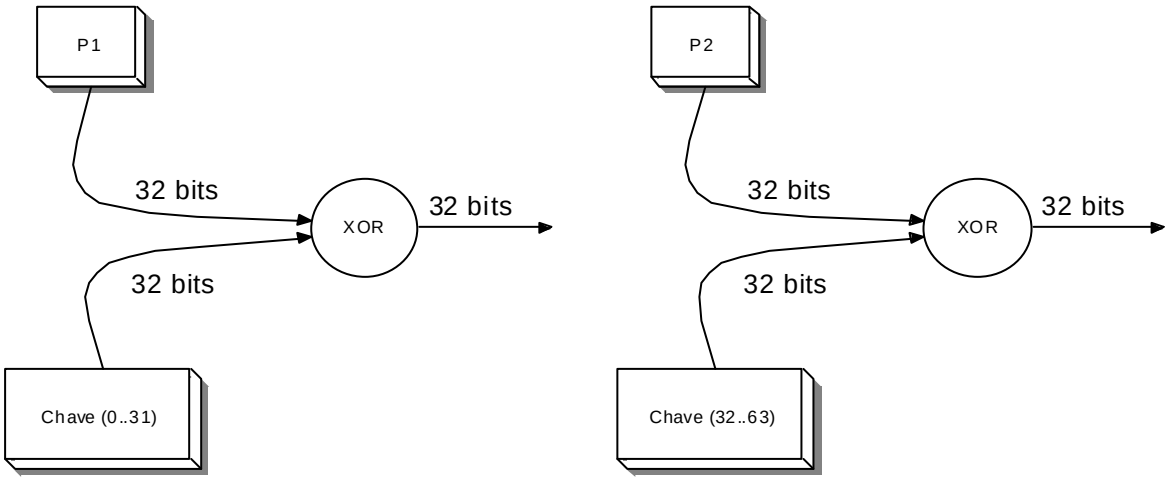

Figura 3.1 – Esquema gráfico para obtenção das subchaves intermediárias P1 e P2

Após esse processo, utilizando-se uma string formada por zeros executa-se o algoritmo Blowfish, utilizando as subchaves já obtidas.

O resultado desse processo deve ser armazenado em P1 e P2 e em seguida servirá de entrada para o algoritmo novamente. Esse novo resultado do processo que obteve os valores de P1 e P2 como entrada será armazenado em P3 e P4, que servirão novamente de entrada, e assim sucessivamente até que todos os elementos do vetor P (subchaves) sejam preenchidos logo após todos os elementos das quatro S-boxes.

Este processo é repetido várias vezes, 9 para preencher o vetor P e 512 para preencher as quatro S-boxes.

Na Figura 3.2 é exemplificado o primeiro passo do processo com a cifragem da *string* formada por zeros e na Figura 3.3 se exemplificam as iterações posteriores onde as saídas da iteração i servem como entradas para a iteração i+1, sendo i ? 0.

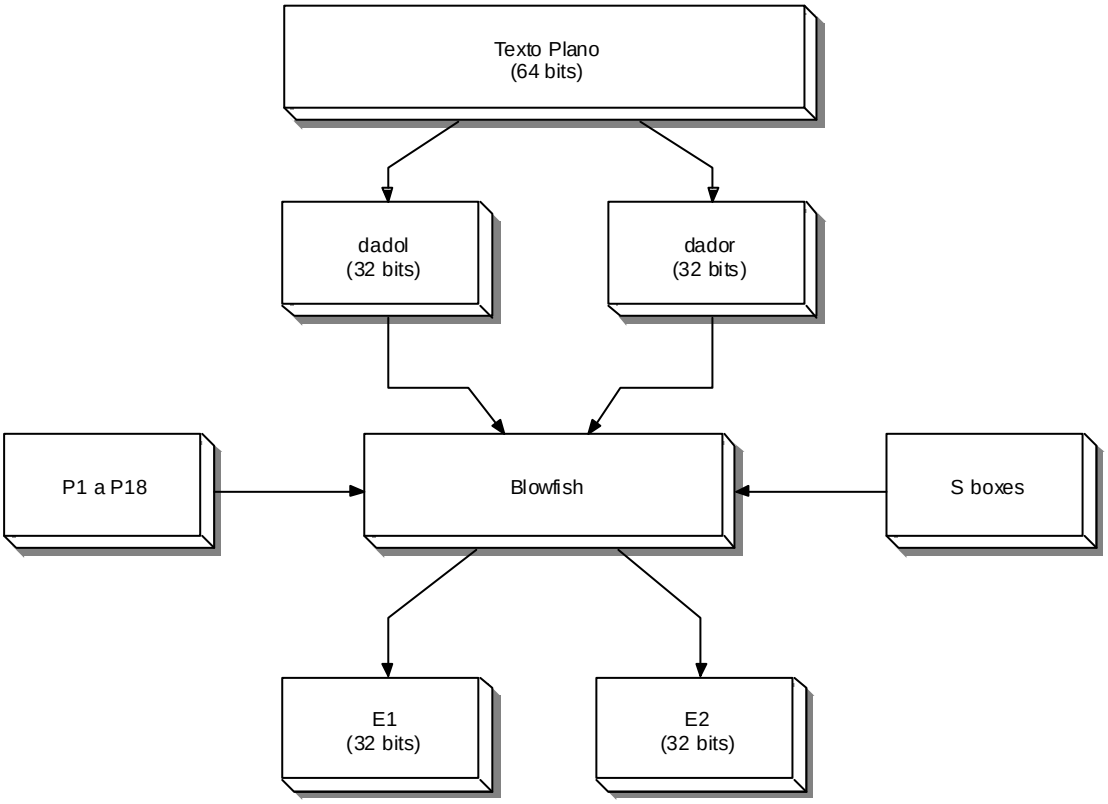

Figura 3.2 – Esquema gráfico para obtenção das subchaves intermediárias E1 e E2 que serão armazenadas em P1 e P2 (SCHNEIER, 1993)

Observa-se na Figura 3.2 que a *string* constituída de zeros antes de ser passada ao algoritmo Blowfish para ser cifrada, é dividida em dois blocos de 32 bits, portanto a *string* constituída de zeros deve possuir um tamanho de 64 bits, e assim, os 64 bits são tratados em dois blocos de 32 bits.

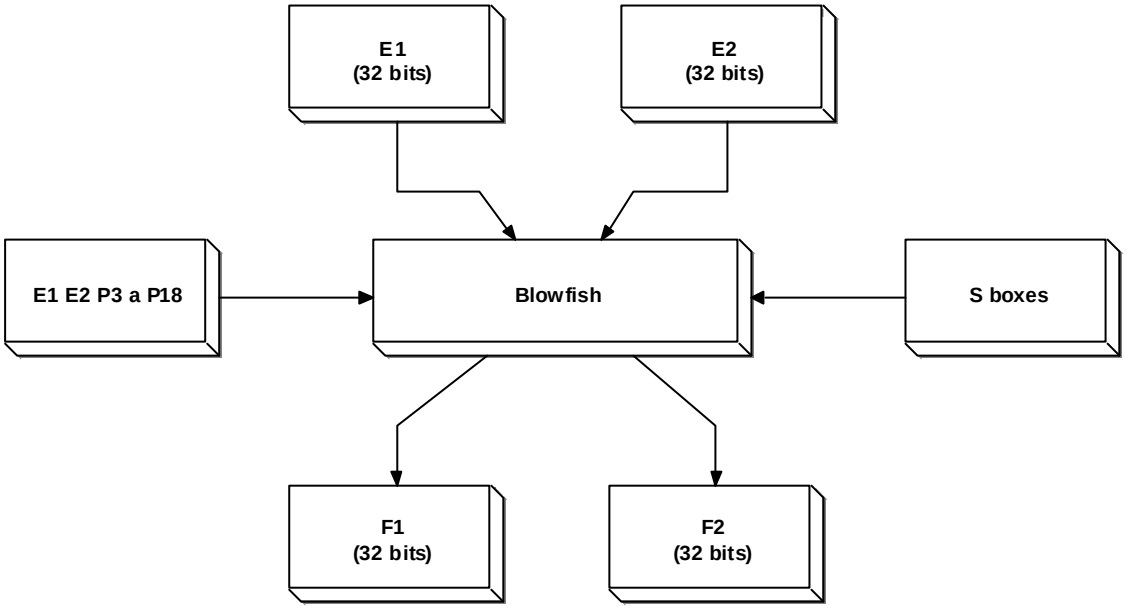

Figura 3.3 – Esquema gráfico para obtenção das subchaves intermediárias F1 e F2 que serão armazenadas em P3 e P4 (SCHNEIER, 1993)

3.3.2 Cifragem dos Dados

Como mencionado anteriormente, o Blowfish é um algoritmo de cifra de bloco simétrica de 64 bits, portanto a entrada para o processo de cifragem dos dados, constituídos de 64 bits, são divididos em dois blocos de 32 bits. Esses blocos são denominados, por questões de convenção, de xL, onde L significa *left*; e xR, onde R significa *rigth.* O bloco xL contém os 32 bits mais significativos do bloco de 64 bits e o bloco xR contém os 32 bits menos significativos. Como o algoritmo consiste de uma rede de Feistel, as operações serão

executadas 16 vezes. O algoritmo para a cifragem do bloco de 64 bits é descrito da seguinte maneira:

"Dividir X (bloco de 64 bits) em dois blocos de 32 bits: xL e xR

Para i ← 1 até 16 faça

xL ← xL XOR Pi

xR ← F(xL) XOR xR

Troca xL e xR

i ← i + 1

Fim para

xR ← xR XOR P17

xL ← xL XOR P18"

Observa-se, no algoritmo de cifragem, utilizando chave de 64 bits, que as operações abaixo relacionadas são executadas 16 vezes e em cada iteração xL e xR sofrem alterações (substituições).

"xL ← xL XOR Pi

xR ← F(xL) XOR xR

Troca xL e xR"

Após a divisão do bloco de 64 bits nos dois blocos de 32 bits xL e xR, inicia-se as rodadas na rede de Feistel, ou seja, as 16 iterações. A cada iteração, é atribuído ao bloco xL o resultado da operação XOR entre os seus 32 bits com os 32 bits do vetor P com índice "i". Já ao bloco xR é atribuído o resultado da operação XOR entre o retorno da função F, à qual é passado como parâmetro o bloco xL, os 32 bits do próprio bloco xR. Logo após essas

operações os valores dos blocos xL e xR são trocados entre si, ou seja, os 32 bits do bloco xL são atribuídos ao bloco xR e vice-versa.

Após a décima sexta iteração, é necessário trocar xL e xR mais uma vez e em seguida realizar as operações:

"xR ← xR XOR P17

xL ← xL XOR P18"

A Figura 3.4 demonstra uma iteração do processo de cifragem de um bloco de 64 bits. O texto cifrado será a união (concatenação) dos dois blocos (xL e xR), formando um bloco de 64 bits. Na última iteração do processo de cifragem, os blocos de 32 bits xL e xR não são trocados.

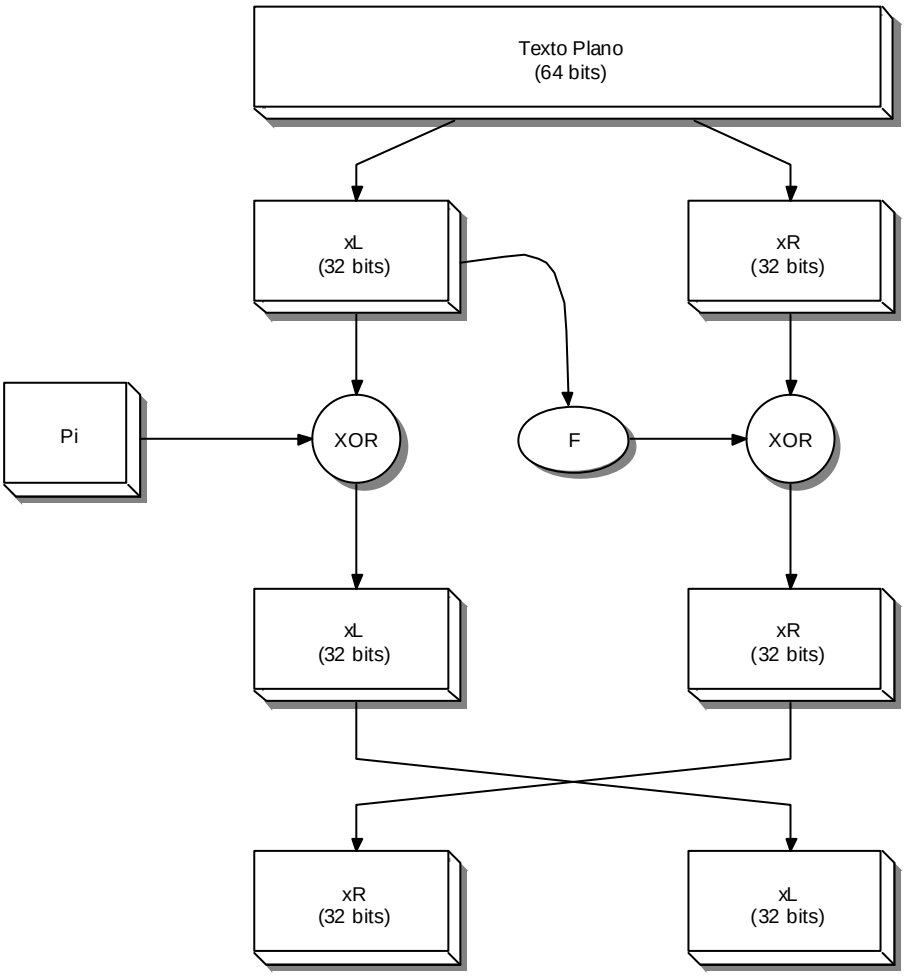

Figura 3.4 – Esquema gráfico do funcionamento de uma iteração (SCHNEIER, 1993)

O processo de decifragem é realizado da mesma forma que a cifragem, porém utilizando o vetor P em ordem inversa, ou seja, primeiramente P18, depois P17 e sucessivamente até atingir o elemento P1.

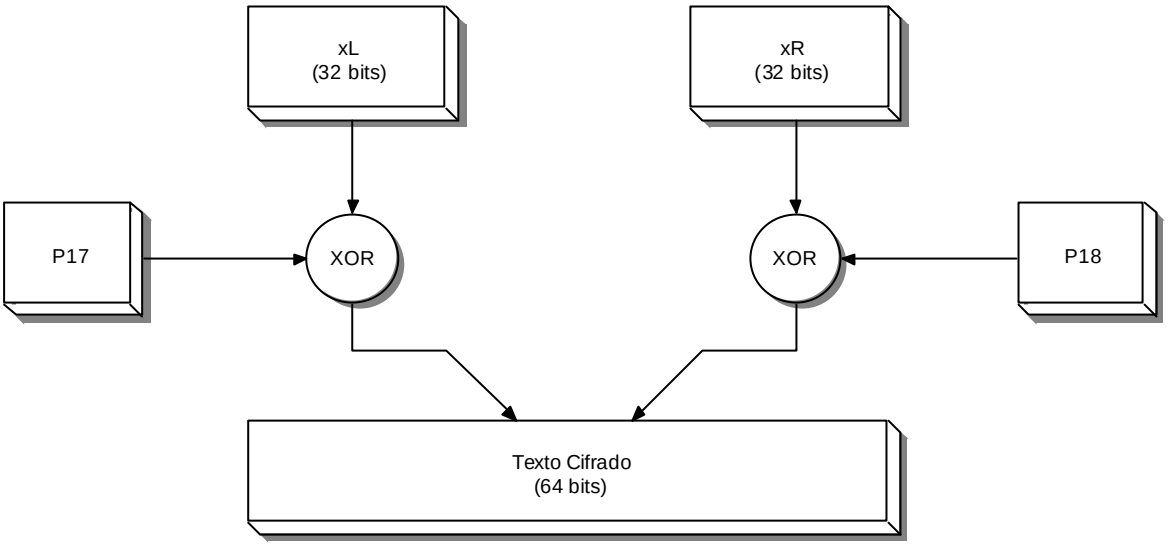

Figura 3.5 – Esquema gráfico do funcionamento da última iteração do processo de cifragem (SCHNEIER, 1993)

3.3.3 Função F

A função F como pode ser observado na Figura 3.4 é parte vital do processo de cifragem.

A função F recebe o bloco de 32 bits de xL, bits mais significativos do bloco de 64 bits para serem cifrados. A função F subdividi xL em quatro outros blocos de 8 bits. O primeiro bloco de 8 bits será utilizado como índice para a primeira S-box pois 2^8 é igual a 256 e cada S-box possui 256 valores, o segundo bloco de 8 bits será utilizado como índice para a segunda S-box e assim sucessivamente para as outras duas S-box restantes.

Em um primeiro passo, é realizada uma adição em módulo de 2^{32} entre os valores obtidos através da indexação da primeira e segunda S-box. Logo após, é realizada uma

operação de XOR entre o resultado dessa adição e o valor obtido através da indexação da terceira S-box, e no final é realizada mais uma adição em módulo de 2^{32} com o resultado da operação XOR executada anteriormente. Esse processo pode ser descrito pela seguinte expressão:

$$F(xL) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \text{ XOR } S_{3,c}) + S_{4,d} \bmod 2^{32}$$

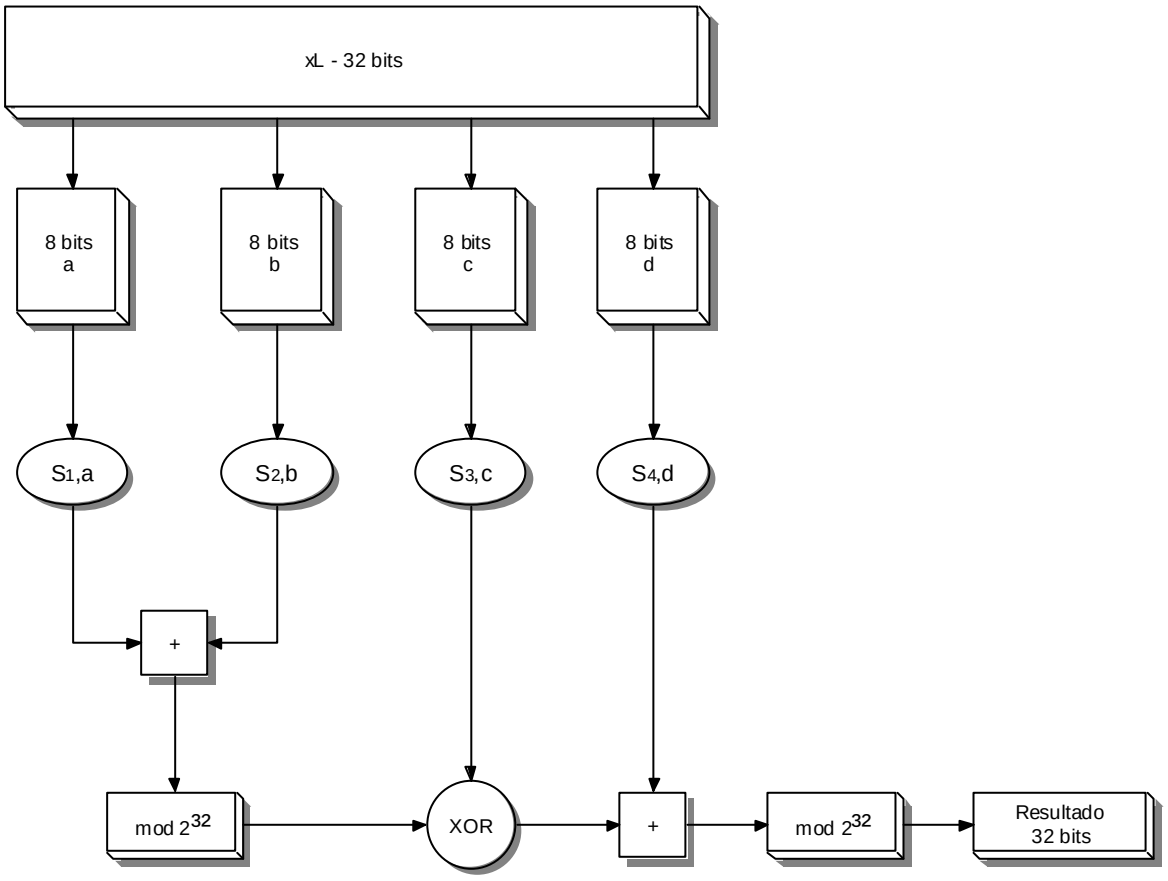

Figura 3.6 – Esquema gráfico de funcionamento da função F(xL) (SCHNEIER, 1993)

A Figura 3.6 mostra graficamente o funcionamento da função F, onde pode ser observado o fluxo de informações de forma clara.

3.4 Considerações Finais

Neste capítulo o algoritmo Blowfish foi estudado e apresentado de forma detalhada. O principal objetivo atingido foi o de obter o conhecimento e domínio detalhado, para usufruir dessas informações e obter eficiência nas implementações em linguagens de alto nível.

A estrutura lógica do algoritmo e as estruturas de dados mais adequadas para os parâmetros utilizados nos processos de cifragem de dados e decifragem de dados foram detalhadas para subsidiar as implementações nas linguagens C e Java.

4 IMPLEMENTAÇÃO DO ALGORITMO BLOWFISH EM C E JAVA

O algoritmo Blowfish foi implementado nas linguagens de alto nível C e Java. Nesta seção serão apresentados os principais aspectos e alguns trechos de código da implementação em linguagem C e linguagem Java desse algoritmo, bem como uma análise de seus respectivos desempenhos temporais.

4.1 Implementação em linguagem C

A linguagem C foi utilizada por ser estruturada, o que facilita a organização e leitura do código fonte de um programa, e também porque as instruções são executadas obedecendo a uma lógica pré-definida. A característica de portabilidade da linguagem também influiu na sua escolha, pois as instruções utilizadas para implementar o algoritmo Blowfish são executadas por processadores encontrados em PCs, estações de trabalho e outros.

A linguagem C é freqüentemente chamada de médio nível por combinar elementos de linguagens de alto nível com a funcionalidade da linguagem de baixo nível (*assembly*). Tratando-se de uma linguagem de médio nível, C permite a manipulação de bits, bytes e endereços, que são elementos básicos com os quais os computadores funcionam e de uso intenso por algoritmos criptográficos como o Blowfish.

Outro ponto importante para a escolha da linguagem C deve-se à portabilidade, pois permite adaptar o código escrito para diferentes processadores.

Segundo Sebesta (2002), a linguagem C possui instruções de controle adequadas, facilidades de estruturação de dados que permitem seu uso em muitas áreas de aplicação e também possui um rico conjunto de operadores que permitem um grau elevado de expressividade.

Em resumo, a linguagem C foi escolhida pelos seguintes fatores: ser uma linguagem estruturada, ser portável, eficiente, de baixo custo e bastante disseminada sendo uma boa opção na implementação de algoritmos criptográficos.

Quando comparada com uma linguagem *Assembly,* o código gerado pela linguagem C pode não ser tão eficiente, mais as características acima relacionadas colocam-na num patamar superior principalmente pela não estruturação, não portabilidade e difícil manutenção apresentada pela linguagem *Assembly*.

Para a implementação do algoritmo Blowfish foram definidas 4 funções:

- Inicia_Blowfish – responsável pela inicialização do algoritmo Blowfish, definindo os valores iniciais utilizados pelo algoritmo, tais como o vetor P e as quatro caixas S, realizando portanto a primeira etapa do algoritmo que trata da expansão de chave.
- Cifra_Blowfish – responsável pelo processo de cifragem dos dados.
- Decifra_Blowfish – responsável pelo processo de decifragem dos dados.
- F – realiza a indexação das caixas S utilizando como índices 8 bits obtidos a partir da divisão de um bloco de 32 bits, assim como operações de adição e XOR sobre os valores obtidos.

Na primeira função são criadas as estruturas de dados necessárias para a criação do vetor P e das quatro S-boxes, já inicializadas com seus respectivos valores.

Após a geração das estruturas básicas, o vetor P e as S-boxes, é possível implementar as funções primordiais do algoritmo Blowfish; cifragem e decifragem.

Os códigos implementados tanto para a cifragem quanto para decifragem são relativamente semelhantes, pois um processo é o inverso do outro como explicado no Capítulo 3.

O código em linguagem C da função que realiza o processo de cifragem é apresentado na Figura 4.1.

Cifra_Blowfish

```
void Cifra_Blowfish(BLOWFISH *ctx, unsigned long *xl, unsigned long *xr){
    unsigned long  Xl;
    unsigned long  Xr;
    unsigned long  temp;
    short i;

    Xl = *xl;
    Xr = *xr;

    for (i = 0; i < N; ++i) {
        Xl = Xl ^ ctx->P[i];
        Xr = F(ctx, Xl) ^ Xr;
        temp = Xl;
        Xl = Xr;
        Xr = temp;
    }

    temp = Xl;
    Xl = Xr;
    Xr = temp;

    Xr = Xr ^ ctx->P[N];
    Xl = Xl ^ ctx->P[N + 1];

    *xl = Xl;
    *xr = Xr;
}
```

Figura 4.1 – Código C Processo de Cifragem

No código apresentado o símbolo ^ representa uma operação XOR realizada bit a bit.

Essa implementação utiliza como parâmetros um ponteiro que permite acessar o vetor P e as S-boxes, e dois blocos de 32 bits que são cifrados (xL e xR).

Para realizar a criptografia, os blocos são cifrados através das 16 iterações baseadas na rede de Feistel.

Para decifrar é necessário executar o laço *for* de forma inversa (de trás para frente) invertendo assim os índices utilizados na indexação do vetor P. Na Figura 4.2 é apresentado o código da função de decifragem.

Decifra_Blowfish

```
void Decifra_Blowfish(BLOWFISH *ctx, unsigned long *xl, unsigned long *xr){
    unsigned long  Xl;
    unsigned long  Xr;
    unsigned long  temp;
    short i;

     Xl = *xl;
     Xr = *xr;

    for (i = N + 1; i > 1; --i) {
       Xl = Xl ^ ctx->P[i];
       Xr = F(ctx, Xl) ^ Xr;

       temp = Xl;
       Xl = Xr;
       Xr = temp;
    }

     temp = Xl;
     Xl = Xr;
     Xr = temp;

     Xr = Xr ^ ctx->P[1];
     Xl = Xl ^ ctx->P[0];

    *xl = Xl;
    *xr = Xr;
}
```

Figura 4.2 – Código C Processo de Decifragem

Após a implementação dos processos de cifragem e decifragem, foi implementada a função F, cujo código está apresentado na Figura 4.3.

Função F

```
static unsigned long F(BLOWFISH *ctx, unsigned long x) {
     unsigned short a, b, c, d;
     unsigned long  y;
     d = (unsigned short)(x & 0xFF); //mascara
      x >>= 8;
     c = (unsigned short)(x & 0xFF);
      x >>= 8;
     b = (unsigned short)(x & 0xFF);
      x >>= 8;
     a = (unsigned short)(x & 0xFF);
     y = ctx->S[0][a] + ctx->S[1][b];
     y = y ^ ctx->S[2][c];
     y = y + ctx->S[3][d];
     return y;
}
```

Figura 4.3 – Código C Função F

Finalmente, foi implementada, a função responsável pela inicialização do algoritmo Blowfish e a expansão de chave. Na Figura 4.4 é apresentada a função de inicialização.

Inicia_Blowfish

```
void Inicia_Blowfish(BLOWFISH *ctx, unsigned char *key, int keyLen) {
    int i, j, k;
    unsigned long data, datal, datar;

    //inicializa S-Boxes
    for (i = 0; i < 4; i++)
        for (j = 0; j < 256; j++)
            ctx->S[i][j] = ORIG_S[i][j];

    j = 0;

    /* realiza XOR do vetor P com os 32 primeiros bits da chave,
       XOR entre os 32 primeiros bits da chave com P0, os próximos
       32 bits da chave com P1, e assim por diante, até P17 */

    for (i = 0; i < N + 2; ++i) {
        data = 0x00000000;

        //obtém os 32 primeiros bits da chave
        for (k = 0; k < 4; ++k) {
            data = (data << 8) | key[j];
            j = j + 1;

            //se a chave chegou ao fim recomeçar com os primeiros bits da chave
            if (j >= keyLen)
                j = 0;
        }
        ctx->P[i] = ORIG_P[i] ^ data; //faz um XOR com P[i] e com 32i bits da chave
    }


    /* Cadeia de 64 bits formada por zeros que serão cifrados para dar inicio
    ao processo de obtenção das subchaves */
    datal = 0x00000000; //32 bits
    datar = 0x00000000; //32 bits

    for (i = 0; i < N + 2; i += 2) {
        Cifra_Blowfish(ctx, &datal, &datar);
        ctx->P[i] = datal;
        ctx->P[i + 1] = datar;
    }

    for (i = 0; i < 4; ++i) {
        for (j = 0; j < 256; j += 2) {
            Cifra_Blowfish(ctx, &datal, &datar);
            ctx->S[i][j] = datal;
            ctx->S[i][j + 1] = datar;
        }
    }
}
```

Figura 4.4 – Código C Inicialização do Algoritmo Blowfish

A implementação da função Inicia_Blowfish utiliza vários parâmetros e estruturas. Um ponteiro para a estrutura que manipula o vetor P e as S-boxes, um vetor do tipo caractere não sinalizado que contém a chave e o tamanho da mesma.

São inicializadas as estruturas que manipulam as S-boxes; As chaves intermediárias são obtidas a partir da chave passada como parâmetro, e é expandida através do processo de expansão de chave.

4.2 Análise de Desempenho do Algoritmo Implementado em Linguagem C

A análise de desempenho do algoritmo Blowfish implementado em linguagem C foi realizada através de testes para a obtenção do tempo que o algoritmo levou para cifrar e decifrar arquivos de diferentes tamanhos (100 kbytes, 500 kbytes, 1 Mbyte, 2 Mbytes e 5 Mbytes), com a utilização de chaves de tamanhos diferentes (32 bits, 64 bits, 128 bits, 192 bits e 256 bits). Após a realização de testes obteve-se uma média dos tempos de cifragem e decifragem para todos os arquivos e chaves utilizadas, sendo possível assim realizar a comparação de desempenho de cada uma das combinações testadas. Para a análise de coerência dos dados foi obtida também a variância bem como o desvio padrão em todos os testes realizados.

Os resultados obtidos nos testes foram agrupados e são apresentados na forma de tabelas ou de gráficos. São apresentadas cinco figuras, uma para cada tamanho de arquivo testado. Para cada arquivo foram utilizadas e testadas todas as chaves citadas. Além disso, para todos os diferentes arquivos, foram obtidos também os desvios padrões considerando os diversos comprimentos de chaves nos processos de cifragem e decifragem.

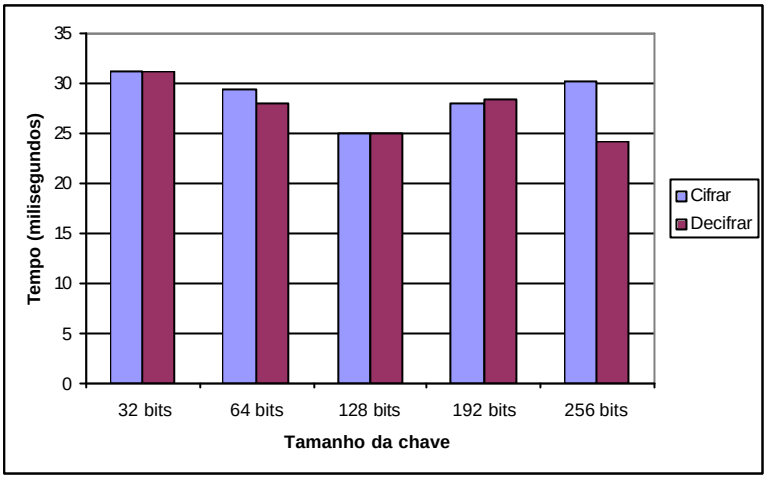

Figura 4.5 – Cifragem e Decifragem de um arquivo de 100 kbytes

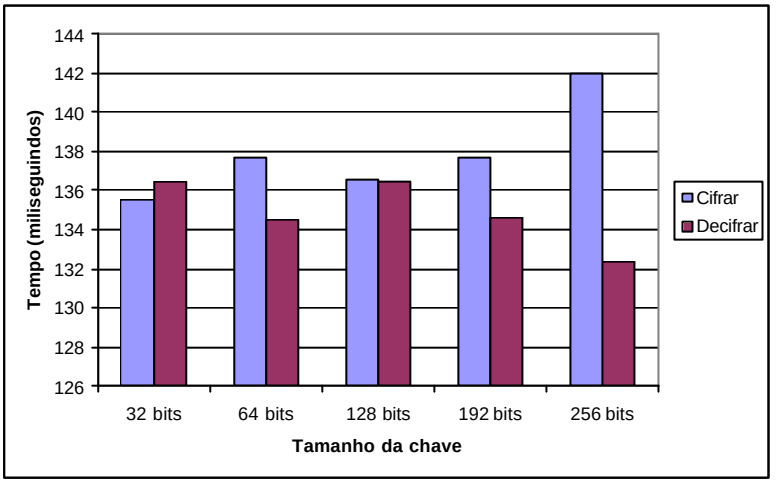

Figura 4.6 – Cifragem e Decifragem de um arquivo de 500 kbytes

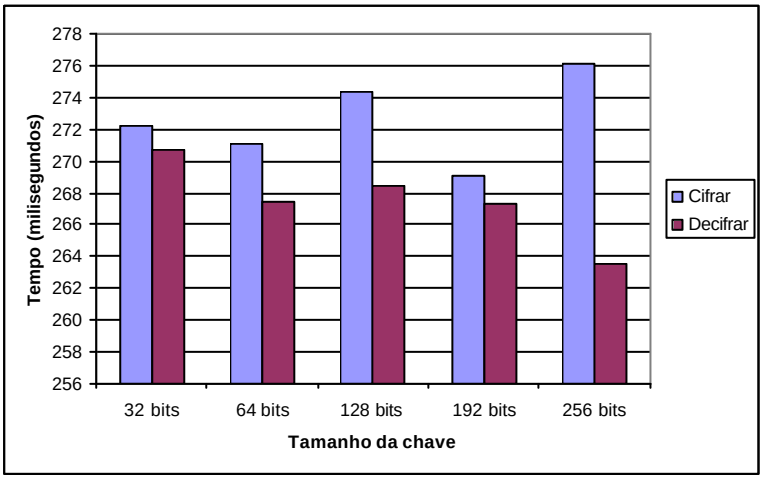

Figura 4.7 – Cifragem e Decifragem de um arquivo de 1 Mbyte

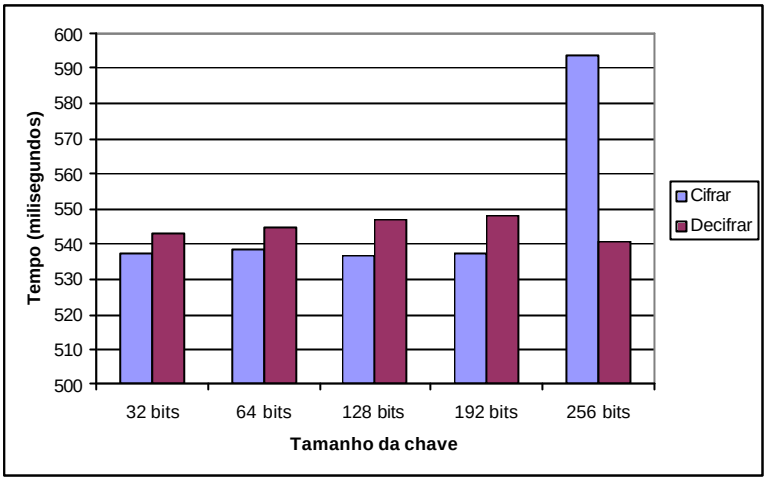

Figura 4.8 – Cifragem e Decifragem de um arquivo de 2 Mbytes

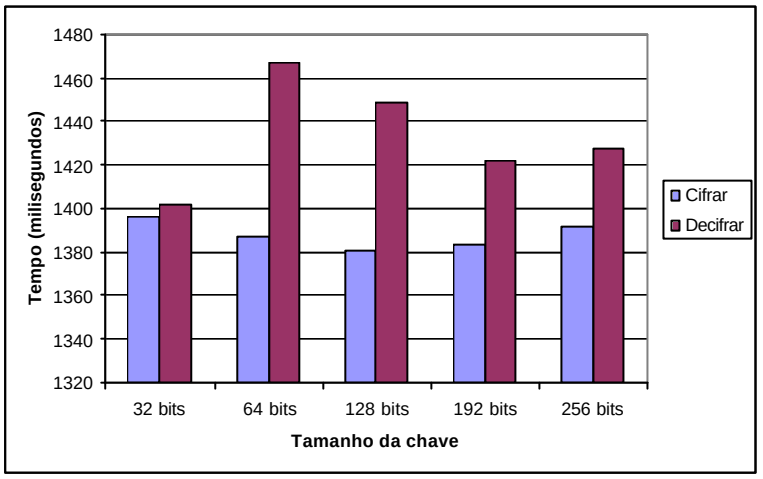

Figura 4.9 – Cifragem e Decifragem de um arquivo de 5 Mbytes

Os resultados apresentados nas figuras 4.5, 4.6, 4.7, 4.8 e 4.9 foram realizados em um microcomputador Pentium IV de 1.6GHz com 128Mbytes de memória RAM e plataforma Windows 2000.

Como se observa nos gráficos das Figuras 4.5 a 4.9 a chave de 128 bits apresentou um desempenho bom e em alguns casos como no caso da Figura 4.5 (arquivo de 100 Kbytes) melhor que as outras chaves, tanto no processo de cifragem quanto decifragem, fato esse que não era esperado. Além disso, a chave de 128 bits, também mostra melhor desempenho no processo de cifragem para arquivos com dimensões de 5 Mbytes e 2 Mbytes. Uma das possíveis explicações para o ocorrido é devido ao algoritmo Blowfish ter se baseado no

padrão de chave com tamanho de 128 bits, que foi requisitado para os participantes do projeto AES (*Advanced Encryption Standard*), no qual também foi utilizado o algoritmo Blowfish, sendo assim, seu desenvolvimento foi voltado para uma melhor performance com uma chave de 128 bits, mesmo aceitando chaves de tamanhos variáveis de 32 bits a 448 bits.

Outro fato relevante em relação aos gráficos refere-se à escala em que eles estão, pois pode parecer que há grande diferença de temporização entre as execuções do algoritmo com chaves de tamanhos diferentes, porém praticamente a variação existente é mínima. Fato que pode ser consolidado através dos desvios padrões obtidos.

Quanto ao desempenho da chave de 256 bits, o resultado esperado era o pior desempenho para todos as dimensões de arquivos. No entanto, no processo de cifragem para arquivos com dimensões de 5 Mbytes e 100 Kbytes o desempenho dessa chave não foi o pior, sendo que, no processo de decifragem a chave de 256 bits apresentou o melhor desempenho para arquivos com dimensões de 2 Mbytes, 1 Mbytes, 500 Kbytes e 100 Kbytes.

Observa-se também que a chave de 32 bits não mostra o melhor desempenho para todos casos, excetuando-se o processo de cifragem para arquivo de 2 Mbytes e o processo de decifragem para arquivos de 5 Mbytes.

As chaves de 64 bits e 192 bits não apresentam o melhor desempenho em nenhum dos dois processos (cifragem e decifragem) para qualquer dimensão de arquivo.

A seguir são compiladas algumas considerações sobre o desempenho obtido entre tamanhos de chave e arquivos de dimensões constantes. Lembrando que foram utilizadas chaves de 32, 64, 128, 192 e 256 bits. São calculadas as variações de desempenho e essas variações são representadas em valores percentuais, para chaves de tamanhos adjacentes.

Exemplificando, para o arquivo de 100 Kbytes obteve-se para chave de 32 bits o tempo de médio de cifragem de 31,2 milisegundos com desvio padrão de 0,6532 e o tempo médio de decifragem de 31,066 milisegundos com desvio padrão de 0,5735. Para a chave de

64 bits o tempo médio de cifragem de 29,4666 milisegundos com desvio padrão de 7,7534 e o tempo médio de decifragem de 28,0666 milisegundos com desvio padrão de 6,0384. Os tempos médios obtidos estão muito próximos, determinando que no processo de cifragem a chave de 64 bits mostra-se 5,8847 % mais eficiente que a chave de 32 bits e no processo de decifragem mostra-se 10,688 % mais eficiente.

De forma análoga foram calculadas todas as variações de eficiência para as chaves com quantidade de bits adjacentes, 64 bits para 128 bits, 128 bits para 192bits e de 192 bits para 256 bits.

As tabelas que seguem mostram as variações de eficiência para os processos de cifrar de decifrar, comparando comprimento de chaves adjacentes, para o algoritmo implementado em linguagem C.

A tabela 4.1.a mostra o tempo médio de processamento (milisegundos) e a variação de eficiência (%) do processo de cifragem para arquivos de diversas dimensões (100Kbytes, 500 Kbytes, 1 Mbyte, 2 Mbytes e 5 Mbytes) e diferentes tamanhos de chaves (32, 64, 128, 192 e 256 bits).

Tabela 4.1.a – Variação de eficiência na cifragem para o algoritmo em linguagem C

Chave (bits)	**Arquivo cifragem C**									
	100 kbytes		**500 kbytes**		**1 Mbyte**		**2 Mbytes**		**5 Mbytes**	
	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)
32	31,200		135,533		272,133		537,333		1396,133	
64	29,466	5,8847	137,733	-1,5972	271,000	0,4182	538,466	-0,2104	1387,533	0,6198
128	24,933	18,1818	136,600	0,8296	274,333	-1,2150	536,533	0,3603	1381,200	0,4585
192	24,933	-11,1638	137,733	-0,8228	269,133	1,9321	537,333	-0,1488	1383,266	-0,1494
256	30,333	-7,47252	142,000	-3,0046	276,200	-2,5585	594,000	-9,5398	1391,600	-0,5988

De forma análoga, a tabela 4.1.b mostra tempo médio de processamento (milisegundos) e a variação de eficiência do processo de decifragem para arquivos com dimensões (100 Kbytes, 500 Kbytes, 1 Mbyte, 2 Mbytes e 5 Mbytes) e diferentes tamanhos de chaves (32, 64, 128, 192 e 256 bits).

Tabela 4.1.b – Variação de eficiência na decifragem para o algoritmo em linguagem C

Chave (bits)	Dimensão de Arquivo									
	100 kbytes		500 kbytes		1 Mbyte		2 Mbytes		5 Mbytes	
	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)
32	31,066		136,400		270,666		543,266		1401,933	
64	28,066	10,6881	134,466	-1,5972	267,466	1,1964	544,800	-0,2814	1466,666	-4,4136
128	25,066	11,9680	136,466	-1,4655	268,400	-0,3477	546,933	-0,3900	1449,066	1,2145
192	28,533	-12,1495	134,600	1,3868	267,333	0,3990	548,066	-0,2067	1421,866	1,9129
256	24,200	17,9063	132,333	1,7128	263,533	1,4419	540,600	1,3811	1428,200	-0,4434

Para a obtenção dos tempos médios, foram realizados 20 testes para todos os arquivos e chaves utilizadas.

Na figura 4.10 é possível observar que à medida que o tamanho do arquivo a ser cifrado ou decifrado aumenta, o tempo de cifragem e decifragem também aumenta. Isso acontece para todos os tamanhos de chaves utilizados. Para os testes realizados tanto nas implementações nas linguagens C e Java (ver seção 4.3), os arquivos utilizados para fins de obtenção dos resultados foram do tipo doc, já que o tipo de arquivo utilizado para cifragem ou decifragem não influi no desempenho temporal, pois o algoritmo realiza as operações criptográficas em baixo nível manipulando os bits.

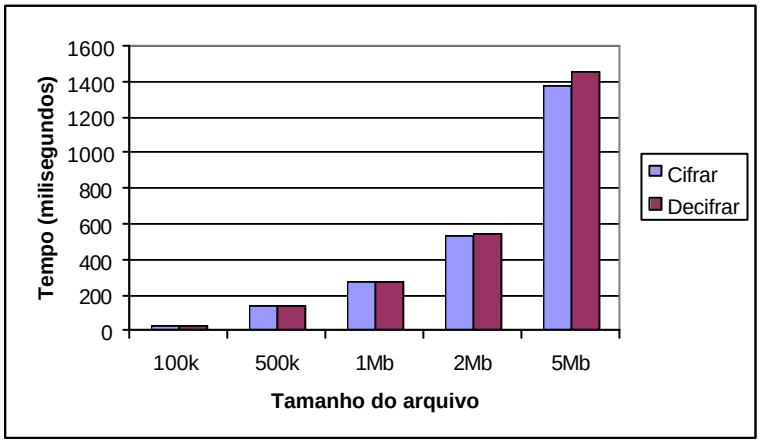

Figura 4.10 – Variação do tempo de cifragem e decifragem para arquivos de tamanhos diferentes utilizando uma chave 128 bits

4.3 Implementação em linguagem Java

A linguagem Java, assim como a linguagem C, permite a manipulação e operações sobre bits e bytes tornando seu uso adequado para o desenvolvimento de algoritmos criptográficos.

Java, segundo Deitel; Deitel (2001), é uma linguagem completamente orientada a objetos com forte suporte para técnicas adequadas de engenharia de software.

Um fator importante para a escolha da linguagem Java foi sua característica de portabilidade e por ser uma linguagem que vem sendo cada vez mais disseminada, tanto no meio acadêmico quanto nas aplicações comerciais de pequeno e grande porte.

Outro fator referente à linguagem Java é o de permitir implementar aplicativos baseados em Intranet e Internet e qualquer outro software para dispositivos que se comuniquem em uma rede, fato que só vem consolidar a motivação e importância da implementação do algoritmo criptográfico em Java, obtendo assim facilidade de implementações de aplicativos que se comunicam em rede juntamente com a segurança.

A linguagem Java possui uma biblioteca em tempo de execução que tem por objetivo proporcionar independência de plataforma, sendo esse um dos aspectos que gera a portabilidade citada anteriormente.

Outra vantagem da programação em Java pode ser atribuída à similaridade de sintaxe com a linguagem C++, porém os projetistas da linguagem Java a deixaram simplificada em relação a C++ acrescentando recursos que eliminam a possibilidade de gerar código com os tipos mais comuns de erros.

Java é uma linguagem interpretada, quando um programa Java é compilado é gerado um arquivo de objeto neutro em relação à arquitetura, o código gerado pode ser executado em vários processadores, devido a existência da máquina virtual Java (JVM – *Java Virtual*

Machine) que interpreta o código gerado. Essa independência de arquitetura é propiciada pelo compilador Java que gera instruções em *bytecodes* que não possuem ligação com um tipo de arquitetura em particular.

Outro aspecto importante da linguagem Java diz respeito ao tamanho dos tipos de dados, por exemplo, um dado do tipo *int* em Java é sempre um inteiro de 32 bits já nas linguagens C ou C++ isso pode variar dependendo do desenvolvedor do compilador. Através desse tamanho fixo dos tipos numéricos, obtém-se maior portabilidade. Os dados binários são armazenados em um formato fixo, eliminando prováveis confusões entre as representações "*big endian / little endian*" (HORSTMANN; CORNELL, 2001).

Um ponto negativo da linguagem Java em relação à linguagem C é que embora o desempenho dos *bytecodes* interpretados seja na maioria das vezes suficiente para a execução de certas aplicações, existem situações em que é necessário um desempenho superior. Porém existe uma forma de compilação do código em linguagem Java, que utiliza compiladores JIT (*Just-In-Time*). Tais compiladores exercem a função de compilar os *bytecodes* em código nativo de máquina, uma vez armazenado, esse código nativo é utilizado quando necessário maior desempenho, dessa forma uma compilador JIT pode monitorar um trecho de código que é mais freqüentemente executado e otimizar esse código melhorando o desempenho geral com relação à velocidade.

Para a implementação do algoritmo Blowfish em Java foram definidas 3 classes:

- Blowfish – possui a implementação propriamente dita do algoritmo Blowfish;
- BlowStrut – refere-se às estruturas básicas necessárias para o funcionamento do algoritmo tais como as S-boxes e o vetor P;
- NewJFrame – possui a implementação da interface gráfica desenvolvida para facilitar a utilização do algoritmo Blowfish e as questões de temporização.

Inicialmente foi pensado na classe BlowStrut e como desenvolvê-la pois é parte fundamental do algoritmo conter as estruturas que o embasam. Sendo a linguagem Java orientada a objetos, um paradigma diferente da linguagem C, que é uma linguagem procedural, foram criados como uma classe (BlowStrut) interna à classe Blowfish, diferenciando da estrutura de dados utilizada na linguagem C para manipular as S-boxes e o vetor P. O código na Figura 4.11 mostra a criação da classe BlowStrut.

BlowStrut

```
public class Blowfish {

    class BlowStrut {
       int S[ ][ ] = new int[4][256];
       int P[ ] = new int[18];
    }
    .....
    .....
```

Figura 4.11 – Trecho de Código Java (Classe interna BlowStrut)

Após a criação da classe BlowStrut foi realizado o desenvolvimento da classe Blowfish que realiza os processos de expansão de chave, cifragem e decifragem dos dados.

A classe Blowfish possui as S-boxes e o vetor P com os valores hexadecimais do número PI, como citado no capitulo 3, bem como outros métodos que serão descritos.

Os principais métodos pertencentes à classe Blowfish, além do método construtor, são os seguintes:

- cifrar – responsável pelo processo de cifragem dos dados;
- decifrar – responsável pelo processo de decifragem dos dados;
- b2d – responsável pelo tratamento do bloco de 64 bits;
- d2b – responsável pelo tratamento do bloco de 64 bits;
- F – realiza a função f, seguindo processo idêntico ao da implementação em C.

O método construtor da classe Blowfish realiza o processo de expansão de chave do algoritmo recebendo como argumento a chave utilizada para a criptografia, que passará pelo processo de expansão, como realizado na implementação em C. O código na Figura 4.12 a seguir mostra o construtor da classe.

Construtor da classe Blowfish

```
public Blowfish ( byte[ ] key ) {

  this.ctx = new BlowStrut();
  System.arraycopy(ps, 0, this.ctx.P, 0, 18);
  System.arraycopy(ks0, 0, this.ctx.S[0], 0, 256);
  System.arraycopy(ks1, 0, this.ctx.S[1], 0, 256);
  System.arraycopy(ks2, 0, this.ctx.S[2], 0, 256);
  System.arraycopy(ks3, 0, this.ctx.S[3], 0, 256);

  int data;
  int j = 0, i;


  /* Processo de operação Xor entre 32 bits da chave
     e 32 bits de determinada posição do vetor P */

  for (i = 0; i < 18; ++i) {
    data = 0x00000000;
    for (int k = 0; k < 4; ++k) {
       data = (data << 8) | (key[j] & 0xFF);
       j++;
       if (j >= key.length)
          j = 0;
    }
     this.ctx.P[i] ^= data;
  }
  byte[ ] b = new byte[8];

  /* Processo de cifragem da string de 64 bits */

  for (i = 0; i < 18; i += 2) {
    cifrar(b);
    this.ctx.P[i] = b2d(b, 0);
     this.ctx.P[i + 1] = b2d(b, 4);
  }

  for (i = 0; i < 4; ++i) {
     for (j = 0; j < 256; j += 2) {
       decifrar(b);
       ctx.S[i][j] = b2d(b, 0);
       ctx.S[i][j + 1] = b2d(b, 4);
     }
  }
} //fim do método construtor
```

Figura 4.12 – Código Java Construtor da Classe Blowfish

Os métodos de cifragem e decifragem recebem como argumento um vetor do tipo *byte* que refere-se ao bloco de dados que será cifrado ou decifrado. A seguir na Figura 4.13 é apresentado o código do método cifrar que realiza a cifragem de um bloco de dados.

Método cifrar

```
public void cifrar(byte[ ] data) {
   int blocks = data.length >> 3;
   for (int k = 0, p; k < blocks; k++) {
     p = k << 3;
     int Xl = b2d(data, p);
     int Xr = b2d(data, p + 4);
     int tmp;
     for (int i = 0; i < 16; i++) {
       Xl = Xl ^ this.ctx.P[i];
       Xr = F(Xl) ^ Xr;
       tmp = Xl;
       Xl = Xr;
       Xr = tmp;
     }
     tmp = Xl;
     Xl = Xr;
     Xr = tmp;
     Xr ^= this.ctx.P[16];
     Xl ^= this.ctx.P[17];
     d2b(Xl, data, p);
     d2b(Xr, data, p + 4);
   }
}
```

Figura 4.13 – Código Java Processo de Cifragem

Além dos métodos de cifragem e decifragem foram criados dois outros métodos adicionais, um que realiza a aglutinação de 4 posições de um vetor do tipo *byte*, ou seja, 32 bits a uma variável do tipo *int* e outro que realiza o processo inverso, desaglutinando os dados de uma variável do tipo *int* de 8 em 8 bits até preencher quatro posições de um vetor do tipo *byte.* Os dois métodos foram denominados respectivamente *b2d* e *d2b*. Esses métodos foram criados, pois na implementação na linguagem Java, os métodos de cifragem e decifragem recebem o bloco de dados de tamanho de 64 bits, ou seja, diferente da implementação na linguagem C, onde as funções de cifragem e decifragem recebem o bloco de 64 bits já dividido em dois blocos de 32 bits (xL e xR). Portanto, como o algoritmo Blowfish recebe blocos de 64 bits e os trata em grupos de 32 bits foi necessária à implementação dos dois

métodos especificados. A diferença na implementação em C e Java ocorreu devido às características peculiares a cada linguagem. Na linguagem C foi utilizada a função *sizeof* para realizar a leitura de dois blocos de 32 bits (xL e xR) separadamente. Já na linguagem Java foi realizada a leitura de um bloco de 64 bits, dessa forma houve a necessidade de separar esse bloco em dois de 32 bits.

A Figura 4.14 ilustra os códigos dos métodos b2d e d2b:

Método b2d

```
private int b2d(byte[ ] b, int p) {
   int r = 0;
   r |= b[p + 3] & 0xFF;
   r <<= 8;
   r |= b[p + 2] & 0xFF;
   r <<= 8;
   r |= b[p + 1] & 0xFF;
   r <<= 8;
   r |= b[p] & 0xFF;
   return r;
}
```

Método d2b

```
private void d2b(int a, byte[ ] b, int p) {
    b[p] = (byte) (a & 0xFF);
    a >>= 8;
    b[p + 1] = (byte) (a & 0xFF);
    a >>= 8;
    b[p + 2] = (byte) (a & 0xFF);
    a >>= 8;
    b[p + 3] = (byte) (a & 0xFF);
}
```

Figura 4.14 – Código Java Métodos b2d e d2b

O método b2d a partir do vetor de *bytes* recebido, executa operações de deslocamento de bits à esquerda juntamente com uma operação de AND bit-a-bit, utilizando o valor hexadecimal 0xFF (11111111 em binário) como máscara para obter o conteúdo da posição especifica do vetor de *bytes*. Logo após é realizada uma operação de OR que é responsável por aglutinar os dados na variável do tipo *int*. Já o método d2b realiza de forma parecida as operações de deslocamento de bits (à direita) e AND com a máscara, só que agora obtendo o processo inverso.

Método F

```
private int F(int x) {
   int a, b, c, d;
   d = x & 0xFF;
   x >>= 8;
   c = x & 0xFF;
   x >>= 8;

   b = x & 0xFF;
```

```
    x >>= 8;
    a = x & 0xFF;
    int y = this.ctx.S[0][a] + this.ctx.S[1][b];
    y ^= this.ctx.S[2][c];
    y += this.ctx.S[3][d];
    return y;
  }
```

Figura 4.15 – Código Java Método F

Na Figura 4.15 é ilustrado o método F implementado na linguagem Java, o qual não difere da implementação em linguagem C, a única mudança foi com relação aos tipos das variáveis, sendo o tipo inteiro da linguagem Java.

Sendo implementada a classe Blowfish e seus métodos, foi inicializada a implementação da classe *NewJFrame* que possui uma interface gráfica com o usuário, onde este escolhe o arquivo que deseja cifrar. Nessa classe foram implementados tanto os métodos para realizar os eventos de interação com usuário quanto os de leitura e escrita de arquivos. As figuras que seguem (Figura 4.16, Figura 4.17, Figura 4.18 e Figura 4.19) mostram as telas da interface desenvolvida e os passos necessários para realizar a cifragem e decifragem de um arquivo especifico.

A figura 4.16 refere-se à interface principal da aplicação desenvolvida. Na figura 4.17 o usuário pode escolher através da instanciação de um objeto da classe *JFileChooser* o arquivo que deseja cifrar o conteúdo. Após a escolha do arquivo desejado, como pode ser observado na figura 4.18, o usuário entra com a chave que deseja utilizar e para realizar o processo de cifragem do arquivo escolhido é necessário apenas clicar no botão “Cifrar” e o arquivo será cifrado. Após o ciframento completo do arquivo, o tempo de cifragem será mostrado em milisegundos em um campo na parte inferior da interface. Na figura 4.19 é demonstrado o processo de decifragem do arquivo escolhido, o qual é realizado da mesma maneira que a cifragem e no final é retornado o tempo de duração da decifragem.

Em Java, o tempo foi calculado através do método *static currentTimeMillis* da classe *System* que retorna o tempo corrente em milisegundos. Já na implementação na linguagem C

foi usada a função *clock()* da biblioteca C padrão através da inclusão do arquivo de cabeçalho *time.h*.

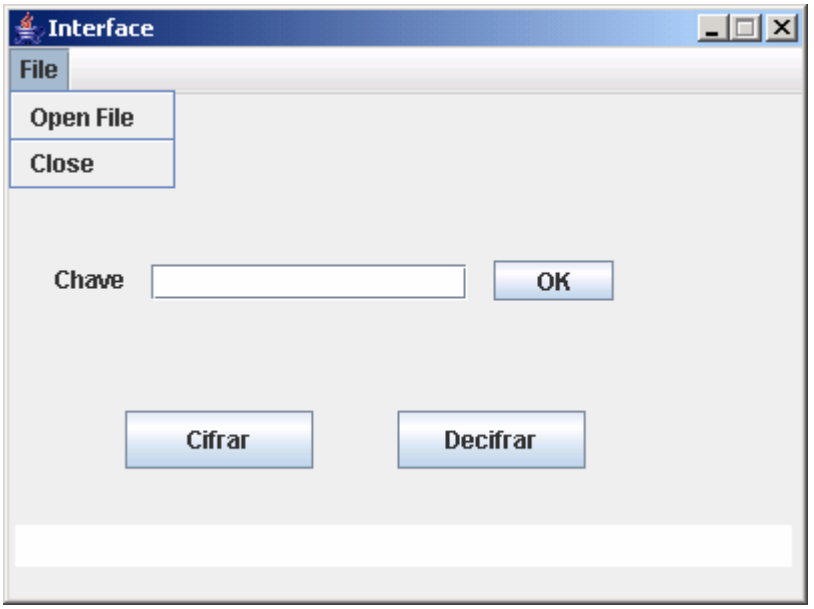

Figura 4.16 – Tela Principal da Interface

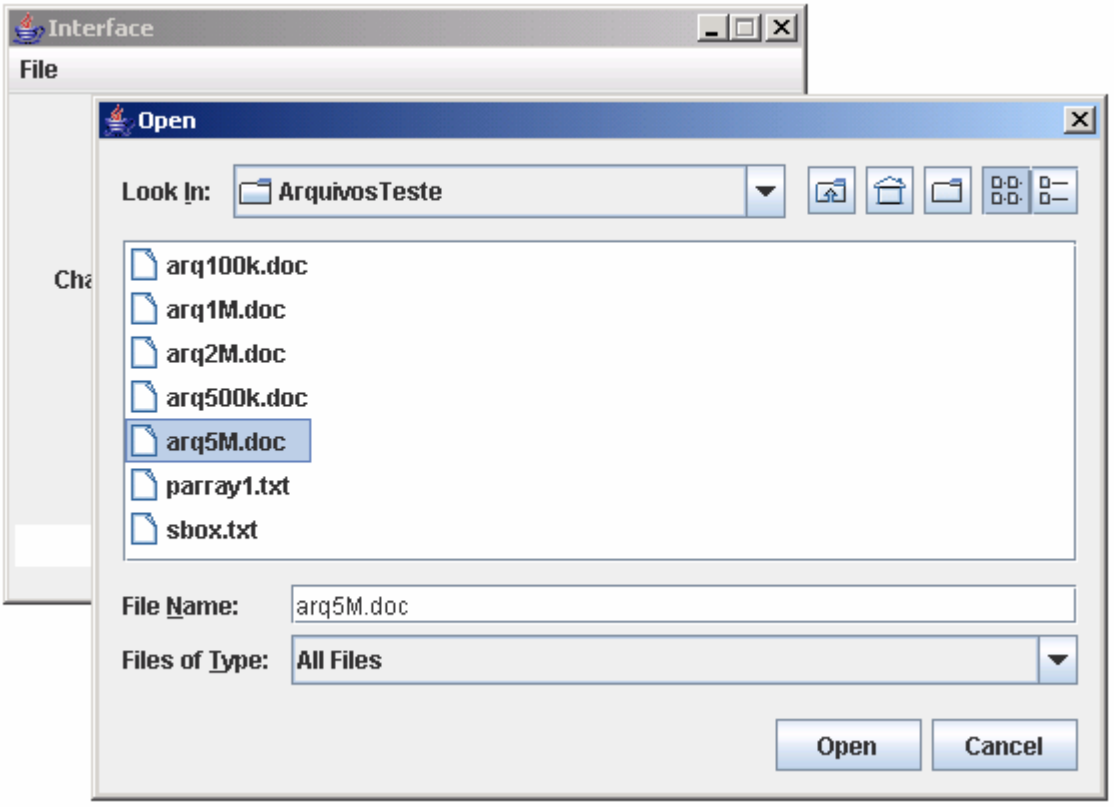

Figura 4.17 – Escolha do Arquivo a ser Cifrado

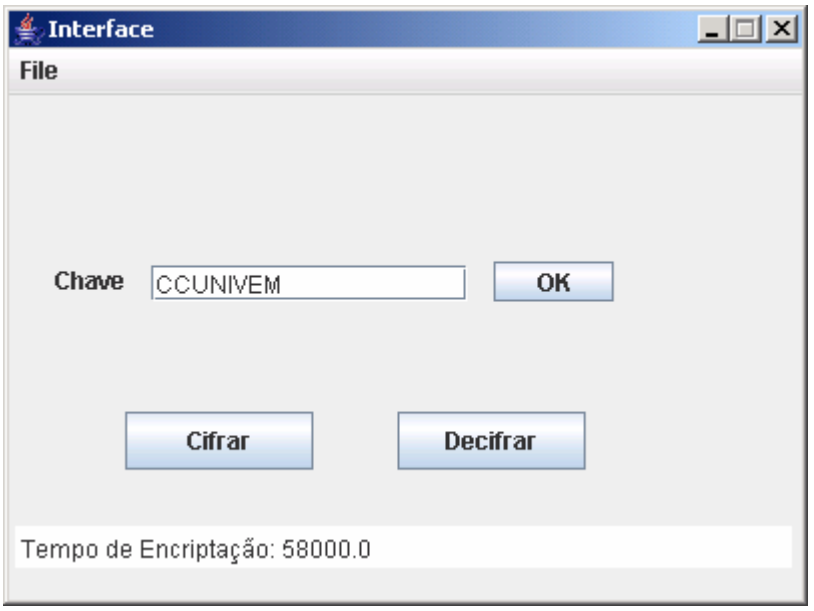

Figura 4.18 – Arquivo cifrado com a chave especificada pelo usuário

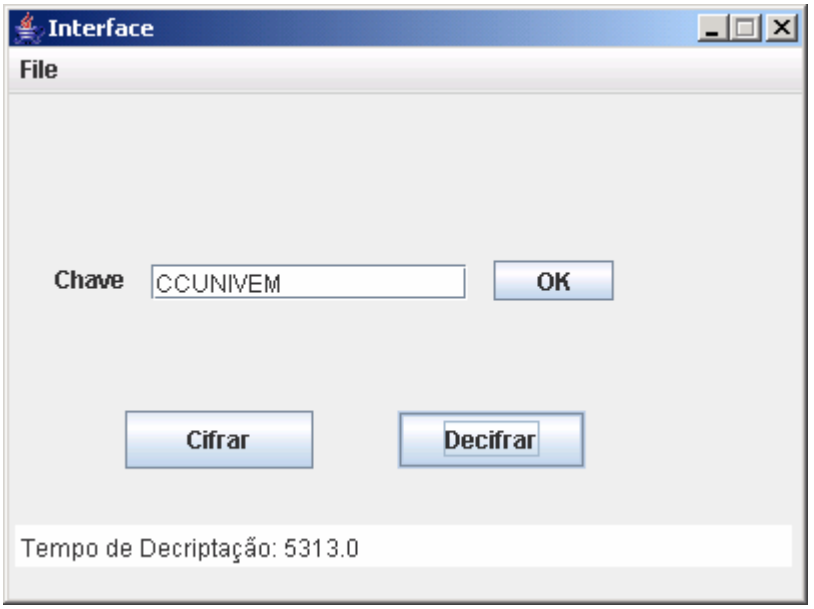

Figura 4.19 – Arquivo Decifrado com a chave especificada pelo usuário

As *strings* na linguagem Java são salvas em um formato *Unicode*, padrão internacionalmente reconhecido para o formato de caractere, nesse formato cada caractere possui o tamanho de 16 bits. Portanto a chave CCUNIVEM (8 caracteres), utilizada nas Figuras 4.16 e 4.19 para realizar respectivamente os processos de cifragem e decifragem, possui o tamanho de 128 bits.

4.4 Análise de Desempenho do Algoritmo Implementado em Linguagem Java

A análise de desempenho do algoritmo Blowfish implementado em linguagem Java foi realizada de forma similar à análise de desempenho efetuada para a implementação do algoritmo em linguagem C. Essa similaridade foi necessária para propiciar comparações de eficiência entre as duas implementações realizadas usando as linguagens utilizadas. Assim, através de testes foram obtidos os tempos que o algoritmo gastou para cifrar e decifrar arquivos de diferentes tamanhos (100 kbytes, 500 kbytes, 1 Mbyte, 2 Mbytes e 5 Mbytes), com a utilização de chaves de tamanhos diferentes (32 bits, 64 bits, 128 bits, 192 bits e 256 bits). Após a realização dos testes obteve-se uma média dos tempos de cifragem e decifragem para todos os arquivos e chaves utilizadas, sendo possível assim realizar a comparação de desempenho de cada uma das combinações testadas. Para a análise de coerência dos dados foi obtida também a variância bem como o desvio padrão em todos os testes realizados.

Para a obtenção da média dos tempos, foram realizados 20 testes para todos os arquivos e chaves utilizadas, assim como na implementação do algoritmo na linguagem C.

Os resultados obtidos nos testes foram agrupados e estão apresentados na forma de figuras e/ou tabelas. São apresentadas cinco figuras, uma para cada tamanho de arquivo testado. Para cada arquivo foram utilizadas e testadas todas as chaves citadas. Além disso, para todos os diferentes arquivos, foram obtidos também os desvios padrões considerando os diversos comprimentos de chaves.

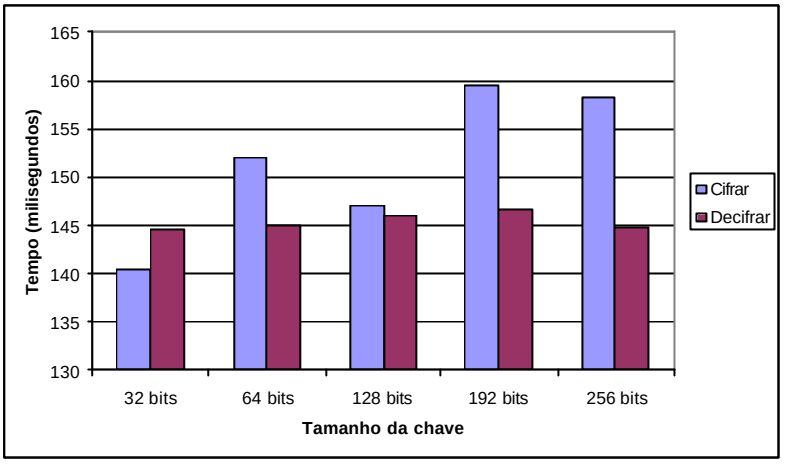

Figura 4.20 – Cifragem e Decifragem de um arquivo de 100 kbytes

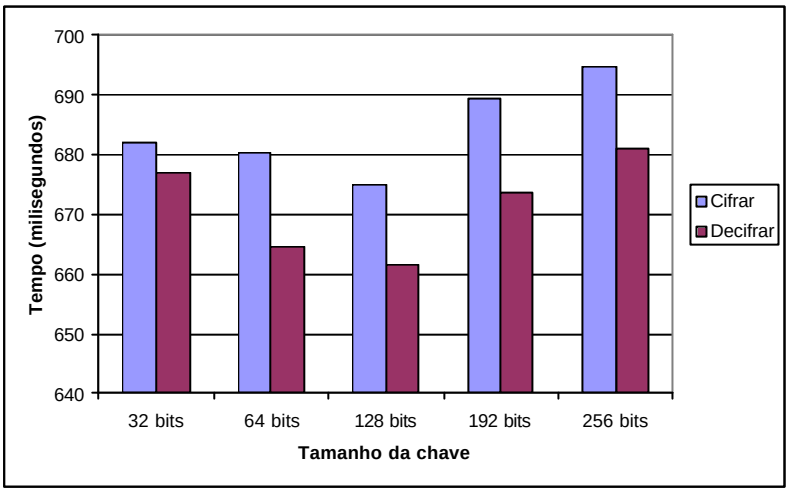

Figura 4.21 – Cifragem e Decifragem de um arquivo de 500 kbytes

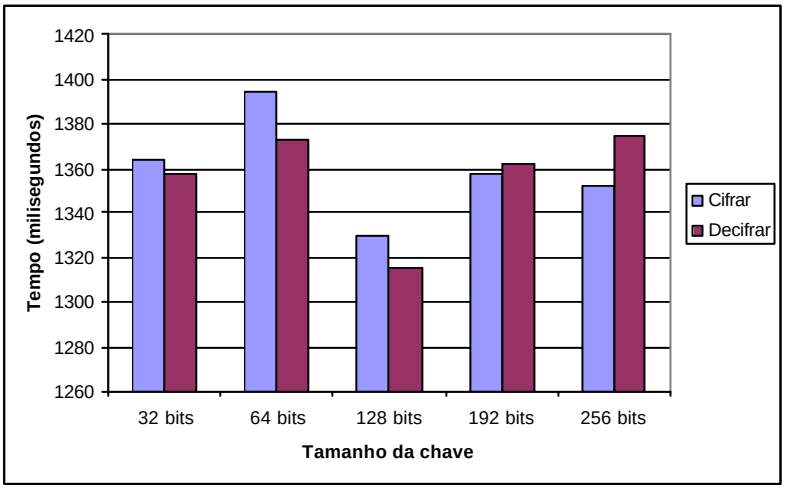

Figura 4.22 – Cifragem e Decifragem de um arquivo de 1 Mbyte

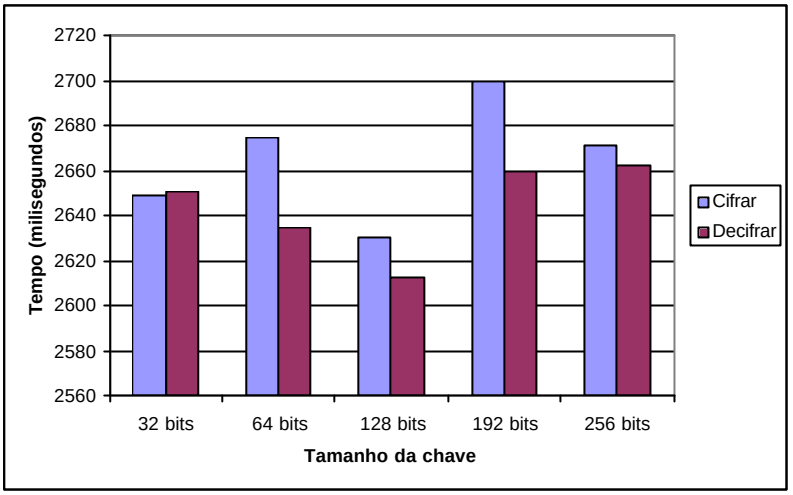

Figura 4.23 – Cifragem e Decifragem de um arquivo de 2 Mbytes

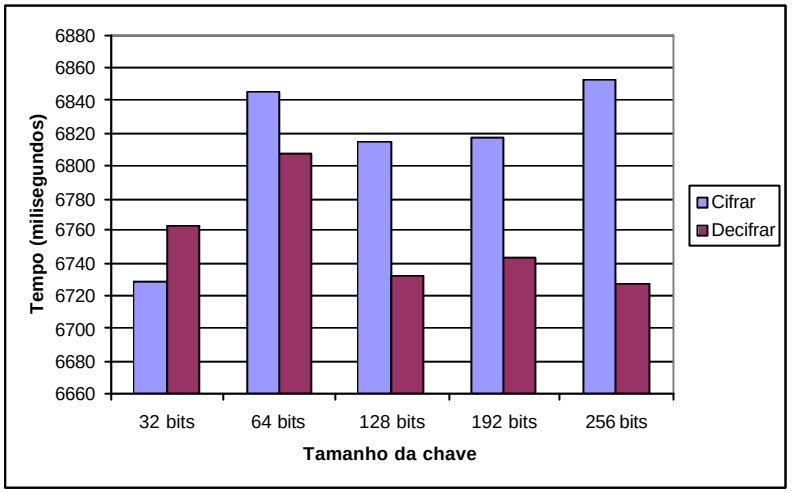

Figura 4.24 – Cifragem e Decifragem de um arquivo de 5 Mbytes

Observa-se nas Figuras 4.20 a 4.24 que a chave de 128 bits, utilizada no processo de cifragem, apresentou bom desempenho. Para arquivos de tamanho 2 Mbytes, 1 Mbyte e 500 Kbytes essa chave mostrou o melhor desempenho.

No processo de decifragem, a chave de 128 bits mostrou melhor desempenho para os arquivos de tamanho 2 Mbytes, 1 Mbyte e 500 Kbytes. Resultado semelhante ao obtido no processo de cifragem. Pode-se então concluir que esse tamanho de chave é o ideal para o algoritmo estudado.

Quanto ao desempenho da chave de 256 bits, o resultado esperado era o pior desempenho para todos as dimensões de arquivos. No entanto, no processo de cifragem para arquivos com dimensões de 2 Mbytes, 1 Mbyte e 100 Kbytes o desempenho dessa chave não foi o pior. Sendo que, nos processos de decifragem esse fato ocorreu para arquivos com dimensões de 5 Mbytes e 100 Kbytes. Além disso, para arquivos de 5 Mbytes, no processo de decifragem, essa chave mostra o melhor desempenho.

Outro fator importante obtido está relacionado à pequena alteração de tempos obtidos para os diversos comprimentos de chave. O aumento do comprimento da chave não diminui de forma decisiva o desempenho dos algoritmos.

A seguir estão compiladas algumas considerações sobre avaliações de desempenho entre tamanhos de chave e arquivos de dimensões constantes do algoritmo escrito em linguagem Java. Lembrando que foram utilizadas chaves de 32, 64, 128, 192 e 256 bits. Foram calculadas as variações de desempenho, e essas variações foram sendo representadas em percentuais, para chaves de tamanhos adjacentes.

Exemplificando, para o arquivo de 100 Kbytes obteve-se para chave de 32 bits o tempo de médio de cifragem de 140,4 milisegundos com desvio padrão de 0,4899 e o tempo médio de decifragem de 144,6 milisegundos com desvio padrão de 7,0408. Para a chave de 64 bits o tempo médio de cifragem de 152 milisegundos com desvio padrão de 8,9592 e o tempo médio de decifragem de 144,933 milisegundos com desvio padrão de 9,1102. Os tempos médios obtidos estão muito próximos, determinando que no processo de cifragem a chave de 64 bits mostra-se 7,6316% menos eficiente que a chave de 32 bits e no processo de decifragem mostra-se 0,23 % menos eficiente.

De forma similar foram calculadas todas as variações de eficiência para as chaves com quantidade de bits adjacentes, 64 bits para 128 bits, 128 bits para 192bits e de 192 bits para 256 bits, e os resultados mostram que há pouca mudança no tempo quando se muda o

tamanho da chave, as variações obtidas foram sempre inferiores a 8% (ver tabelas 4.2.a e 4.2.b).

As tabelas que seguem mostram as variações de eficiência para os processos de cifragem e de decifragem, comparando comprimento de chaves adjacentes, para o algoritmo implementado em linguagem Java.

A tabela 4.2.a mostra o tempo médio de processamento (milisegundos) e a variação de eficiência (%) do processo de cifragem para arquivos de diversas dimensões (100Kbytes, 500 Kbytes, 1 Mbyte, 2 Mbytes e 5 Mbytes) e diferentes tamanhos de chaves (32, 64, 128, 192 e 256 bits). A tabela 4.2.b mostra, de forma análoga, o tempo médio de processamento (milisegundos) e a variação de eficiência do processo de decifragem para os mesmos arquivos utilizados no processo de cifragem.

Tabela 4.2.a – Variação de eficiência na cifragem para o algoritmo em linguagem Java

Chave (bits)	**Dimensão de Arquivo**									
	100 kbytes		**500 kbytes**		**1 Mbyte**		**2 Mbytes**		**5 Mbytes**	
	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)
32	140,400		682,200		1364,400		2648,933		6729,000	
64	152,000	-7,6315	680,133	0,3038	1394,867	-2,1842	2674,067	-0,9398	6844,733	-1,6908
128	147,000	3,4013	675,066	0,7505	1330,133	4,8666	2630,667	1,6497	6814,400	0,4451
192	159,600	-7,8947	689,466	-2,0885	1357,333	-2,0039	2698,867	-2,5269	6816,800	-0,0352
256	158,266	0,8424	694,800	-0,7676	1352,067	0,3895	2670,800	-0,1176	6852,067	-0,5146

Tabela 4.2.b – Variação de eficiência na decifragem para o algoritmo em linguagem Java

Chave (bits)	**Dimensão de Arquivo**									
	100 kbytes		**500 kbytes**		**1 Mbyte**		**2 Mbytes**		**5 Mbytes**	
	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)	Tempo Médio	Variação (%)
32	144,600		677,000		1357,400		2650,133		6762,600	
64	144,933	-0,2299	664,533	1,8760	1372,867	-1,1266	2634,400	0,5972	6807,133	-0,6542
128	146,000	-0,7305	661,333	0,4838	1315,533	4,3581	2612,600	0,8344	6732,200	1,1130
192	146,666	-0,4545	673,800	-1,8502	1361,400	-3,3698	2659,333	-1,7573	6743,667	-0,1700
256	144,866	1,2425	681,133	-1,0766	1374,067	-0,9218	2662,467	1,0508	6727,133	0,2457

4.5 Comparação de Desempenho entre as Linguagens C e Java

Nesse tópico são realizadas as comparações de desempenho entre as implementações nas linguagens C e Java, quanto ao tempo de cifragem e decifragem, bem como a velocidade dos dois processos em Mbits por segundo (Mbits/s).

4.5.1 Tempo de Cifragem e Decifragem

Como pode ser observado nos gráficos que são apresentados a seguir, o desempenho do algoritmo Blowfish implementado na linguagem C apresentou-se superior ao desempenho na linguagem Java, fato que pode ser explicado devido à característica da linguagem Java ser interpretada. Outro fato que também pode ser observado é a escala do gráfico, pode-se perceber que o aumento no tamanho da chave não gera grande aumento nos processos de cifragem e decifragem.

Conforme explicitado, os testes realizados consideraram condições idênticas de dimensões de arquivos e comprimentos de chaves. Os testes foram realizados em um mesmo equipamento (máquina – Pentium 4 1,6GHz com 128Mbytes de memória RAM) e os desvios padrões obtidos contribuem para considerar consistentes os dados obtidos. A implementação em linguagem Java apresentou um aumento de aproximadamente 5 vezes maior na temporização dos processos de cifragem e decifragem em relação à implementação na linguagem C, independentemente do tamanho da chave, e do tamanho do arquivo (ver Figuras 4.25 a 4.29).

Os resultados obtidos foram organizados nas Figuras 4.25 a 4.29, comparando os desempenhos temporais entre as implementações nas linguagens C e Java

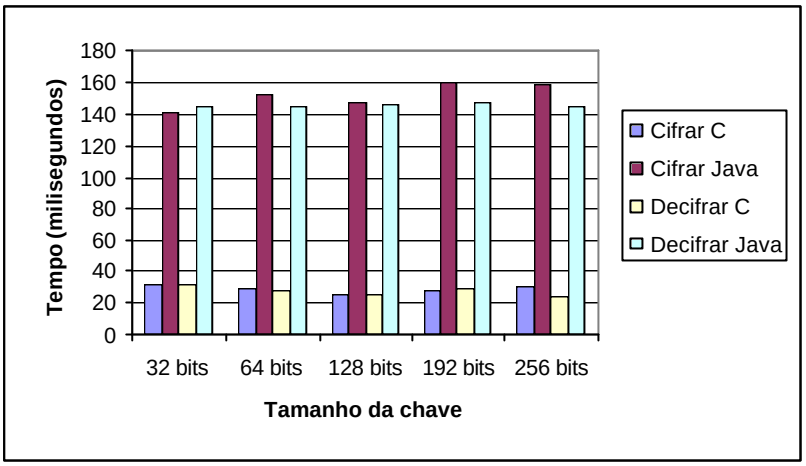

Figura 4.25 – Comparação cifragem e decifragem entre C e Java (100 kbytes)

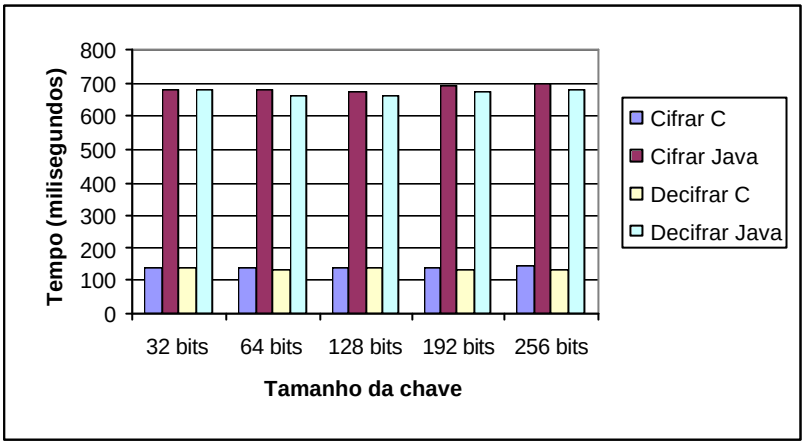

Figura 4.26 – Comparação cifragem e decifragem entre C e Java (500 kbytes)

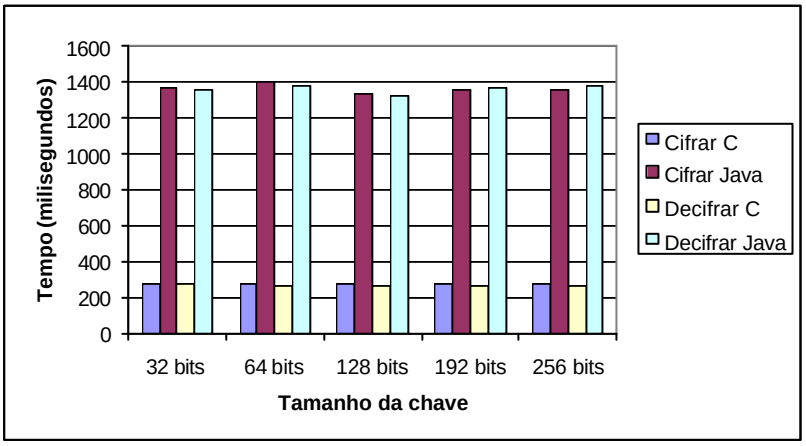

Figura 4.27 – Comparação cifragem e decifragem entre C e Java (1 Mbyte)

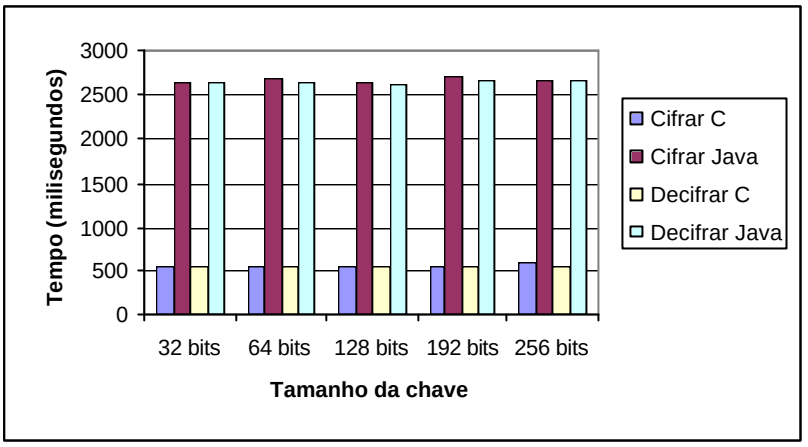

Figura 4.28 – Comparação cifragem e decifragem entre C e Java (2 Mbytes)

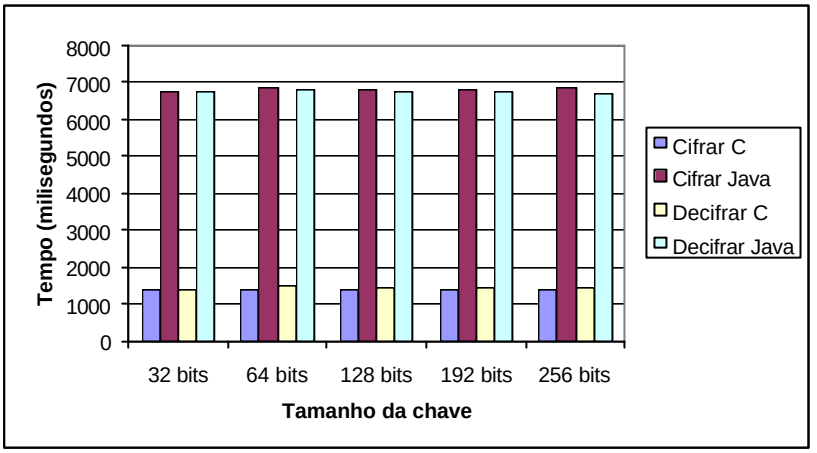

Figura 4.29 – Comparação cifragem e decifragem entre C e Java (5 Mbytes)

As considerações que seguem foram obtidas da análise das tabelas 4.1.a, 4.1.b, 4.2.a e 4.2.b juntamente com as Figuras 4.5 a 4.9 referentes à implementação na linguagem C e as Figuras 4.20 a 4.24 referentes a implementação na linguagem Java. Essa metodologia propiciou as comparações e as principais considerações:

- a implementação em linguagem C mostra-se mais eficiente que a implementação em Java. Esse resultado já era esperado, pois se atribui esse fato principalmente às características de geração de códigos nas diferentes linguagens. O código gerado pela linguagem C é compilado para o código nativo da máquina específica e o código gerado pela linguagem Java é compilado para *bytecodes* que ainda serão interpretados pela máquina virtual Java;

- a chave de 128 bits apresentou bom desempenho em ambas linguagens. Para arquivos com dimensões de 2 Mbytes essa chave mostrou o melhor desempenho nas duas linguagens;
- a chave de 32 bits não mostra o melhor desempenho tanto em C quanto Java. O melhor desempenho dessa chave, tanto no processo de cifragem quanto no de decifragem aconteceu apenas com arquivos com dimensões de 100Kbytes implementados na linguagem Java;
- a chave de 64 bits não apresenta bom desempenho nas duas linguagens, apresentando o pior desempenho no processo de cifragem na linguagem Java para arquivos com dimensão de 1 Mbytes (1395 milisegundos), e no processo de decifragem para arquivos com dimensões de 5 Mbytes (6810 milisegundos) na linguagem Java e na linguagem C (1470 milisegundos);
- a chave de 192 bits também não apresenta bom desempenho nas duas linguagens. Além disso, apresenta o pior desempenho no processos de cifragem da linguagem Java para arquivos com dimensões de 2Mbytes (2700 milisegundos) e 100 Kbytes (159 milisegundos), e o pior desempenho no processo de decifragem da linguagem Java para arquivo com dimensão de 100 Kbytes (147 milisegundos);
- a chave de 256 bits também apresentou algumas surpresas, apresenta o melhor desempenho no processo de decifragem na linguagem C para arquivos com dimensões de 2 Mbytes (540 milisegundos), 1 Mbytes (263 milisegundos), 500 Kbytes (133 milisegundos) e 100 Kbytes (24 milisegundos) e na linguagem Java para arquivos de 5 Mbytes (6730 milisegundos).
- a implementação na linguagem C é aproximadamente 5 vezes mais rápida do que a implementação na linguagem Java.

4.5.2 Velocidade em Mbits por segundo

As figuras 4.30 a 4.25 demonstram o desempenho do algoritmo Blowfish quanto à velocidade em Mbits por segundo para arquivos de diferentes tamanhos (100 kbytes, 500 kbytes, 1 Mbyte, 2 Mbytes e 5 Mbytes), bem como chaves de tamanhos variáveis (32 bits, 64 bits, 128 bits, 192 bits e 256 bits).

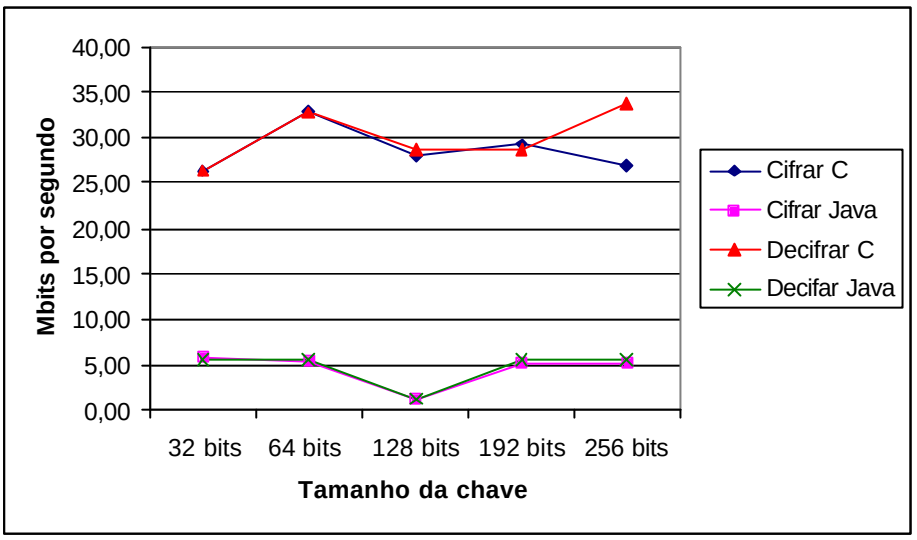

Figura 4.30 – Velocidade em Mbits/seg para um arquivo de 100 kbytes

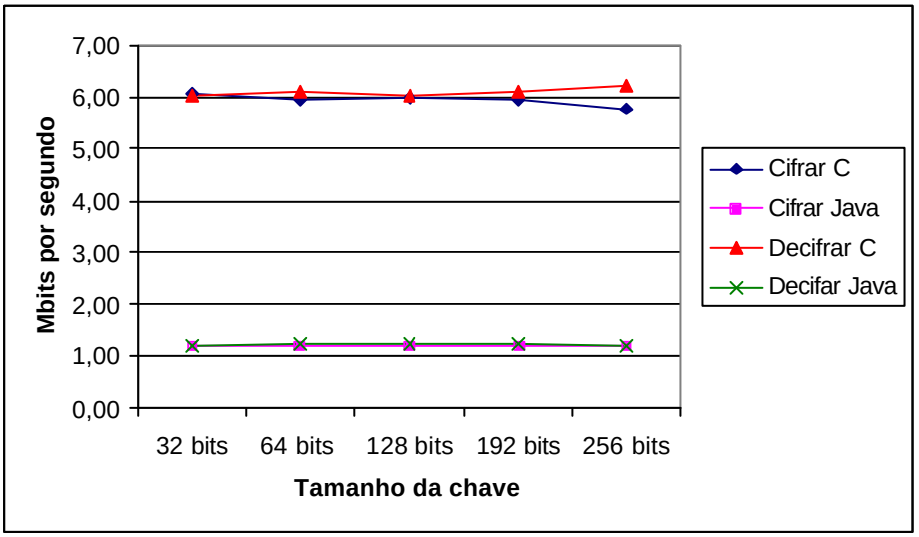

Figura 4.31 – Velocidade em Mbits/seg para um arquivo de 500 kbytes

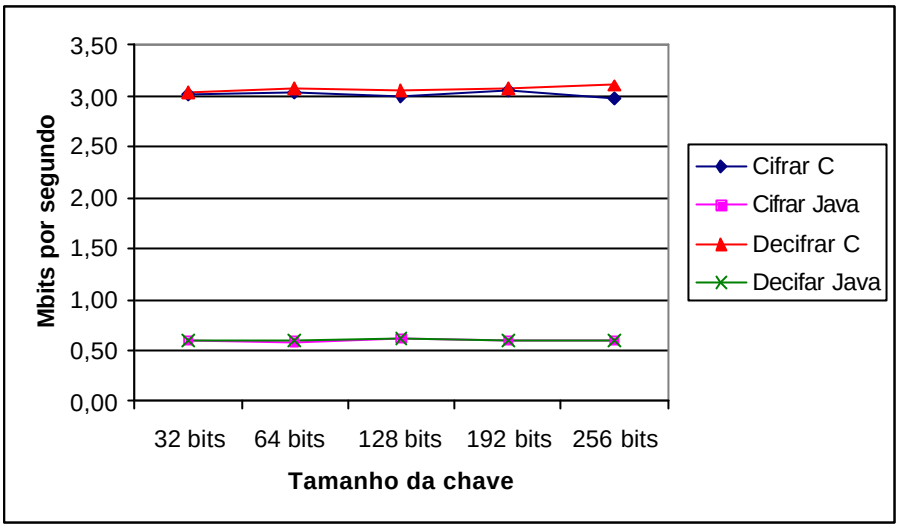

Figura 4.32 – Velocidade em Mbits/seg para um arquivo de 1 Mbyte

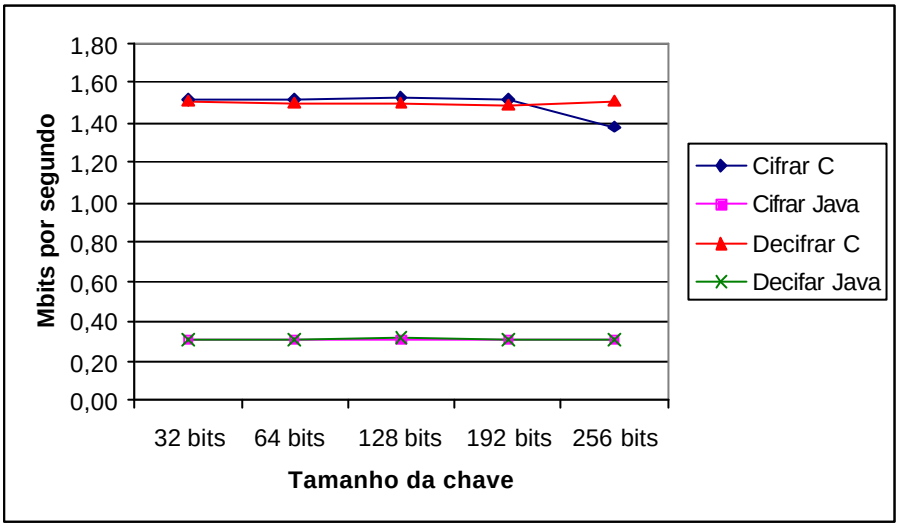

Figura 4.33 – Velocidade em Mbits/seg para um arquivo de 2 Mbytes

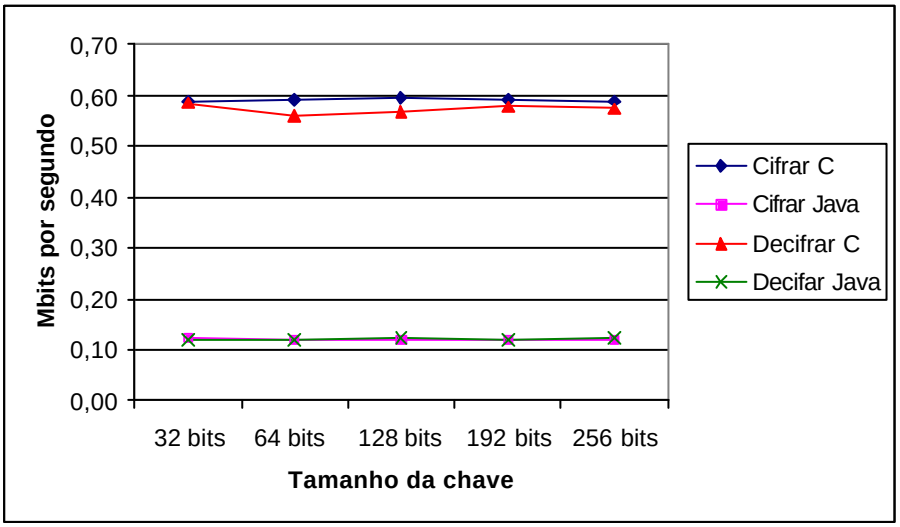

Figura 4.34 – Velocidade em Mbits/seg para um arquivo de 5 Mbytes

Na análise do desempenho percebe-se um crescimento linear na velocidade em Mbits por segundo, à medida que a dimensão do arquivo é reduzida. Se tomarmos como base a Figura 4.30 onde na implementação na linguagem C o processo de cifragem obteve uma média temporal de 25 Mbits/seg para um arquivo de 100 kbytes para uma chave de 32 bits e compararmos com a Figura 4.31 onde na implementação C, no processo de cifragem a média temporal foi de 6 Mbits/seg, para um arquivo de 500 Kbytes e chave de também 32 bits.

Conclui-se que a proporção de aumento de arquivo se reflete na velocidade em Mbits por segundo, pois houve uma diferença de aproximadamente 5 vezes tanto no tamanho do arquivo quanto na velocidade. O mesmo crescimento ou decrescimento linear pode ser observado para as Figuras 4.32 a 4.34 tanto para implementação em C quanto em Java.

Outro ponto interessante é que a curva se mantém constante com o aumento no tamanho da chave.

4.6 Implementação em C e Java utilizando chaves estáticas

Para a realização de comparação de desempenho entre as implementações de alto nível em linguagens C e Java processadas em microcontroladores, foram realizadas implementações do algoritmo Blowfish em C e Java que utilizam chaves estáticas de 32 bits e 128 bits, pois a implementação em microcontroladores, em virtude principalmente de restrições referentes à capacidade de memória e manutenção da integridade do algoritmo, portanto foi necessária a simplificação do algoritmo. O motivo e justificativa dessa implementação será abordado no capítulo 5 em mais detalhes.

Na implementação do algoritmo utilizando chaves estáticas, o processo de inicialização do algoritmo Blowfish foi suprimido, sendo o seu resultado calculado previamente para uma chave de 32 bits e armazenado de forma estática para uso posterior. O

mesmo processo foi utilizado para a implementação com a chave de 128 bits, foram utilizados valores pré-computados das S-boxes e do vetor P.

O código da função em C Inicia_Blowfish e do método construtor da classe Blowfish em Java ficaram da seguinte maneira (Figura 4.35):

Inicia_Blowfish

```
void Inicia_Blowfish(BLOWFISH *ctx) {
    int i, j, k;
  unsigned long data, datal, datar;

    for (i = 0; i < 4; i++)
      for (j = 0; j < 256; j++)
       ctx->S[i][j] = ORIG_S[i][j];

     for (i = 0; i < 18; i++ )
       ctx->P[i] = ORIG_P[i];
}
```

Método construtor da classe Blowfish

```
public Blowfish( ) {
    this.ctx = new blf_ctx();
    System.arraycopy(ps, 0, this.ctx.P, 0, 18);
    System.arraycopy(ks0, 0, this.ctx.S[0], 0, 256);
    System.arraycopy(ks1, 0, this.ctx.S[1], 0, 256);
     System.arraycopy(ks2, 0, this.ctx.S[2], 0, 256);
    System.arraycopy(ks3, 0, this.ctx.S[3], 0, 256);
 }
```

Figura 4.35 - Códigos C e Java Implementação Estática

Nos códigos apresentados anteriormente na Figura 4.35, a chave não é mais passada como argumento pois não há necessidade, já que as S-boxes e o vetor P possuem os valores pré-computados para a chave especifica.

Os gráficos a seguir (4.36 a 4.40) demonstram a comparação de desempenho entre as implementações das chaves estáticas de tamanho 32 bits e 128 bits, nas linguagens C e Java, para arquivos de diferentes tamanhos (100 kbytes, 500 kbytes, 1 Mbyte, 2 Mbytes e 5 Mbytes).

Novamente a implementação em linguagem C apresentou desempenho temporal melhor que a implementação em linguagem Java. Mostrando novamente que o processo de interpretação dos *bytecodes* pela Máquina Virtual Java, causa um impacto considerável no desempenho temporal do algoritmo, mesmo na implementação com chaves estáticas.

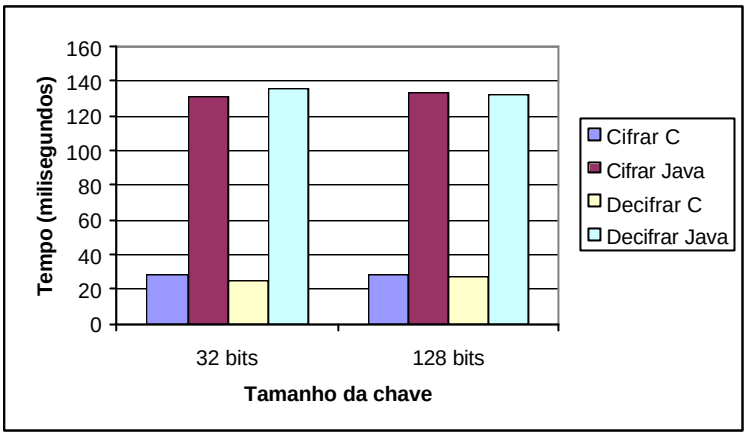

Figura 4.36 – Comparação entre cifragem e decifragem de um arquivo de 100 kytes

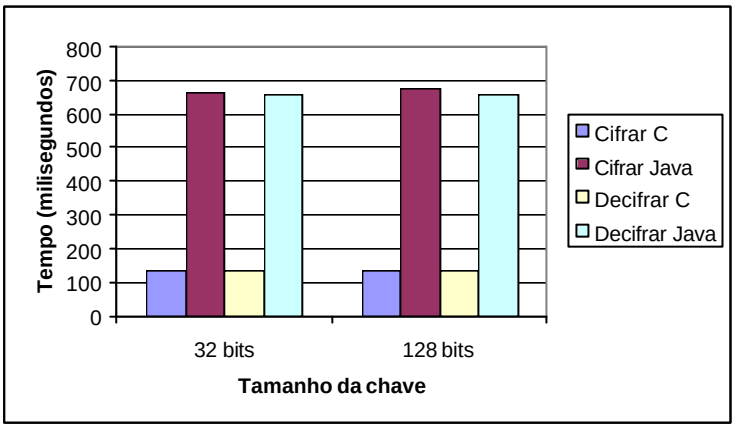

Figura 4.37 – Comparação entre cifragem e decifragem de um arquivo de 500 kytes

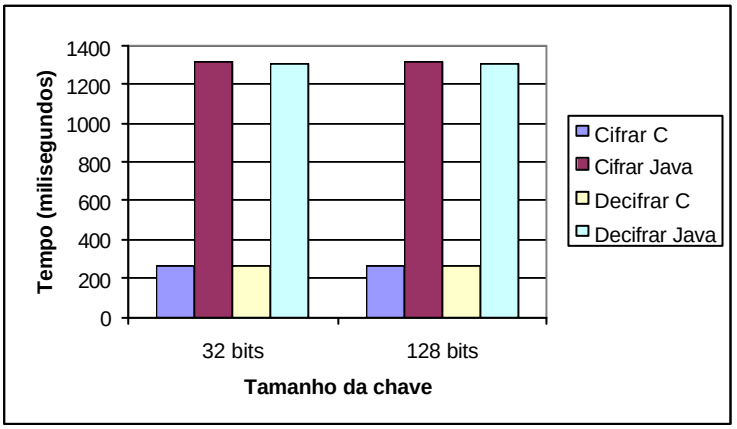

Figura 4.38 – Comparação entre cifragem e decifragem de um arquivo de 1 Mbyte

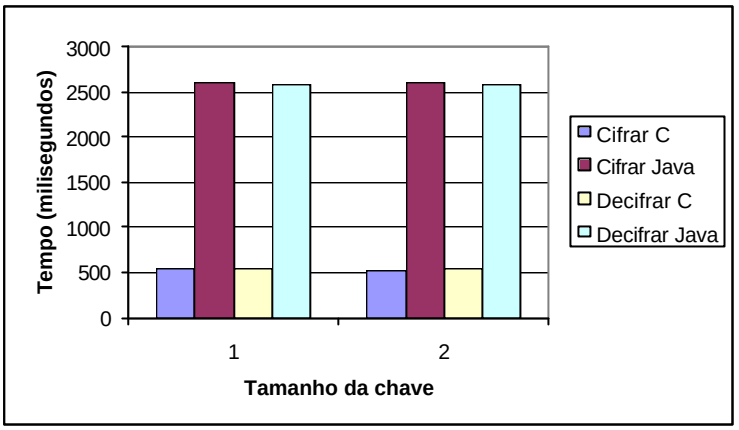

Figura 4.39 – Comparação entre cifragem e decifragem de um arquivo de 2 Mbytes

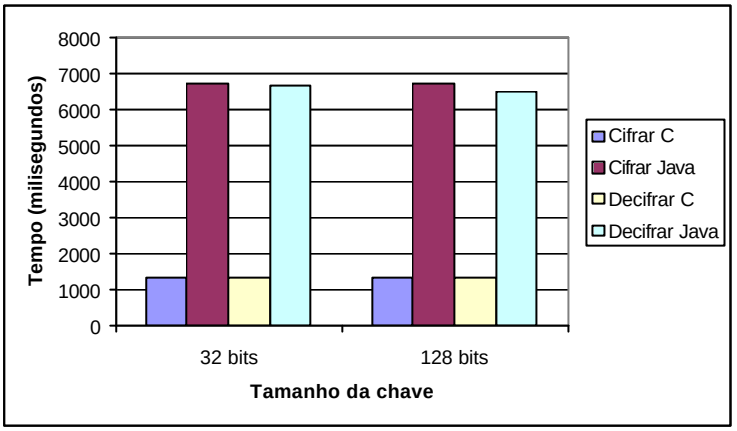

Figura 4.40 – Comparação entre cifragem e decifragem de um arquivo de 5 Mbytes

4.7 Considerações Finais

Esse capítulo deu ênfase às implementações do algoritmo Blowfish nas linguagens C e Java, e a questões de desempenho. Na análise de desempenho os resultados obtidos foram "compilados" e analisados. A linguagem C obteve um melhor desempenho em relação à linguagem Java, essa diferença foi da ordem de 5 vezes, tanto no processo de cifragem quanto no de decifragem, esse resultado era esperado, devido à característica da linguagem Java ser interpretada.

Um fato importante na análise dos tempos de cifragem e decifragem com chaves de diferentes tamanhos está relacionado ao impacto mínimo que o aumento de chave causou nos tempos de cifragem e decifragem, pois o esperado seria, quanto maior a chave utilizada maior o tempo. Avaliações semelhantes foram obtidas nas implementações utilizando chaves para as linguagens C e Java. O mesmo fato ocorreu em ambas as implementações tanto em C quanto Java nas implementações utilizando chaves fixas.

5 O BLOWFISH EM MICROCONTROLADORES

Esse capítulo expõe noções básicas sobre microcontroladores e suas aplicações. Apresenta também principais características da família de microcontroladores PIC e a implementação e análise de desempenho do algoritmo Blowfish implementado num microncontrolador dessa família.

5.1 Noções Básicas

Um microcontrolador caracteriza-se como sendo um circuito integrado que agrega num único componente eletrônico todos os subsistemas necessários e existentes em um microprocessador de uso geral. O microcontrolador possui os seguintes subsistemas internos: unidade lógica aritmética (ULA); memória não volátil (ROM/EPROM); memória volátil (RAM); portas de entrada e saída que propiciam comunicação serial e paralela com o exterior e também conversores analógico/digital (A/D) e digital/analógico (D/A).

O microcontrolador possui todos os componentes existentes na arquitetura de um microprocessador tradicional implementado em um único circuito integrado. Essa característica gera benefícios quando utilizado em aplicações específicas, no entanto, não é adequada quando utilizados em aplicações de uso geral. Assim, quando utilizados em aplicações específicas os microcontroladores são bastante eficientes e quando aplicados em aplicações de uso geral apresentam queda de eficiência dado a restrições contidas em sua arquitetura, tais como: limitação de memória e conjunto restrito de instruções que geralmente atuam em estruturas de bytes ou bits.

Os microprocessadores não possuem limitação de memória, pois essa pode ser expandida através da adição de circuitos externos de memória, tanto volátil quanto não volátil,

e o conjunto de instruções inerentes ao microprocessador é amplo, contendo instruções bastante elaboradas com capacidade de tratar estruturas de dados mais complexas. No entanto, alguns subsistemas intrínsecos a uma arquitetura, tais como portas de comunicação de entrada e saída seriais ou paralelas, são disponibilizados apenas externamente. Esses fatores tornam os microprocessadores com flexibilidade de expansão e úteis para aplicações genéricas, em contrapartida perdem eficiência por necessitarem de mecanismos elaborados de entrada e saída para realizar a troca de informações (comunicação) entre esses componentes externos.

Atualmente os microcontroladores estão sendo utilizados de forma ampla na área de automação. Podemos encontrá-los no controle de eletrodomésticos: microondas, televisores, geladeiras, fogões, DVDs entre outros; no controle de equipamentos contendo eletrônica embarcada como: injeção eletrônica de veículos automotores, sistemas de controle de aeronaves, equipamentos de diagnósticos médicos, automação de sistemas de controle de tráfego, controle e automação de auto-fornos, controle de tornos fresas e similares, aparelhos celulares, marca-passos, automação de equipamentos de embalagens; entre outras aplicações que podem ser também enumeradas. Essa diversificação de utilização dos microcontroladores é amplamente generalizada e comum, que a tecnologia propiciada pelos microcontroladores torna-o transparente ao indivíduo que a utiliza diariamente em serviços que embutem em sua concepção microcontroladores. Os benefícios produzidos pelos mesmos, embora não sentidos ou visualizados pelo indivíduo estão presentes e tornaram-se indispensáveis no seu dia a dia.

Essa universalização do uso de microcontroladores, acontece dado as vantagens que ele oferece quando utilizados em aplicações específicas, principalmente eficiência, baixo custo, flexibilidade de projeto, facilidade de manutenção, custo reduzido e grande disponibilidade e diversidade desses componentes no mercado.

A Figura 5.1 mostra, em diagramas de blocos, um microcontrolador genérico com arquitetura de Von-Neumann.

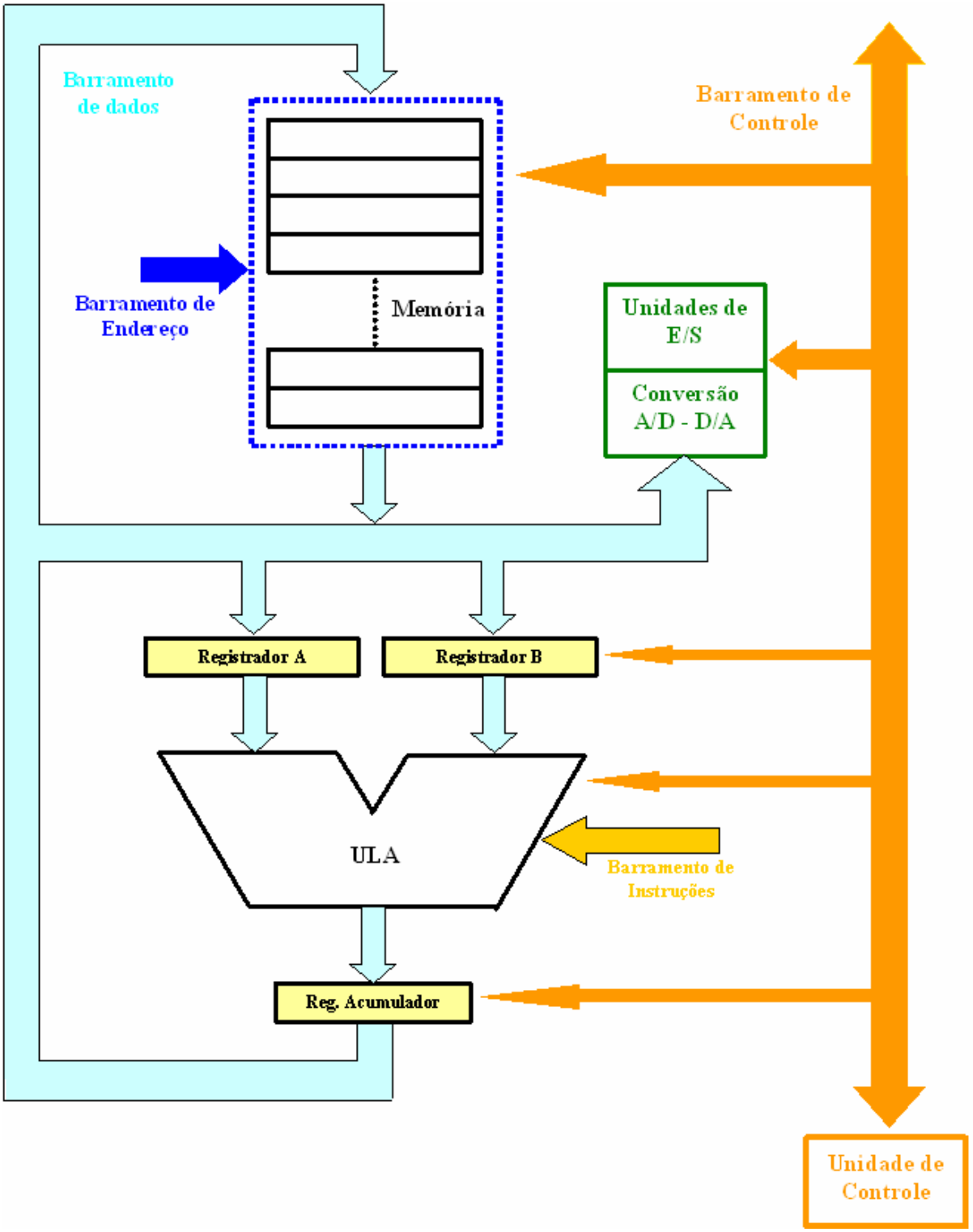

Figura 5.1 - Diagrama de Blocos de um microcontrolador tradicional com Arquitetura de Von-Neumann

5.2 Microcontrolador PIC

Os microcontroladores da família PIC apresentam arquitetura de *Harvard*, diferenciando-o de famílias de microcontroladores tradicionais que apresentam arquitetura de *Von-Neumann*. A principal diferença existente entre as duas propostas está no mecanismo (forma) de execução de instruções.

Conforme apresentado na Figura 5.1, na arquitetura de *Von-Neumann*, as instruções e dados fluem através de barramentos e as instruções são executadas através da alternância entre ciclos de busca de instruções (*fetch*) na memória externa e ciclos de operação dessas instruções. Por sua vez, o ciclo de operação de instruções é composto por uma alternância entre: ciclos de busca (*fetch*) de operandos na memória (ciclos de leitura), ciclos de execução da operação na ULA e ciclos de escrita do resultado da operação na memória externa.

A Figura 5.2 mostra, em diagrama de blocos, um microcontrolador PIC genérico com arquitetura *Harvard*. O tempo de execução de uma instrução pode ser otimizado através de mecanismos que propiciam a antecipação de ciclos de busca de instruções em sua memória interna. Esse mecanismo permite a execução de um ciclo de busca de uma instrução a ser executada posteriormente, durante o ciclo de operação de uma instrução atual. Dessa forma o tempo total de execução de instruções é reduzido pois o tempo de busca de instruções é eliminado do tempo total de execução de instruções. Essa eficiência é enfatizada quando são implementados algoritmos seqüenciais para a execução de uma determinada tarefa. A arquitetura de *Harvard* explora essa organização e execução seqüencial de instruções para obter maior eficiência (DATASHEET, 2003).

Os microcontroladores PIC apresentam um conjunto reduzido de instruções, em média 35 instruções. Como já descrito anteriormente, dado a simplicidade dessas instruções,

elas são executadas de forma eficiente, no entanto exigem maior esforço de programação para a execução de tarefas mais elaboradas.

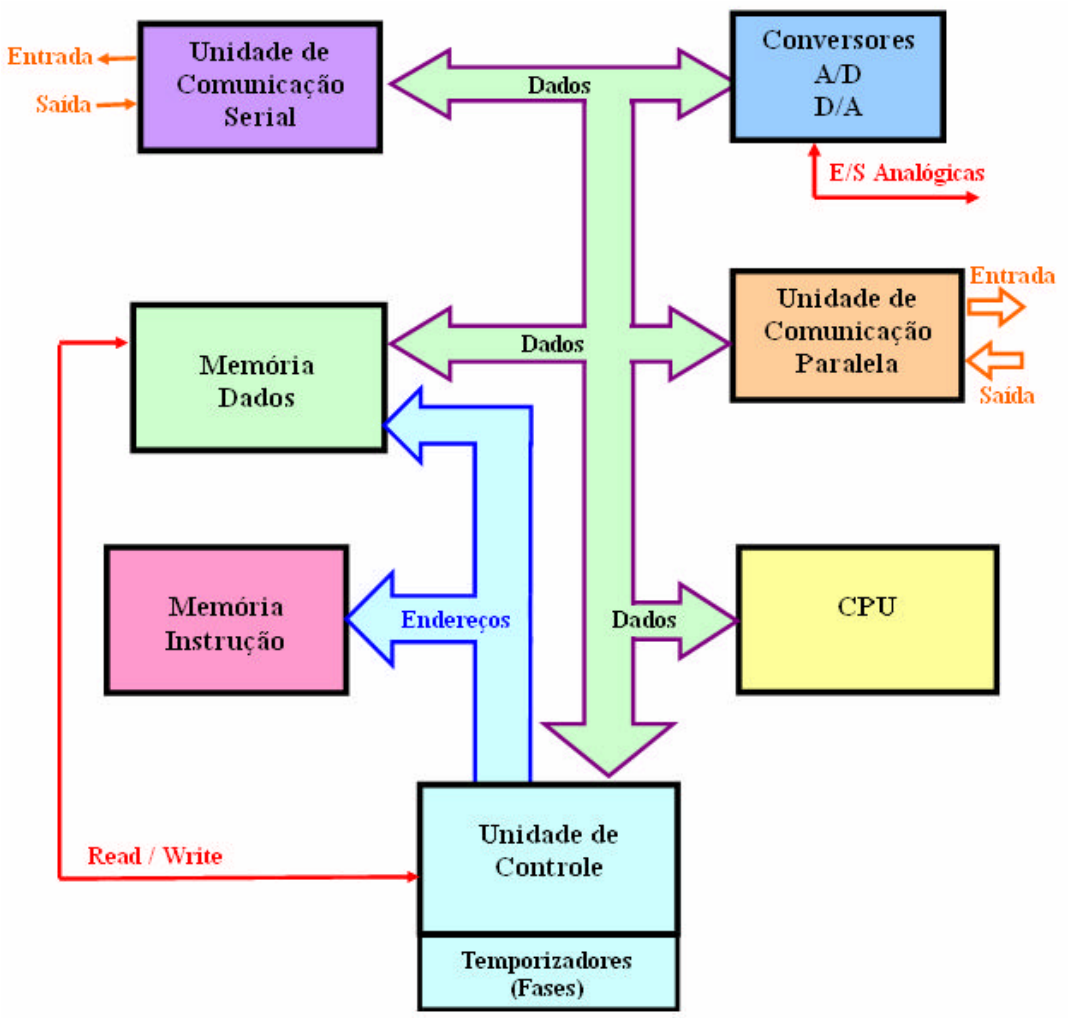

Figura 5.2 - Diagrama de blocos de um Microcontrolador PIC (SOUZA, 2002)

A Figura 5.2 mostra, em diagramas de blocos, um microcontrolador da família PIC.

Nos microcontroladores PIC, o barramento de dados é sempre de oito bits e o barramento de instruções pode conter 12, 14 ou 16 bits dependendo do microcontrolador. Essa arquitetura permite a construção de instruções cujo formato inclui código da instrução e seus operandos, exigindo apenas uma palavra de memória para armazenar a instrução.

5.2.1 Ciclos de Máquina

O microcontrolador PIC possui uma entrada externa de relógio (*clock*) com freqüência de 4 MHz (mega hertz). O relógio interno é de 1 MHz caracterizando um ciclo de máquina de 1 MHz. O *clock* externo é gerado por um oscilador cuja freqüência de trabalho é determinada por um cristal. O sinal gerado pelo oscilador é fornecido ao microcontrolador através da entrada denominada OSC1 (pino OSC1) e o microcontrolador possui um divisor de freqüência interna para obter 4 fases distintas que determinam 4 ciclos de máquina necessários para a execução de cada instrução, denominados de fase Q1, Q2, Q3 e Q4 respectivamente (DATASHEET, 2003).

Durante a execução de uma instrução, o registrador contador de instruções é atualizado sempre na fase Q1, isso permite a busca da próxima instrução na memória enquanto a instrução atual está sendo durante o tempo determinado nas fases Q2, Q3 e Q4. A instrução obtida com a antecipação do ciclo de busca é armazenada no registrador de instruções sempre na fase Q4.

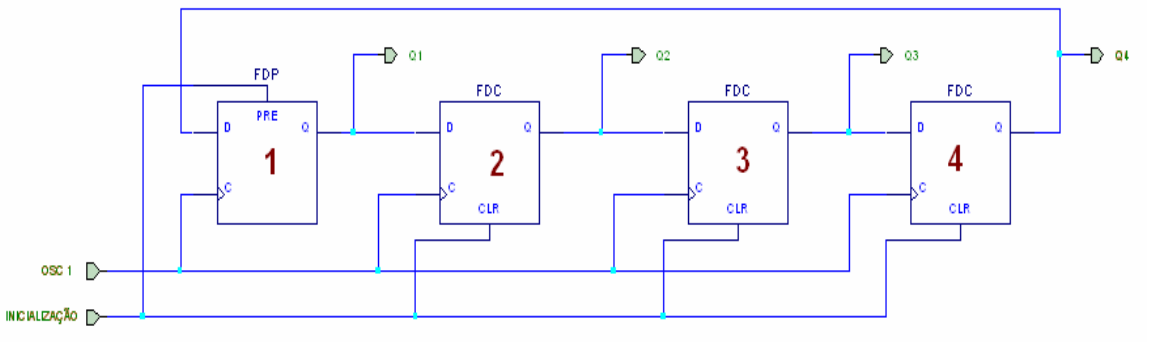

Figura 5.3 - Circuito divisor de freqüência e gerador de 4 fases distintas.

A Figura 5.3 mostra o circuito eletrônico que realiza a divisão de freqüência introduzida no pino OSC1 (4MHz) de um microcontrolador PIC (DATASHEET, 2003).

Trata-se de um circuito seqüencial síncrono composto por quatro 4 *Flip-Flops* tipo D, cujas saídas de dados “Q” identificam as 4 fases que compõem todos os ciclos necessários para a execução de uma instrução. A Figura 5.4 apresenta o diagrama de ondas do circuito divisor de freqüência. As fases Q1, Q2, Q3 e Q4 são consideradas ativas quando assumem o valor lógico alto (*High*). O circuito apresentado na Figura 5.3 e a simulação do mesmo circuito apresentada na Figura 5.4 foram gerados no software *Project Manager* da empresa XILINX (XILINX, 2005).

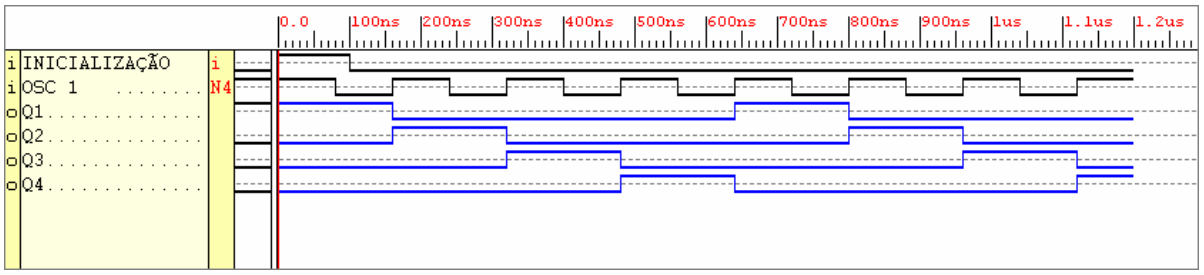

Figura 5.4 - Diagrama de ondas do circuito divisor de freqüência da Figura 5.3

5.2.2 Grupo de Instruções

Como explicado anteriormente os microcontroladores PIC possuem um conjunto de instruções reduzido em média 35 instruções. A tabela 5.1 explicita o conjunto de operações básicas do microcontrolador da família PIC. Os termos utilizados na construção dos nomes das instruções e seus argumentos, para melhor entendimento da tabela 5.1 são os seguintes:

- Work: Trata-se de um registrador temporário para operações da ULA. No Assembler do PIC, ele é conhecido como W. Também é comum chamá-lo de acumulador;
- File: Referência a um registrador (posição de memória) propriamente dito. Foi utilizada a letra F para sua representação nos nomes de instruções e f nos argumentos delas;

- Literal: Um número qualquer que pode ser escrito na forma decimal. Hexadecimal ou binária. Utiliza-se a letra L para sua representação nos nomes de instruções e k nos argumentos delas;
- Destino: O local onde deve ser armazenado o resultado da operação. Existem somente dois destinos possíveis: F, que guardará o resultado no próprio registrador passado como argumento, ou W, que colocará o resultado em Work. Na verdade, na sintaxe das instruções, o destino deve ser expresso pelos números 0 (W) e 1 (F);
- Bit: Refere-se a um bit especifico dentro de um byte. Foi utilizada a letra B para sua representação nos nomes das instruções e b nos argumentos delas;
- Teste: Quando se quer testar o estado de um bit, para descobrir se ele é zero ou um. Foi utilizada a letra T para representá-lo nos nomes das instruções;
- Skip: Significa “pulo”, e é utilizado para a criação de desvios, pulando a próxima linha. Foi utilizada a letra S para representá-lo nos nomes das instruções.
- Set: Refere-se ao ato de setar um bit, isto é, torná-lo equivalente a UM. Foi utilizada a letra S para representá-lo nos nomes das instruções;
- Clear: Refere-se ao *clear* de um bit, isto é, torná-lo equivalente a ZERO. Foi utilizada a letra C para representá-lo nos nomes das instruções.
- Zero: Algumas instruções podem gerar desvios se o resultado da operação efetuada for zero. Neste caso, foi utilizada a letra Z para indicar tal condição.

Os demais termos utilizados são específicos das ações realizadas pelas instruções e são praticamente auto-explicativos, tais como:

ADD: Operação Soma;

AND: Operação Lógica “E”;

CLR: Operação Limpar, zerar (*Clear*);

COM: Operação Complemento;

DEC: Operação Decremento de uma unidade;

INC: Operação Incremento de uma unidade;

IOR: Operação Lógica “OU”

MOV: Operação Mover, transferir para algum lugar;

RL: Operação Rotacionar 1 bit para a esquerda;

RR: Operação Rotacionar um 1 bit para a direita;

SUB: Operação Subtração

SWAP: Operação Inversão entre as partes alta e baixa de um registrador

XOR: Operação Lógica “OU exclusivo”

Tabela 5.1 – Resumo das Instruções básicas de um microcontrolador da família PIC (SOUZA, 2002)

Operações com registradores		
Instrução	**Argumentos**	**Descrição**
ADDWF	f, d	Soma W e f guardando o resultado em d
ANDWF	f, d	Lógica “E” entre W e f, guardando o resultado em d
CLRF	f	Limpar f
COMF	f,d	Calcula o complemento de f, guardando o resultado em d
DECF	f,d	Decrementa f, guardando o resultado em d
DECFSZ	f,d	Decrementa f, guardando o resultado em d, e pula para a próxima linha se o resultado for zero.
INCF	f,d	Incrementa f, guardando o resultado em d
INCFSZ	f,d	Incrementa f, guardando o resultado em d, e pula a próxima linha se o resultado for zero
IORWF	f,d	Lógica “OU” entre W e f, guardando o resultado em d
MOVF	f,d	Move f para f (copia)
MOVWF	f	Move W para f
RLF	f,d	Rotaciona f 1 bit para a esquerda
RRF	f,d	Rotaciona f 1 bit para a direita
SUBWF	f,d	Subtrai W de f (f – W), guardando o resultado em d
SWAPF	f,d	Executa uma inversão entre as partes alta e baixa de f, guardando o resultado em d
XORWF	f,d	Lógica “OU exclusivo” entre W e f, guardando o resultado em d
Operações com literais		
Instrução	**Argumentos**	**Descrição**
ADDLW	k	Soma k com W, guardando o resultado em W
ANDLW	k	Lógica “E” entre k e W, guardando o resultado em W
IORLW	k	Lógica “OU” entre k e W, guardando o resutado em W
MOVLW	k	Move k para W
SUBLW	k	Subtrai W de k (k – W) guardando o resultado em W
XORLW	k	Lógica “OU exclusivo” entre k e W, guardando o resultado em W

Tabela 5.1 – Resumo das Instruções básicas de um microcontrolador da família PIC (SOUZA, 2002)

Operações com bits		
Instrução	**Argumentos**	**Descrição**
BCF	f,b	Impõe 0 (zero) ao bit b do registrador f
BSF	f,b	Impõe 1 (zero) ao bit b do argumento f
BTFSC	f,b	Testa o bit b do registrador f, e pula a próxima linha se ele for 0 (zero)
BTFSS	f,b	Testa o bit b do registrador f, e pula a próxima linha se ele for 1 (um)
Controles		
Instrução	**Argumentos**	**Descrição**
CLRW	-	Limpa W
NOP	-	Gasta um ciclo de máquina sem fazer absolutamente nada
CALL	R	Executa a rotina R
CLRWDT	-	Limpa o registrador WDT para não acontecer o reset
GOTO	R	Desvia para o ponto R, mudando o PC
RETFIE	-	Retorna de uma interrupção
RETLW	k	Retorna de uma rotina, com k em W
RETURN	-	Retorna de uma rotina, sem afetar W
SLEEP	-	Coloca o PIC em modo sleep (dormindo) para economia de energia

5.3 Implementação do Algoritmo Blowfish no Microcontrolador PIC

A implementação do algoritmo Blowfish no microcontrolador da família PIC foi realizada utilizando o ambiente de desenvolvimento integrado (IDE – *Integrated Development Environment*) *CodeDesigner Lite* (CODEDESIGN, 2005) em conjunto com o compilador PICBasic Pro, versão 2.40 (PICBASIC, 2005). Esse compilador produz código que pode ser programado em uma variedade de microcontroladores PIC.

O PICBasic gera um arquivo hexadecimal para a configuração do microcontrolador PIC, a partir do programa fonte no CodeDesigner.

Para a realização da implementação do algoritmo foi realizado o estudo da sintaxe e semântica da linguagem utilizada pelo compilador PICBasic, bem como suas características peculiares.

Em uma primeira etapa, o algoritmo Blowfish foi implementado de maneira que o processo de inicialização do algoritmo fosse realizado como nas implementações em C e Java, porém por questões de capacidade de memória do microcontrolador, decidiu-se realizar a implementação do algoritmo de maneira estática onde o processo de inicialização do mesmo

pudesse ser suprimido liberando assim maior capacidade de memória para a implementação. Por este motivo foram realizadas as implementações em linguagem C e Java do algoritmo Blowfish utilizando uma chave estática, isto é, para a possibilidade de comparação de desempenho temporal em relação às implementações em C, Java e no microcontrolador da família PIC.

Em virtude do compilador PICBasic, trabalhar com dados de até 16 bits, houve necessidade de reformular a implementação do algoritmo, em certos aspectos.

As 4 S-boxes do algoritmo como citado no capítulo 3, possuem cada uma 256 dados com tamanho de 32 bits, sendo assim, como o compilador não suporta a manipulação de 32 bits foi necessária a separação em duas partes de cada valor das S-boxes, ou seja, os 16 bits mais significativos e os 16 bits menos significativos. Por exemplo, no caso da S-box implementada em C ou Java com chave estática, um dos valores de uma posição dessa S-box é o hexadecimal e7414d90 (decimal 3879816592), com a divisão em duas partes ficaria da seguinte forma: e741 e 4d90 que são transformados na base decimal para utilização na programação realizada no PICBasic, gerando respectivamente os seguintes valores 59201 e 19856.

Em virtude das mudanças realizadas nas S-boxes, isto é, de 256 entradas para 512 entradas pois seus valores de 32 bits foram separados em 2 de 16 bits cada um, a capacidade de memória do microcontrolador não foi suficiente, bem como as estruturas para manipulação de tais dados pelo compilador PICBasic. Sendo assim as S-boxes foram reduzidas, de tal maneira que seus valores foram compostos somente dos 16 bits mais significativos dos 32 bits originais que foram separados, como citado anteriormente.

O trecho de código na Figura 5.5 demonstra a instrução *lookup2*, compilador PICBasic, que foi utilizada para a implementação das S-boxes. Essa instrução retorna as

entradas de uma tabela de valores. A sintaxe da declaração *lookup*2 (PICBASIC, 2004) é a seguinte:

Lookup2 Índice**,**[Valores{**,**Valores**...**}]**,** Variável

Se o índice é zero, a variável é setada para o primeiro valor, na lista de valores. Se o índice é um, a variável é setada para o segundo valor, e assim por diante. Essa declaração só suporta valores com até de 16 bits de comprimento e apenas 256 valores. Portanto fica clara uma das razões para as mudanças que foram necessárias para a implementação do algoritmo.

Código da primeira S-box

```
LookUp2 s0,[59201, _
        38106, 35773, 38759, 24787, 34149, 55475, 46380, 33961, 6969, 43306, 34931, 60437, _
        54978, 26295, 27843, 16010, 64033, 9906, 50563, 40402, 24365, 45402, 62939, 10122, _
        5674, 56669, 10692, 40974, 49718, 45169, 951, 62250, 24021, 23798, 781, 5505, _
        38591, 38860, 25978, 64307, 32239, 40005, 50823, 52722, 22077, 14779, 36216, 39613, _
        15411, 18654, 12841, 23669, 47608, 971, 32336, 59251, 40450, 30925, 34418, 14328, _
        25027, 29704, 9594, 34831, 6936, 46857, 23086, 40093, 38871, 30502, 51177, 11370, _
        38248, 5995, 17609, 44572, 48557, 36122, 40025, 56417, 20686, 38945, 60652, 50048, _
        8026, 9278, 25850, 56395, 28977, 28196, 22506, 27928, 64475, 37984, 8259, 8440, _
        18198, 25544, 21374, 27394, 10031, 11881, 11687, 52755, 10777, 61259, 57953, 46366, _
        28311, 8834, 30676, 32033, 22531, 30384, 61493, 61658, 7867, 1000, 61733, 18495, _
        33072, 43388, 25647, 6320, 65248, 23563, 34635, 35486, 57429, 34429, 54932, 30188, _
        32166, 13077, 42939, 20430, 36452, 51027, 52884, 22133, 30312, 36, 21599, 13615, _
        21382, 46948, 26700, 37847, 43141, 64458, 42928, 64924, 50747, 4699, 25298, 47206, _
        41743, 55898, 60723, 57174, 39856, 25270, 38554, 929, 23443, 47417, 22691, 22795, _
        2748, 11405, 4973, 1474, 44849, 53289, 32992, 28365, 45381, 5109, 52743, 51996, _
        37236, 43258, 19550, 57672, 40457, 8181, 46904, 54207, 8873, 11934, 52668, 25756, _
        57207, 65428, 20978, 17737, 22874, 617, 47362, 18231, 14400, 17210, 42509, 45514, _
        36069, 61308, 22086, 38321, 23300, 26341, 14120, 55821, 31791, 26522, 14433, 11446, _
        63418, 45746, 38500, 51321, 4501, 48462, 14668, 22592, 56820, 486, 34013, 42415, _
        43977, 9892, 61140, 20188, 46669, 24816, 25998, 63128, 54488, 23267, 6855, 30364, _
        47478, 63767, 33554, 20172, 58291, 44961, 125, 3171, 14118, 47978, 4169, 52599, _
        53850, 9235, 44698 ], s0temp
```

Figura 5.5 – Código S-Box1

No trecho de código ilustrado na Figura 5.5 observa-se que o primeiro valor da declaração *lookup* (59201) é o valor na base decimal dos 16 bits mais significativos referentes ao valor na base hexadecimal de e7414d90. O vetor P foi implementado da mesma forma e com o uso da declaração *lookup2*.

Outra mudança no algoritmo foi com relação às iterações da rede de Feistel, que foram dobradas de 16 iterações para 32 iterações, novamente devido à restrição de manipulação de apenas 16 bits, porém o incremento da rede de Feistel foi realizado de duas em duas unidades. O código na Figura 5.6 explicita a implementação da rotina que realiza a cifragem de um bloco de 64 bits, isto é, que foi tratado como dois vetores de 2 posições (XL e XR) cada um, sendo assim possibilitando o tratamento em blocos de 32 bits.

Rotina Cifrar

```
Cifrar:

    i VAR WORD

    ;---- variáveis de 32bits
    XL VAR WORD[2]
    XR VAR WORD[2]

    ;---- recebe bloco a ser cifrado
    XL[0] = L[0]
    XL[1] = L[1]

    XR[0] = R[0]
    XR[1] = R[1]

    For i = 0 TO i < 32 STEP 2
        P = I
        XL[0] = XL[0] ^ Ptemp

        P = i + 1
        XL[1] = XL[0] ^ Ptemp

        ;---- chamada da rotina F
        GoSub functionF
        XR[0] = y ^ XR[0]

        GoSub functionF
        XR[1] = y ^ XR[1]

        Swap XL[0], XR[0]
        Swap XL[1], XR[1]
    Next i
    ;------------- fim loop for -----------

    Swap XL[0], XR[0]
    Swap XL[1], XR[1]

    P = i
    XR[0] = XR[0] ^ Ptemp

    P = i + 1
    XR[1] = XR[1] ^ Ptemp

    P = i + 2
    XL[0] = XL[0] ^ Ptemp

    P = i + 3
    XL[1] = XL[1] ^ Ptemp

    L[0] = XL[0
    L[1] = XL[1]

    R[0] = XR[0]
    R[1] = XR[1]

    Return
```

Figura 5.6 – Código Rotina Cifrar

No trecho de código da rotina Cifrar, o vetor XL aglutina dois blocos de 16 bits, formando um bloco de 32 bits, o mesmo é realizado pelo vetor XR. Logo após inicia-se as iterações na rede de Feistel e o processo de cifragem do bloco de 64 bits. Uma instrução muito útil do compilador PICBasic é a *Swap* (PICBASIC, 2004) que realiza a troca de valores

entre duas variáveis sem a necessidade do uso de qualquer variável intermediária e possui a seguinte sintaxe:

Swap Variável, Variável

O código da rotina que implementa a decifragem do bloco de 64 bits é apresentado na Figura 5.7, e a implementação é parecida com o processo de cifragem, porém como já citado nos Capítulos 3 e 4 é realizado o processo de maneira inversa.

Rotina Decifrar

```
Decifrar

    XL[0] = L[0]
    XL[1] = L[1]

    XR[0] = R[0]
    XR[1] = R[1]

    For i = 35 TO i > 3 STEP -2
       P = i
       XL[0] = XL[0] ^ Ptemp

       P = i - 1
       XL[1] = XL[0] ^ Ptemp

       GoSub functionF
       XR[0] = y ^ XR[0]

       GoSub functionF
       XR[1] = y ^ XR[1]

       Swap XL[0], XR[0]
       Swap XL[1], XR[1]
    Next i
```

```
    Swap XL[0], XR[0]
    Swap XL[1], XR[1]

    P = 3
    XR[0] = XR[0] ^ Ptemp

    P = 2
    XR[1] = XR[1] ^ Ptemp

    P = 1
    XL[0] = XL[0] ^ Ptemp

    P = 0
    XL[1] = XL[1] ^ Ptemp

    L[0] = XL[0]
    L[1] = XL[1]

    R[0] = XR[0]
    R[1] = XR[1]
  Return
```

Figura 5.7 – Código Rotina Decifrar

A seguir observa-se na Figura 5.8 o trecho de código da rotina functionF, na qual o compilador PICBasic oferece algumas operações que auxiliaram no seu desenvolvimento como os operadores para operações e manipulação sobre bits tais como o de AND (&), XOR (^) e deslocamento de bits (à direita).

Rotina FunctionF

```
functionF:
    d VAR BYTE
    c VAR BYTE
    b VAR BYTE
    a VAR BYTE
    temp VAR BYTE
    tempA VAR WORD

    tempB VAR WORD
    tempC VAR WORD
    tempD VAR WORD

    d = XL[0] & %11111111
    temp = (XL[0] >> 8) & %11111111
    c = temp
    b = XL[1] & %11111111
    temp = (Xl[1] >> 8) & %11111111

    a = temp
    s0 = a
    tempA = s0temp
    s1 = b
    tempB = s1temp
    s2 = c
    tempC = s2temp
    s3 = d
    tempD = s3temp

    y = tempA + tempB
    y = y ^ tempC
    y = y + tempD
Return
```

Figura 5.8 – Código Rotina FunctionF

5.4 Análise de Desempenho

Para a realização da análise de desempenho temporal do algoritmo Blowfish implementado no microcontrolador PIC em relação às implementações nas linguagens C e Java com chaves estáticas, foram realizados testes quanto ao tempo de cifragem e decifragem de um bloco de 64 bits.

Para a obtenção da temporização na linguagem C, foi criado um arquivo com tamanho múltiplo de 64 bits, mais precisamente com dimensão de 416000 bits, equivalendo a

6500 blocos de 64 bits. Para esses parâmetros com uma chave de 32 bits foi obtido o tempo de 110 milisegundos no processo de cifragem e de 109 milisegundos para o processo de decifragem. Para compatibilizar esses tempos com os tempos obtidos para um bloco de 64 bits utilizado no microcontrolador PIC foi calculado de forma linear o tempo que o algoritmo em C gastaria para avaliar um bloco de também 64 bits. O tempo obtido nessa conversão foi de 0,01692 milisegundos para o processo de cifragem e de 0,01677 milisegundos para o processo de decifragem. O mesmo processo foi realizado para a chave de 128 bits. A tabela 5.2 fornece os respectivos valores obtidos em laboratório com chaves de 32 bits e 128 bits.

É possível observar que na tabela 5.2 os tempos dos processos de cifragem e decifragem para as implementações em C e Java foram somados, pois na implementação no PIC16F628 o tempo obtido refere-se à execução do algoritmo dos processos de cifragem e decifragem somados, não havendo individualização dos processos.

Tabela 5.2 – Tempos de execução do algoritmo com chaves estáticas

	Chave 32 bits		**Chave 128 bits**	
	Tempo (µs)	**Ciclos de Clock**	**Tempo (µs)**	**Ciclos de Clock**
PIC16F628	9501	38000	9501	38000
C chave estática (PC)	33,69	53904	33.85	54160
Java chave estática (PC)	1683,454	2693526	1696,034	2713654

Deve-se também ressaltar e informar as diferenças existentes entre as características dos ambientes utilizados para a execução dos algoritmos implementados em C e Java e o microcontrolador utilizado para a implementação em *hardware*. O microcontrolador utilizado (PIC16F628) possui freqüência de trabalho 4MHz e um ciclo de *clock* de 250ns (nanosegundos). O microcomputador utilizado para executar os programa em C e Java possui um processador Pentium IV com freqüência de trabalho de 1,6 GHz e tempo de ciclo de instrução de 0.625ns (nanosegundos). Considerando essas diferentes características, o tempo obtido no microcontrolador (9501µs), equivale a 38.000 ciclos de *clock*. Já o tempo obtido no

microcomputador para a execução do algoritmo escrito em linguagem C (33.69µs) equivale a 53.904 ciclos de clock. Assim, conclui-se que embora o tempo obtido no microcontrolador PIC seja maior ele utilizou quantidade menor (70%) de ciclos de clock para executar o algoritmo escrito em Assembly para chave de 32 bits. Para chave de 128 bits foram obtidos resultados muito próximos.

5.5 Considerações Finais

Nesse capítulo foi dada ênfase à implementação do algoritmo Blowfish implementado em hardware, mais precisamente no microcontrolador da família PIC.

Com relação à comparação de desempenho entre as implementações nas linguagens C e Java, obteve-se o resultado esperado, ou seja, o algoritmo implementado em hardware dedicado apresentou melhor desempenho temporal, apesar das modificações realizadas devido às restrições impostas pelo hardware em questão para a implementação do mesmo. Optou-se por uma implementação de chaves estáticas objetivando manter a segurança do algoritmo Blowfish, pois por questões de capacidade de memória seria necessário suprimir algumas estruturas de grande importância no processo de criptografia como as S-boxes, pois essas deveriam ser simplificadas de tal forma para serem inseridas na memória do microcontrolador, fato que poderia prejudicar a integridade do algoritmo.

6 CONCLUSÕES

O presente trabalho de pesquisa teve como objetivo principal o estudo, implementação e avaliação de desempenho do algoritmo criptográfico simétrico Blowfish tanto em software (implementado-o nas linguagens C, Java) quanto em hardware (através da implementação do algoritmo para microcontroladores da família PIC).

Durante o projeto foi também enfatizado e estudado o projeto AES em virtude de sua relevância para a área de criptografia.

Os objetivos iniciais foram plenamente alcançados e comprovados através de testes das implementações do algoritmo estudado nas linguagens citadas e no microcontrolador PIC16F628.

O algoritmo Blowfish foi estudado e apresentado de forma detalhada para a obtenção de maior eficiência nas implementações realizadas. A estrutura lógica do algoritmo foi dominada, assim como foram definidas estruturas de dados adequadas para os parâmetros utilizados nos processos de cifragem e decifragem de dados que subsidiaram as implementações nos ambientes descritos..

Após o domínio detalhado do algoritmo, o mesmo foi implementado numa primeira instância nas linguagens C e Java para realizar avaliações de desempenho. Nessa avaliação foram utilizados critérios que avaliaram tempo de execução, taxa de transferência de dados, dimensões de arquivos, comprimento de chaves (32, 64, 128, 192 e 256 bits), eficiência dos processos de cifragem e decifragem de dados.

Na análise de desempenho realizada através de testes das implementações do algoritmo nas linguagens C e Java, foram obtidos os seguintes resultados e informações:

- a linguagem C obteve um melhor desempenho em relação á linguagem Java com relação aos tempos de cifragem e decifragem, sendo um resultado já esperado,

demonstrando o impacto que a característica de interpretação dos *bytecodes* realizada pela JVM (*Java Virtual Machine*) exerceu sobre o código gerado.

- na análise dos dados obtidos nos processos de cifragem e decifragem com chaves de diferentes tamanhos notou-se que o aumento de chave gerou impacto mínimo no aumento de tempo de execução e taxas transferência de dados. O esperado seria, quanto maior a chave utilizada maior o tempo. Resultados semelhantes foram obtidos nas implementações nas linguagens C e Java;
- a chave de 128 bits apresentou bom desempenho em ambas linguagens. Para arquivos com dimensões de 2 Mbytes essa chave mostrou o melhor desempeno nas duas linguagens;
- fato semelhante ocorreu em ambas às implementações do algoritmo utilizando chaves fixas nas mesmas linguagens.
- a implementação do algoritmo na linguagem C apresentou ser 5 vezes mais rápida que a implementação em Java.

Na última etapa do projeto, o algoritmo foi implementado para sua utilização no microcontrolador PIC16F628 caracterizando e enfatizando a implementação do algoritmo em ambiente de “hardware”.

As avaliações de desempenho realizadas em laboratório permitiram comparações entre as implementações nas linguagens C e Java e a implementação realizada no microcontrolador. Os resultados obtidos foram os esperados, o algoritmo implementado em hardware dedicado apresentou melhor desempenho temporal.

Em conseqüência das limitações de memória impostas pelo microcontrolador, na implementação em *hardware* foram utilizadas chaves estáticas objetivando eficiência e a

integridade do algoritmo. Os resultados obtidos através de testes são apresentados e discutidos em detalhes nesse documento.

Os objetivos definidos foram plenamente alcançados através da metodologia adotada para a implementação do projeto. Todas as implementações propostas foram realizadas e avaliadas através de testes em laboratórios, organizadas e discutidas nesse documento.

Sugerimos para a continuidade do projeto num futuro próximo, as seguintes iniciativas:

- utilização de recursos *multithread,* disponibilizados na linguagem Java, objetivando aumento da eficiência do algoritmo implementado nessa linguagem;
- implementação do algoritmo Blowfish em microcontroladores implementados em FPGAs em particular no microcontrolador desenvolvido no projeto entitulado “UPEM: Um Sistema que Integra Microcontroladores e FPGAs- Arquiteturais, Protótipo e Desempenho de Periféricos”, apresentado por César Giacomini Penteado em sua dissertação apresentada no programa de mestrado do UNIVEM (PENTEADO, 2005);
- implementação do algoritmo Blowfish utilizando DSPs (*Digital Signal Processing*) e também em outros microcontroladores das famílias PIC e Motorola.
- utilização dessa mesma metodologia para realizar o estudo, implementação e avaliação de outros algoritmos criptográficos tais como Twofish, Cast-128 e AES entre outros para obter parâmetros para a avaliar o desempenho entre esses algoritmos e dessa forma gerar subsídios para a utilização dos mesmos para as aplicações diversas desses algoritmos criptográficos.

REFERÊNCIAS

ADAMS, C. **The CAST-128 Encryption Algorithm**. Network Working Group - Request for Comments: 2144, 1997. Disponível em: <http://www.ietf.org/rfc/rfc2144.txt>. Acesso em: 18 fev. 2005.

ANDERSON, R.; BIHAM, E.; KNUDSEN, L. **Serpent: A Proposal for the Advanced Encryption Standard**. In: AES Round 1 Candidate Algorithms, 1999. Disponível em: <http://www.cl.cam.ac.uk/ftp/users/rja14/serpent.pdf>. Acesso em: 20 abr. 2005

PICBASIC. **PicBasic Pro Compiler, microEngineering Labs, Inc.,** Compilador Basic para Microcontroladores PIC.
Disponível em: <http://www.rentron.com/PicBasic/products/PICBASIC-PRO.htm>. Acesso em: 10 set. 2005.

_____. **PicBasic Pro Compiler - Manual, microEngineering Labs, Inc.**, 2004. Disponível em: <http://www.melabs.com/downloads/pbpm304.pdf>. Acesso em: 10 set. 2005.

BIHAM, E. S. **A Note on Comparing the AES Candidates**. In: Second AES Conference. Technical Report, Computer Science Department, Technio. Israel Institute of Technology, 1999. Disponível em: <http://www.cs.technion.ac.il/~biham/>. Acesso em: 10 jun. 2005.

CODEDESIG. **CodeDesigner Lite, microEnginnering Labs Inc.**, Disponível em: <http://picbasic.com/resources/win_ide.htm>. Acesso em: 10 set. 2005.

DAEMEM, J.; RIJMEN, V. **AES Proposal: Rijndael**. In: AES Round 1 Candidate Algorithms, 1999.
Disponível em: <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf>. Acesso em: 20 abr. 2005

DATASHEET. **PIC16F62X FLASH-Based 8-Bit CMOS Microcontrollers Datasheet**. MicrochipTechnology Inc. 2003.
Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/40300c.pdf>. Acesso em: 09 set. 2005.

DEITEL, H., M; DEITEL, P., J. **Java como Programar**. 3ª ed. Porto Alegre: Bookman, 2001.

DESKEY. **Cryptography Research, DES Key Search Machine**. 2001. Disponível em: <http://www.cryptography.com/resources/whitepapers/DES.html>. Acesso em: 25 mar. 2005.

IBM. **MARS – a candidate cipher for AES**. 1999. Disponível em: < http://www.research.ibm.com/security/mars.pdf>. Acesso em: 20 abr. 2005

FIPS197 – **Announcing the Advanced Encryption Standard (AES)**. Federal Information Processing Standards Publication 197, 2001.
Disponível em: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. Acesso em: 15 mai. 2005.

FIPS46-3. **Data Encryption Standard**. Federal Information Processing Standards Publication 46-3, 2001. Disponível em: <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>. Acesso em: 15 mai. 2005.

HORSTMANN, C., S; CORNELL, G. **Core Java 2**: Volume 1 – Fundamentos. São Paulo: Makron Books, 2001. v. 1.

MENEZES, A., J; OORSCHOT, P., C; VANSTONE, S., A. **Handbook of Applied Cryptography**. New York: CRC Press, Fifth Printing, 1997.

MESERVE, J. **DES code cracked in record time Network World**, 1999. Disponível em: <http://www.networkworld.com/news/1999/0120cracked.html>. Acesso em: 23 mar. 2005.

MORENO, E., D., O. et al. **Projeto, Desempenho e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs)**. Marília: Bless, 2003.

MORENO, E., D., O; PEREIRA, F., D; CHIARAMONTE, R., B. **Criptografia em Software e Hardware**. Marília: Novatec Ltda, 2005.

NIST01. **AES Round 2 Information**. National Institute of Standards and Technology - Federal Register, v.64, n.178, p.50058-50061, 1999.
Disponível em: <http://csrc.nist.gov/CryptoToolkit/aes/round2/round2.htm>. Acesso em: 29 mar. 2005.

NIST02. **First AES Candidate Conference (AES1)**. National Institute of Standards and Technology, 1998.
Disponível em: <http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/aes1conf.htm>. Acesso em: 29 mar. 2005.

NIST03. **Announcing Request for Candidate Algorithm Nominations for The Advanced Encryption Standard (AES)**. National Institute of Standards and Technology - Federal Register, v.62, n.177, p.48051-48058, 1997.
Disponível em: <http://csrc.nist.gov/CryptoToolkit/aes/pre-round1/aes_9709.htm#sec3>. Acesso em: 29 mar. 2005.

PENTEADO, C., G. **UPEM:** Um Sistema que Integra Microcontroladores e FPGAs-Arquiteturais, Protótipo e Desempenho de Periféricos. 2004. 170 f. Grau: Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

RIVEST, R; L. et al. **The RC6 Block Cipher**. RSA Laboratories. 1998. Disponível em: <http://theory.lcs.mit.edu/~rivest/rc6.pdf>. Acesso em: 20 abr. 2005

STALLINGS, W. **Cryptography and Networks Security: Principles and Practices**. ed. Prentice Hall, Second Edition, 1995.

SCHNEIER, B; WHITING, D. **A Performance Comparison of the Five AES Finalists**, 2000. Disponível em: <http://www.schneier.com/paper-aes-comparison.pdf>. Acesso em: 5 mar. 2005.

SCHNEIER, B. et al. **Twofish: A 128-Bit Block Cipher**, 1998. Disponível em: <http://www.schneier.com/paper-twofish-paper.pdf>. Acesso em: 25 fev. 2005.

SCHNEIER, B. **Applied Cryptograph y: Protocols, Algorithms and Source in C**. 2nd ed. New York: John Wiley and Sons, 1996.

_____. **Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)**. In: Fast Software Encryption, Cambridge Security Workshop Proceedings (Dec 1993), pp. 191-204, Springer-Verlag, 1994. Disponível em: < http://www.schneier.com/paper-blowfish-fse.html>. Acesso em: 18 fev. 2005

SEBESTA, R., W. **Conceitos de Linguagens de Programação**. Porto Alegre: Bookman, 2002

SOUZA, D., J. **Desbravando o PIC:** Baseado no Microcontrolador PIC16F84. 2.ed. São Paulo: Érica, 2002.

TANENBAUM, A., S. **Redes de Computadores**. 4. ed. Rio de Janeiro: Campus, 2003.

TOKTZ, V. **Criptologia Numaboa**. Disponível em: <http://www.numaboa.com/criptologia>. Acesso em: 23 mar. 2005.

XILINX. **Xilinx: Programmable Logical Devices, FPGA & CLPD**. Disponível em: <http://www.xilinx.com>. Acesso em: 01 de out. 2005

.

APÊNDICE I

Códigos das implementações em linguagem C, Java e *assembly* do microcontrolador PIC16F628.

Código C Chave Dinâmica

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 16

static const unsigned long ORIG_P[16 + 2] = {
    0x243F6A88L, 0x85A308D3L, 0x13198A2EL, 0x03707344L,
    0xA4093822L, 0x299F31D0L, 0x082EFA98L, 0xEC4E6C89L,
    0x452821E6L, 0x38D01377L, 0xBE5466CFL, 0x34E90C6CL,
    0xC0AC29B7L, 0xC97C50DDL, 0x3F84D5B5L, 0xB5470917L,
    0x9216D5D9L, 0x8979FB1BL
};

static const unsigned long ORIG_S[4][256] = {
{ 0xD1310BA6L, 0x98DFB5ACL, 0x2FFD72DBL, 0xD01ADFB7L,
0xB8E1AFEDL, 0x6A267E96L, 0xBA7C9045L, 0xF12C7F99L,
0x24A19947L, 0xB3916CF7L, 0x0801F2E2L, 0x858EFC16L,
0x636920D8L, 0x71574E69L, 0xA458FEA3L, 0xF4933D7EL,
0x0D95748FL, 0x728EB658L, 0x718BCD58L, 0x82154AEEL,
0x7B54A41DL, 0xC25A59B5L, 0x9C30D539L, 0x2AF26013L,
0xC5D1B023L, 0x286085F0L, 0xCA417918L, 0xB8DB38EFL,
0x8E79DCB0L, 0x603A180EL, 0x6C9E0E8BL, 0xB01E8A3EL,
0xD71577C1L, 0xBD314B27L, 0x78AF2FDAL, 0x55605C60L,
0xE65525F3L, 0xAA55AB94L, 0x57489862L, 0x63E81440L,
0x55CA396AL, 0x2AAB10B6L, 0xB4CC5C34L, 0x1141E8CEL,
0xA15486AFL, 0x7C72E993L, 0xB3EE1411L, 0x636FBC2AL,
0x2BA9C55DL, 0x741831F6L, 0xCE5C3E16L, 0x9B87931EL,
0xAFD6BA33L, 0x6C24CF5CL, 0x7A325381L, 0x28958677L,
0x3B8F4898L, 0x6B4BB9AFL, 0xC4BFE81BL, 0x66282193L,
0x61D809CCL, 0xFB21A991L, 0x487CAC60L, 0x5DEC8032L,
0xEF845D5DL, 0xE98575B1L, 0xDC262302L, 0xEB651B88L,
0x23893E81L, 0xD396ACC5L, 0x0F6D6FF3L, 0x83F44239L,
0x2E0B4482L, 0xA4842004L, 0x69C8F04AL, 0x9E1F9B5EL,
0x21C66842L, 0xF6E96C9AL, 0x670C9C61L, 0xABD388F0L,
0x6A51A0D2L, 0xD8542F68L, 0x960FA728L, 0xAB5133A3L,
0x6EEF0B6CL, 0x137A3BE4L, 0xBA3BF050L, 0x7EFB2A98L,
0xA1F1651DL, 0x39AF0176L, 0x66CA593EL, 0x82430E88L,
0x8CEE8619L, 0x456F9FB4L, 0x7D84A5C3L, 0x3B8B5EBEL,
0xE06F75D8L, 0x85C12073L, 0x401A449FL, 0x56C16AA6L,
0x4ED3AA62L, 0x363F7706L, 0x1BFEDF72L, 0x429B023DL,
0x37D0D724L, 0xD00A1248L, 0xDB0FEAD3L, 0x49F1C09BL,
0x075372C9L, 0x80991B7BL, 0x25D479D8L, 0xF6E8DEF7L,
0xE3FE501AL, 0xB6794C3BL, 0x976CE0BDL, 0x04C006BAL,
0xC1A94FB6L, 0x409F60C4L, 0x5E5C9EC2L, 0x196A2463L,
0x68FB6FAFL, 0x3E6C53B5L, 0x1339B2EBL, 0x3B52EC6FL,
0x6DFC511FL, 0x9B30952CL, 0xCC814544L, 0xAF5EBD09L,
0xBEE3D004L, 0xDE334AFDL, 0x660F2807L, 0x192E4BB3L,
0xC0CBA857L, 0x45C8740FL, 0xD20B5F39L, 0xB9D3FBDBL,
0x5579C0BDL, 0x1A60320AL, 0xD6A100C6L, 0x402C7279L,
0x679F25FEL, 0xFB1FA3CCL, 0x8EA5E9F8L, 0xDB3222F8L,
0x3C7516DFL, 0xFD616B15L, 0x2F501EC8L, 0xAD0552ABL,
0x323DB5FAL, 0xFD238760L, 0x53317B48L, 0x3E00DF82L,
0x9E5C57BBL, 0xCA6F8CA0L, 0x1A87562EL, 0xDF1769DBL,
0xD542A8F6L, 0x287EFFC3L, 0xAC6732C6L, 0x8C4F5573L,
0x695B27B0L, 0xBBCA58C8L, 0xE1FFA35DL, 0xB8F011A0L,
0x10FA3D98L, 0xFD2183B8L, 0x4AFCB56CL, 0x2DD1D35BL,
0x9A53E479L, 0xB6F84565L, 0xD28E49BCL, 0x4BFB9790L,
0xE1DDF2DAL, 0xA4CB7E33L, 0x62FB1341L, 0xCEE4C6E8L,
0xEF20CADAL, 0x36774C01L, 0xD07E9EFEL, 0x2BF11FB4L,
0x95DBDA4DL, 0xAE909198L, 0xEAAD8E71L, 0x6B93D5A0L,
0xD08ED1D0L, 0xAFC725E0L, 0x8E3C5B2FL, 0x8E7594B7L,
0x8FF6E2FBL, 0xF2122B64L, 0x8888B812L, 0x900DF01CL,
0x4FAD5EA0L, 0x688FC31CL, 0xD1CFF191L, 0xB3A8C1ADL,
0x2F2F2218L, 0xBE0E1777L, 0xEA752DFEL, 0x8B021FA1L,
0xE5A0CC0FL, 0xB56F74E8L, 0x18ACF3D6L, 0xCE89E299L,
0xB4A84FE0L, 0xFD13E0B7L, 0x7CC43B81L, 0xD2ADA8D9L,
0x165FA266L, 0x80957705L, 0x93CC7314L, 0x211A1477L,
0xE6AD2065L, 0x77B5FA86L, 0xC75442F5L, 0xFB9D35CFL,
0xEBCDAF0CL, 0x7B3E89A0L, 0xD6411BD3L, 0xAE1E7E49L,
0x00250E2DL, 0x2071B35EL, 0x226800BBL, 0x57B8E0AFL,
0x2464369BL, 0xF009B91EL, 0x5563911DL, 0x59DFA6AAL,
0x78C14389L, 0xD95A537FL, 0x207D5BA2L, 0x02E5B9C5L,
0x83260376L, 0x6295CFA9L, 0x11C81968L, 0x4E734A41L,
0xB3472DCAL, 0x7B14A94AL, 0x1B510052L, 0x9A532915L,
0xD60F573FL, 0xBC9BC6E4L, 0x2B60A476L, 0x81E67400L,
0x08BA6FB5L, 0x571BE91FL, 0xF296EC6BL, 0x2A0DD915L,
0xB6636521L, 0xE7B9F9B6L, 0xFF34052EL, 0xC5855664L,
0x53B02D5DL, 0xA99F8FA1L, 0x08BA4799L, 0x6E85076AL },
{ 0x4B7A70E9L, 0xB5B32944L, 0xDB75092EL, 0xC4192623L,
0xAD6EA6B0L, 0x49A7DF7DL, 0x9CEE60B8L, 0x8FEDB266L,
0xECAA8C71L, 0x699A17FFL, 0x5664526CL, 0xC2B19EE1L,
0x193602A5L, 0x75094C29L, 0xA0591340L, 0xE4183A3EL,
0x3F54989AL, 0x5B429D65L, 0x6B8FE4D6L, 0x99F73FD6L,
0xA1D29C07L, 0xEFE830F5L, 0x4D2D38E6L, 0xF0255DC1L,
0x4CDD2086L, 0x8470EB26L, 0x6382E9C6L, 0x021ECC5EL,
0x09686B3FL, 0x3EBAEFC9L, 0x3C971814L, 0x6B6A70A1L,
0x687F3584L, 0x52A0E286L, 0xB79C5305L, 0xAA500737L,
0x3E07841CL, 0x7FDEAE5CL, 0x8E7D44ECL, 0x5716F2B8L,
0xB03ADA37L, 0xF0500C0DL, 0xF01C1F04L, 0x0200B3FFL,
0xAE0CF51AL, 0x3CB574B2L, 0x25837A58L, 0xDC0921BDL,
0xD19113F9L, 0x7CA92FF6L, 0x94324773L, 0x22F54701L,
0x3AE5E581L, 0x37C2DADCL, 0xC8B57634L, 0x9AF3DDA7L,
0xA9446146L, 0x0FD0030EL, 0xECC8C73EL, 0xA4751E41L,
0xE238CD99L, 0x3BEA0E2FL, 0x3280BBA1L, 0x183EB331L,
0x4E548B38L, 0x4F6DB908L, 0x6F420D03L, 0xF60A04BFL,
0x2CB81290L, 0x24977C79L, 0x5679B072L, 0xBCAF89AFL,
0xDE9A771FL, 0xD9930810L, 0xB38BAE12L, 0xDCCF3F2EL,
0x5512721FL, 0x2E6B7124L, 0x501ADDE6L, 0x9F84CD87L,
0x7A584718L, 0x7408DA17L, 0xBC9F9ABCL, 0xE94B7D8CL,
0xEC7AEC3AL, 0xDB851DFAL, 0x63094366L, 0xC464C3D2L,
0xEF1C1847L, 0x3215D908L, 0xDD433B37L, 0x24C2BA16L,
0x12A14D43L, 0x2A65C451L, 0x50940002L, 0x133AE4DDL,
0x71DFF89EL, 0x10314E55L, 0x81AC77D6L, 0x5F11199BL,
0x043556F1L, 0xD7A3C76BL, 0x3C11183BL, 0x5924A509L,
0xF28FE6EDL, 0x97F1FBFAL, 0x9EBABF2CL, 0x1E153C6EL,
0x86E34570L, 0xEAE96FB1L, 0x860E5E0AL, 0x5A3E2AB3L,
0x771FE71CL, 0x4E3D06FAL, 0x2965DCB9L, 0x99E71D0FL,
0x803E89D6L, 0x5266C825L, 0x2E4CC978L, 0x9C10B36AL,
0xC6150EBAL, 0x94E2EA78L, 0xA5FC3C53L, 0x1E0A2DF4L,
0xF2F74EA7L, 0x361D2B3DL, 0x1939260FL, 0x19C27960L,
0x5223A708L, 0xF71312B6L, 0xEBADFE6EL, 0xEAC31F66L,
0xE3BC4595L, 0xA67BC883L, 0xB17F37D1L, 0x018CFF28L,
0xC332DDEFL, 0xBE6C5AA5L, 0x65582185L, 0x68AB9802L,
0xEECEA50FL, 0xDB2F953BL, 0x2AEF7DADL, 0x5B6E2F84L,
0x1521B628L, 0x29076170L, 0xECDD4775L, 0x619F1510L,
0x13CCA830L, 0xEB61BD96L, 0x0334FE1EL, 0xAA0363CFL,
0xB5735C90L, 0x4C70A239L, 0xD59E9E0BL, 0xCBAADE14L,
0xEECC86BCL, 0x60622CA7L, 0x9CAB5CABL, 0xB2F3846EL,
0x648B1EAFL, 0x19BDF0CAL, 0xA02369B9L, 0x655ABB50L,
0x40685A32L, 0x3C2AB4B3L, 0x319EE9D5L, 0xC021B8F7L,
0x9B540B19L, 0x875FA099L, 0x95F7997EL, 0x623D7DA8L,
0xF837889AL, 0x97E32D77L, 0x11ED935FL, 0x16681281L,
0x0E358829L, 0xC7E61FD6L, 0x96DEDFA1L, 0x7858BA99L,
0x57F584A5L, 0x1B227263L, 0x9B83C3FFL, 0x1AC24696L,
0xCDB30AEBL, 0x532E3054L, 0x8FD948E4L, 0x6DBC3128L,
0x58EBF2EFL, 0x34C6FFEAL, 0xFE28ED61L, 0xEE7C3C73L,
0x5D4A14D9L, 0xE864B7E3L, 0x42105D14L, 0x203E13E0L,
0x45EEE2B6L, 0xA3AAABEAL, 0xDB6C4F15L, 0xFACB4FD0L,
0xC742F442L, 0xEF6ABBB5L, 0x654F3B1DL, 0x41CD2105L,
0xD81E799EL, 0x86854DC7L, 0xE44B476AL, 0x3D816250L,
0xCF62A1F2L, 0x5B8D2646L, 0xFC8883A0L, 0xC1C7B6A3L,
0x7F1524C3L, 0x69CB7492L, 0x47848A0BL, 0x5692B285L,
0x095BBF00L, 0xAD19489DL, 0x1462B174L, 0x23820E00L,
0x58428D2AL, 0x0C55F5EAL, 0x1DADF43EL, 0x233F7061L,
0x3372F092L, 0x8D937E41L, 0xD65FECF1L, 0x6C223BDBL,
0x7CDE3759L, 0xCBEE7460L, 0x4085F2A7L, 0xCE77326EL,
0xA6078084L, 0x19F8509EL, 0xE8EFD855L, 0x61D99735L,
0xA969A7AAL, 0xC50C06C2L, 0x5A04ABFCL, 0x800BCADCL,
0x9E447A2EL, 0xC3453484L, 0xFDD56705L, 0x0E1E9EC9L,
0xDB73DBD3L, 0x105588CDL, 0x675FDA79L, 0xE3674340L,
0xC5C43465L, 0x713E38D8L, 0x3D28F89EL, 0xF16DFF20L,
0x153E21E7L, 0x8FB03D4AL, 0xE6E39F2BL, 0xDB83ADF7L },
{ 0xE93D5A68L, 0x948140F7L, 0xF64C261CL, 0x94692934L,
0x411520F7L, 0x7602D4F7L, 0xBCF46B2EL, 0xD4A20068L,
0xD4082471L, 0x3320F46AL, 0x43B7D4B7L, 0x500061AFL,
0x1E39F62EL, 0x97244546L, 0x14214F74L, 0xBF8B8840L,
0x4D95FC1DL, 0x96B591AFL, 0x70F4DDD3L, 0x66A02F45L,
0xBFBC09ECL, 0x03BD9785L, 0x7FAC6DD0L, 0x31CB8504L,
0x96EB27B3L, 0x55FD3941L, 0xDA2547E6L, 0xABCA0A9AL,
0x28507825L, 0x530429F4L, 0x0A2C86DAL, 0xE9B66DFBL,
0x68DC1462L, 0xD7486900L, 0x680EC0A4L, 0x27A18DEEL,
0x4F3FFEA2L, 0xE887AD8CL, 0xB58CE006L, 0x7AF4D6B6L,
```

```
0xAACE1E7CL, 0xD3375FECL, 0xCE78A399L, 0x406B2A42L,
0x20FE9E35L, 0xD9F385B9L, 0xEE39D7ABL, 0x3B124E8BL,
0x1DC9FAF7L, 0x4B6D1856L, 0x26A36631L, 0xEAE397B2L,
0x3A6EFA74L, 0xDD5B4332L, 0x6841E7F7L, 0xCA7820FBL,
0xFB0AF54EL, 0xD8FEB397L, 0x454056ACL, 0xBA489527L,
0x55533A3AL, 0x20838D87L, 0xFE6BA9B7L, 0xD096954BL,
0x55A867BCL, 0xA1159A58L, 0xCCA92963L, 0x99E1DB33L,
0xA62A4A56L, 0x3F3125F9L, 0x5EF47E1CL, 0x9029317CL,
0xFDF8E802L, 0x04272F70L, 0x80BB155CL, 0x05282CE3L,
0x95C11548L, 0xE4C66D22L, 0x48C1133FL, 0xC70F86DCL,
0x07F9C9EEL, 0x41041F0FL, 0x404779A4L, 0x5D886E17L,
0x325F51EBL, 0xD59BC0D1L, 0xF2BCC18FL, 0x41113564L,
0x257B7834L, 0x602A9C60L, 0xDFF8E8A3L, 0x1F636C1BL,
0x0E12B4C2L, 0x02E1329EL, 0xAF664FD1L, 0xCAD18115L,
0x6B2395E0L, 0x333E92E1L, 0x3B240B62L, 0xEEBEB922L,
0x85B2A20EL, 0xE6BA0D99L, 0xDE720C8CL, 0x2DA2F728L,
0xD0127845L, 0x95B794FDL, 0x647D0862L, 0xE7CCF5F0L,
0x5449A36FL, 0x877D48FAL, 0xC39DFD27L, 0xF33E8D1EL,
0x0A476341L, 0x992EFF74L, 0x3A6F6EABL, 0xF4F8FD37L,
0xA812DC60L, 0xA1EBDDF8L, 0x991BE14CL, 0xDB6E6B0DL,
0xC67B5510L, 0x6D672C37L, 0x2765D43BL, 0xDCD0E804L,
0xF1290DC7L, 0xCC00FFA3L, 0xB5390F92L, 0x690FED0BL,
0x667B9FFBL, 0xCEDB7D9CL, 0xA091CF0BL, 0xD9155EA3L,
0xBB132F88L, 0x515BAD24L, 0x7B9479BFL, 0x763BD6EBL,
0x37392EB3L, 0xCC115979L, 0x8026E297L, 0xF42E312DL,
0x6842ADA7L, 0xC66A2B3BL, 0x12754CCCL, 0x782EF11CL,
0x6A124237L, 0xB79251E7L, 0x06A1BBE6L, 0x4BFB6350L,
0x1A6B1018L, 0x11CAEDFAL, 0x3D25BDD8L, 0xE2E1C3C9L,
0x44421659L, 0x0A121386L, 0xD90CEC6EL, 0xD5ABEA2AL,
0x64AF674EL, 0xDA86A85FL, 0xBEBFE988L, 0x64E4C3FEL,
0x9DBC8057L, 0xF0F7C086L, 0x60787BF8L, 0x6003604DL,
0xD1FD8346L, 0xF6381FB0L, 0x7745AE04L, 0xD736FCCCL,
0x83426B33L, 0xF01EAB71L, 0xB0804187L, 0x3C005E5FL,
0x77A057BEL, 0xBDE8AE24L, 0x55464299L, 0xBF582E61L,
0x4E58F48FL, 0xF2DDFDA2L, 0xF474EF38L, 0x8789BDC2L,
0x5366F9C3L, 0xC8B38E74L, 0xB475F255L, 0x46FCD9B9L,
0x7AEB2661L, 0x8B1DDF84L, 0x846A0E79L, 0x915F95E2L,
0x466E598EL, 0x20B45770L, 0x8CD55591L, 0xC902DE4CL,
0xB90BACE1L, 0xBB8205D0L, 0x11A86248L, 0x7574A99EL,
0xB77F19B6L, 0xE0A9DC09L, 0x662D09A1L, 0xC4324633L,
0xE85A1F02L, 0x09F0BE8CL, 0x4A99A025L, 0x1D6EFE10L,
0x1AB93D1DL, 0x0BA5A4DFL, 0xA186F20FL, 0x2868F169L,
0xDCB7DA83L, 0x573906FEL, 0xA1E2CE9BL, 0x4FCD7F52L,
0x50115E01L, 0xA70683FAL, 0xA002B5C4L, 0x0DE6D027L,
0x9AF88C27L, 0x773F8641L, 0xC3604C06L, 0x61A806B5L,
0xF0177A28L, 0xC0F586E0L, 0x006058AAL, 0x30DC7D62L,
0x11E69ED7L, 0x2338EA63L, 0x53C2DD94L, 0xC2C21634L,
0xBBCBEE56L, 0x90BCB6DEL, 0xEBFC7DA1L, 0xCE591D76L,
0x6F05E409L,0x4B7C0188L, 0x39720A3DL, 0x7C927C24L,
0x86E3725FL, 0x724D9DB9L, 0x1AC15BB4L, 0xD39EB8FCL,
0xED545578L, 0x08FCA5B5L, 0xD83D7CD3L, 0x4DAD0FC4L,
0x1E50EF5EL, 0xB161E6F8L, 0xA28514D9L, 0x6C51133CL,
0x6FD5C7E7L, 0x56E14EC4L, 0x362ABFCEL, 0xDDC6C837L,
0xD79A3234L, 0x92638212L, 0x670EFA8EL, 0x406000E0L },
{ 0x3A39CE37L, 0xD3FAF5CFL, 0xABC27737L, 0x5AC52D1BL,
0x5CB0679EL, 0x4FA33742L, 0xD3822740L, 0x99BC9BBEL,
0xD5118E9DL, 0xBF0F7315L, 0xD62D1C7EL, 0xC700C47BL,
0xB78C1B6BL, 0x21A19045L, 0xB26EB1BEL, 0x6A366EB4L,
0x5748AB2FL, 0xBC946E79L, 0xC6A376D2L, 0x6549C2C8L,
0x530FF8EEL, 0x468DDE7DL, 0xD5730A1DL, 0x4CD04DC6L,
0x2939BBDBL, 0xA9BA4650L, 0xAC9526E8L, 0xBE5EE304L,
0xA1FAD5F0L, 0x6A2D519AL, 0x63EF8CE2L, 0x9A86EE22L,
0xC089C2B8L, 0x43242EF6L, 0xA51E0 3AAL, 0x9CF2D0A4L,
0x83C061BAL, 0x9BE96A4DL, 0x8FE51550L, 0xBA645BD6L,
0x2826A2F9L, 0xA73A3AE1L, 0x4BA99586L, 0xEF5562E9L,
0xC72FEFD3L, 0xF752F7DAL, 0x3F046F69L, 0x77FA0A59L,
0x80E4A915L, 0x87B08601L, 0x9B09E6ADL, 0x3B3EE593L,
0xE990FD5AL, 0x9E34D797L, 0x2CF0B7D9L, 0x022B8B51L,
0x96D5AC3AL, 0x017DA67DL, 0xD1CF3ED6L, 0x7C7D2D28L,
0x1F9F25CFL, 0xADF2B89BL, 0x5AD6B472L, 0x5A88F54CL,
0xE029AC71L, 0xE019A5E6L, 0x47B0ACFDL, 0xED93FA9BL,
0xE8D3C48DL, 0x283B57CCL, 0xF8D56629L, 0x79132E28L,
0x785F0191L, 0xED756055L, 0xF7960E44L, 0xE3D35E8CL,
0x15056DD4L, 0x88F46DBAL, 0x03A16125L, 0x0564F0BDL,
0xC3EB9E15L, 0x3C9057A2L, 0x97271AECL, 0xA93A072AL,
0x1B3F6D9BL, 0x1E6321F5L, 0xF59C66FBL, 0x26DCF319L,
0x7533D928L, 0xB155FDF5L, 0x03563482L, 0x8ABA3CBBL,
0x28517711L, 0xC20AD9F8L, 0xABCC5167L, 0xCCAD925FL,
0x4DE81751L, 0x3830DC8EL, 0x379D5862L, 0x9320F991L,
0xEA7A90C2L, 0xFB3E7BCEL, 0x5121CE64L, 0x774FBE32L,
0xA8B6E37EL, 0xC3293D46L, 0x48DE5369L, 0x6413E680L,
0xA2AE0810L, 0xDD6DB224L, 0x69852DFDL, 0x09072166L,
0xB39A460AL, 0x6445C0DDL, 0x586CDECFL, 0x1C20C8AEL,
0x5BBEF7DDL, 0x1B588D40L, 0xCCD2017FL, 0x6BB4E3BBL,
0xDDA26A7EL, 0x3A59FF45L, 0x3E350A44L, 0xBCB4CDD5L,
0x72EACEA8L, 0xFA6484BBL, 0x8D6612AEL, 0xBF3C6F47L,
0xD29BE463L, 0x542F5D9EL, 0xAEC2771BL, 0xF64E6370L,
0x740E0D8DL, 0xE75B1357L, 0xF8721671L, 0xAF537D5DL,
0x4040CB08L, 0x4EB4E2CCL, 0x34D2466AL, 0x0115AF84L,
0xE1B00428L, 0x95983A1DL, 0x06B89FB4L, 0xCE6EA048L,
0x6F3F3B82L, 0x3520AB82L, 0x011A1D4BL, 0x277227F8L,
0x611560B1L, 0xE7933FDCL, 0xBB3A792BL, 0x344525BDL,
0xA08839E1L, 0x51CE794BL, 0x2F32C9B7L, 0xA01FBAC9L,
0xE01CC87EL, 0xBCC7D1F6L, 0xCF0111C3L, 0xA1E8AAC7L,
0x1A908749L, 0xD44FBD9AL, 0xD0DADECBL, 0xD50ADA38L,
0x0339C32AL, 0xC6913667L, 0x8DF9317CL, 0xE0B12B4FL,
0xF79E59B7L, 0x43F5BB3AL, 0xF2D519FFL, 0x27D9459CL,
0xBF97222CL, 0x15E6FC2AL, 0x0F91FC71L, 0x9B941525L,
0xFAE59361L, 0xCEB69CEBL, 0xC2A86459L, 0x12BAA8D1L,
0xB6C1075EL, 0xE3056A0CL, 0x10D25065L, 0xCB03A442L,
0xE0EC6E0EL, 0x1698DB3BL, 0x4C98A0BEL, 0x3278E964L,
0x9F1F9532L, 0xE0D392DFL, 0xD3A0342BL, 0x8971F21EL,
0x1B0A7441L, 0x4BA3348CL, 0xC5BE7120L, 0xC37632D8L,
0xDF359F8DL, 0x9B992F2EL, 0xE60B6F47L, 0x0FE3F11DL,
0xE54CDA54L, 0x1EDAD891L, 0xCE6279CFL, 0xCD3E7E6FL,
0x1618B166L, 0xFD2C1D05L, 0x848FD2C5L, 0xF6FB2299L,
0xF523F357L, 0xA6327623L, 0x93A83531L, 0x56CCCD02L,
0xACF08162L, 0x5A75EBB5L, 0x6E163697L, 0x88D273CCL,
0xDE966292L, 0x81B949D0L, 0x4C50901BL, 0x71C65614L,
0xE6C6C7BDL, 0x327A140AL, 0x45E1D006L, 0xC3F27B9AL,
0xC9AA53FDL, 0x62A80F00L, 0xBB25BFE2L, 0x35BDD2F6L,
0x71126905L, 0xB2040222L, 0xB6CBCF7CL, 0xCD769C2BL,
0x53113EC0L, 0x1640E3D3L, 0x38ABBD60L, 0x2547ADF0L,
0xBA38209CL, 0xF746CE76L, 0x77AFA1C5L, 0x20756060L,
0x85CBFE4EL, 0x8AE88DD8L, 0x7AAAF9B0L, 0x4CF9AA7EL,
0x1948C25CL, 0x02FB8A8CL, 0x01C36AE4L, 0xD6EBE1F9L,
0x90D4F869L, 0xA65CDEA0L, 0x3F09252DL, 0xC208E69FL,
0xB74E6132L, 0xCE77E25BL, 0x578FDFE3L, 0x3AC372E6L }
};


typedef struct {
  unsigned long P[16 + 2];
  unsigned long S[4][256];
} BLOWFISH;

static unsigned long F(BLOWFISH *ctx, unsigned long x) {
    unsigned short a, b, c, d;
    unsigned long  y;

    d = (unsigned short)(x & 0xFF); //mascara
    x >>= 8;
    c = (unsigned short)(x & 0xFF);
    x >>= 8;
    b = (unsigned short)(x & 0xFF);
    x >>= 8;
    a = (unsigned short)(x & 0xFF);
    y = ctx->S[0][a] + ctx->S[1][b];
    y = y ^ ctx->S[2][c];
    y = y + ctx->S[3][d];

    return y;
}

/*Recebe um texto plano de 64 bits dividido em 2 de 32 bits xl
e xr*/
void Cifra_Blowfish(BLOWFISH *ctx, unsigned long *xl,
unsigned long *xr){
  unsigned long  Xl;
  unsigned long  Xr;
  unsigned long  temp;
  short       i;
```

```
  Xl = *xl;
  Xr = *xr;
  for (i = 0; i < N; ++i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;
    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }
  temp = Xl;
  Xl = Xr;
  Xr = temp;
  Xr = Xr ^ ctx->P[N];
  Xl = Xl ^ ctx->P[N + 1];
  *xl = Xl;
  *xr = Xr;
}

void Decifra_Blowfish(BLOWFISH *ctx, unsigned long *xl,
unsigned long *xr){
  unsigned long  Xl;
  unsigned long  Xr;
  unsigned long  temp;
  short      i;

  Xl = *xl;
  Xr = *xr;
  for (i = N + 1; i > 1; --i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;
    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }
  temp = Xl;
  Xl = Xr;
  Xr = temp;
  Xr = Xr ^ ctx->P[1];
  Xl = Xl ^ ctx->P[0];
  *xl = Xl;
  *xr = Xr;
}

//criação das subchaves
void Inicia_Blowfish(BLOWFISH *ctx, unsigned char *key,
int keyLen) {
  int i, j, k;
  unsigned long data, datal, datar;

  //inicializa S-Boxes
  for (i = 0; i < 4; i++) {
    for (j = 0; j < 256; j++)
      ctx->S[i][j] = ORIG_S[i][j];
  }

  j = 0;

  for (i = 0; i < N + 2; ++i) {
    data = 0x00000000;
    //pega os 32 primeiros bits da chave
    for (k = 0; k < 4; ++k) {
      data = (data << 8) | key[j];
      j = j + 1;
      if (j >= keyLen)
        j = 0;
    }
    ctx->P[i] = ORIG_P[i] ^ data;
  }

  datal = 0x00000000; //32 bits

  datar = 0x00000000; //32 bits

  for (i = 0; i < N + 2; i += 2) {
    Cifra_Blowfish(ctx, &datal, &datar);
    ctx->P[i] = datal;
    ctx->P[i + 1] = datar;
  }

   for (i = 0; i < 4; ++i) {
    for (j = 0; j < 256; j += 2) {
      Cifra_Blowfish(ctx, &datal, &datar);
      ctx->S[i][j] = datal;
      ctx->S[i][j + 1] = datar;
    }
  }

}

int main( int argc, char *argv[] )
{
  BLOWFISH ctx;
  unsigned long l, r;

  FILE *arquivo;
  FILE *cifrado;
  FILE *decifrado;
  char nome[30];
  char chave[30];
  long int inicio, fim;

    printf("Digite a chave: ");
    gets(chave);
    Inicia_Blowfish(&ctx, (unsigned char*)chave,
strlen(chave));
    printf("Digite o nome do arquivo a ser encriptado: ");
    gets(nome);

    arquivo = fopen(nome, "rb"); //arquivo = fopen(argv[1],
"rb");
    if(arquivo == NULL) {
      printf("\nErro ao abrir arquivo! <%s>\n", nome);
      system("PAUSE");
      exit(1);
    }

    cifrado = fopen("cifrado", "wb");
    if(cifrado == NULL) {
      printf("\nErro ao abrir arquivo! <cifrado1>");
      system("PAUSE");
      exit(1);
    }

    inicio = clock();
    while(!feof(arquivo)) {
      //leitura de dois blocos de 32 bits - plain text
      fread(&l, sizeof(unsigned long), 1, arquivo);
      fread(&r, sizeof(unsigned long), 1, arquivo);
      //Chamada da função de cifragem
      Cifra_Blowfish(&ctx, &l, &r);
      //escrita de dois blocos de 32 bits - cipher text
      fwrite(&l, sizeof(unsigned long), 1, cifrado);
      fwrite(&r, sizeof(unsigned long), 1, cifrado);
    }

    fim = clock();
    printf("\nTempo de encriptacao: %ld", fim - inicio);
    fclose(arquivo);
    fclose(cifrado);

    cifrado = fopen("cifrado", "rb");
```

```
    if(cifrado == NULL) {
     printf("\nErro ao abrir arquivo! <cifrado>");
     system("PAUSE");
     exit(1);
    }

    decifrado = fopen("decifrado", "wb");
    if(decifrado == NULL) {
     printf("\nErro ao abrir arquivo! <decifrado>");
     system("PAUSE");
     exit(1);
    }

    inicio = clock();
    while(!feof(cifrado)) {
}
      fread(&l, sizeof(unsigned long), 1, cifrado);
      fread(&r, sizeof(unsigned long), 1, cifrado);
      //Chamada da função de decifragem
      Decifra_Blowfish(&ctx, &l, &r);
      fwrite(&l, sizeof(unsigned long), 1, decifrado);
      fwrite(&r, sizeof(unsigned long), 1, decifrado);
     }

     fim = clock();
     printf("\nTempo de decriptacao: %ld \n", fim - inicio);
     fclose(cifrado);
     fclose(decifrado);

    system("PAUSE");
    return 0;
```

Código C Chave Estática 32 bits

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 16

static const unsigned long ORIG_P[16 + 2] = {
  0x710c58a7L, 0x7c2e7c45L, 0x3f24416fL, 0x306529abL,
  0x1f9dd3fdL, 0x59883516L, 0x588a615aL,  0xe92a123dL,
  0x74275d9eL, 0xf7d34577L, 0x720c8f60L, 0xf085e4caL,
  0xec697049L, 0xca4419abL, 0xcc08c212L, 0x732f707fL,
  0x2038daf0L, 0xec448a5bL,
};
//S-Boxes geradas a partir da chave fixa 0123
static const unsigned long ORIG_S[4][256] = {
  { 0x2563ef98L,0x4d9ee216L,
    0xb6003691L,0xd743587eL,0x6c556e23L,0xfe5a01e7L,
    0x94fd298L,0x5ada3678L,0x48dc8203L,0x688c884L,
    0xefe74433L,0x5739b6d4L,0x4e55f038L,0x5b112d6aL,
    0x73818d9bL,0x24f40946L,0x39927ee0L,0x98588592L,
    0x5bbe064dL,0xf6d7df53L,0xf7d00343L,0xab5924L,
    0x543673f4L,0x98952205L,0xad2efddbL,0x86d3c8d6L,
    0xe5b3c0cdL,0x2577b2a6L,0xcc048a34L,0x6c73f006L,
    0xff216c39L,0xfc357c1eL,0xe0ac49d6L,0x6562f9a9L,
    0xa24605cfL,0xe2507bd7L,0xd8787e72L,0x5fe5b04bL,
    0xdf203a63L,0x399ff53dL,0xf45888d6L,0x47c9565aL,
    0x6d972b34L,0x3cf1dc1eL,0xa77de156L,0x547c0033L,
    0x1f652364L,0x45c1cdabL,0xf7716dadL,0x343dd0ecL,
    0xe2b25d93L,0x7a9f71d6L,0x90fcfd0L,0xa7fef7ebL,
    0x7641a772L,0xff4de4e6L,0x66766321L,0x72ff8f4fL,
    0xf3546786L,0xd9a5d12bL,0x99f0e138L,0x8b71b680L,
    0x18d1d64L,0xbfc82e03L,0xd08efd74L,0x5a064aL,
    0x265e2c4L,0x6a29fef9L,0x91c1968fL,0xb5602d76L,
    0x1aa425b5L,0x5a2e515dL,0x4095b414L,0x566c3d1eL,
    0x236221bdL,0xe5dbe10cL,0x23ea8591L,0x49efdf0bL,
    0xf7813fb0L,0x1c86e8b4L,0x6383c15bL,0x3e3cce20L,
    0x54d07067L,0xbe4c151dL,0x6a30653dL,0xfb45beb1L,
    0x6741ca45L,0xd7e1b18L,0x77b73ba9L,0xb8f048f0L,
    0x678676bfL,0x1a4a4650L,0x8b2b5963L,0x9b9dbbebL,
    0x4c5cad64L,0x7d6e6318L,0x75164606L,0x2338a791L,
    0xf056dc6aL,0x744584dL,0xe7562d2eL,0x3120aa7L,
    0xe84917a6L,0xf2ac5610L,0x99820eaaL,0xf2fefcc9L,
    0xae005c51L,0x53387dacL,0x577edee5L,0x36763e11L,
    0xe08f195bL,0xb6c91ef1L,0x2c7299b1L,0xa118ead5L,
    0xb9ac69f1L,0x6d7eac5dL,0x5a7a4e18L,0xd58e662cL,
    0x8741d041L,0x85c251d0L,0x1715c829L,0x229bc5d1L,
    0x314d31ebL,0xe441c239L,0xf645ad9cL,0xcd1c6e32L,
    0x7272f8fcL,0x3ca09e82L,0x7da0c78aL,0x5532c440L,
    0x16cd9a80L,0xeb317d36L,0xb9f1ccc9L,0x796485fbL,
    0x1f925a82L,0x1718e380L,0x5b5eebefL,0x72411723L,
    0x1f5c0ca4L,0xc54582faL,0xd8a26990L,0xc669af9eL,
    0x1599c3feL,0xc59f1b08L,0xdf65702aL,0x4e0453b4L,
    0xcc7c8cbcL,0xf086c86bL,0x67c10136 L,0x868b5c63L,
    0x13edc457L,0xcbf94ecfL,0x5509db5eL,0x8ceafb95L,
    0xbd8ed38L,0x243dff80L,0xd050df08L,0xae6df7f6L,
    0xb43daa49L,0x65f8bf8bL,0xf27ca452L,0xa78fbee2L,
    0xc506605fL,0x541790c0L,0xa87ba0dbL,0x7f57c98dL,
    0x639a0b66L,0xdfd0f0b1L,0x9d506d69L,0x38e05852L,
    0xc6371ed8L,0x626a466fL,0x892bb592L,0x61a420faL,
    0xca43db5bL,0x2512bdccL,0x687eff5aL,0x60b81f41L,
    0x88cb0ccL,0xa0090888L,0xcd6a0d95L,0x1579efb7L,
    0x70210cdcL,0x2524d71dL,0x3ff18db9L,0xffc43e34L,
    0x977515a8L,0xea5d971L,0xfd0527dbL,0x89751c30L,
    0x4185c420L,0x12a207abL,0x5e5bfe70L,0x8a05cca7L,
    0x6e19b5a5L,0xb449dbbcL,0x899f320fL,0x19f00040L,
    0xcf6b9d93L,0xde840554L,0xbecdfc25L,0xbec56c53L,
    0x25f3e298L,0xe0d35e00L,0x4337f116L,0x7ba2e28cL,
    0x6bc404b2L,0xe20755fcL,0x3ba37fa6L,0x156fade5L,
    0x512c46fL,0xf0492f0dL,0x966e5968L,0xe095992eL,
    0xdeef240cL,0x68430c2dL,0xea61a460L,0x93c66934L,
    0x7dc9d90cL,0x8208d01eL,0x44cfec46L,0x9c9fb7c3L,
    0xebf147feL,0x60802f83L,0x8a637604L,0xde7fb976L,
    0x652ec336L,0x33f43477L,0x19e7361fL,0x2f6fca1fL,
    0x7a9ebc74L,0xb2c7e414L,0x664424bcL,0x9590641fL,
    0x5fa68f5cL,0xc1b8067eL,0x4be31b2cL,0xb489f46cL,
    0xc7c7b1d7L,0x2cb007f9L,0x50cc7dd2L,0xeae2dc6cL,
    0xb5ff84e6L,0x9c9c5231L,0xea5472ecL,0xf3dbe0a4L,
    0x3d02174L,0xc115627eL,0x60f2054dL,0x8304e4fcL,
    0xcfedca27L,0x64010003L,0x21cf5e1dL,0x7b79cfcbL,
    0x1551a100L,0xfbdee2a9L },
    { 0x78fc09a0L,0xa727b2eaL,
    0x272ea1f7L,0x274fdf99L,0xc6f0b1d9L,0x56b4d41dL,
    0x4828fd61L,0x4dfe24eeL,0x5c626d5aL,0xde68e1c2L,
    0x119fd658L,0xf3f48eeL,0x5152994cL,0x6dfc67bdL,
    0xcea534d5L,0x981facbL,0x1868c88eL,0x353896b3L,
    0x7ff22b13L,0x84dc9f43L,0x2663c951L,0x92ac59e4L,
    0x3b24190aL,0xa1d7636L,0x39e381c5L,0x394dc1a1L,
    0x7b8c1fbaL,0xc00a0626L,0x4f1bc88eL,0x93660ad9L,
    0xdd57bc0dL,0xadd1214dL,0xa3a4bdfL,0x843fc8beL,
    0x6b30882dL,0x6214a9c2L,0x5c817a2eL,0x25f43c3L,
    0x664f7632L,0xfceb3ce1L,0xda1fb3b3L,0xaa6f3e09L,
    0xf46d7c78L,0x401e0084L,0x5814a348L,0x55474d25L,
    0xb9910aedL,0xb1242417L,0xb05adaeaL,0x339693e9L,
    0x5e3bc946L,0x96a8fbd9L,0x9525aa5bL,0xd401718L,
    0x3d536838L,0x2fe06715L,0xeed26717L,0x75c09f81L,
    0x7e266bf3L,0x7deba5aL,0x2dbc0c0bL,0x1f0fe4cL,
    0xb8246d57L,0x783fd8f6L,0x2defd5d6L,0x9c5e45d6L,
    0x14124357L,0x8a44d360L,0xbacbaa5dL,0x27c91a7fL,
    0xab9a35b7L,0x186c3770L,0x565829f0L,0x11fb483fL,
    0xb9f0951fL,0x7cca442aL,0x97bbd48bL,0x73445c0fL,
    0xf4600b6dL,0x3aac1393L,0x1bd7ecf1L,0xf9885026L,
    0x433e6413L,0xd44c76b6L,0x57910544L,0xe6e75a51L,
    0x198ef1cfL,0xccba163aL,0x93d1606bL,0xc27f4201L,
    0xcb66ef57L,0x9467872cL,0xdfb603bL,0x7adcd07fL,
    0x9feb6470L,0x6a78f4c4L,0x5a9e905bL,0xad312098L,
    0x1d3397eeL,0x3fe7f0faL,0xe1445eecL,0x93213904L,
    0xf334b173L,0x293c08d9L,0x3a15a30L,0xc14beca3L,
    0x69ec2298L,0xf32e28f9L,0x946d4161L,0x117e8014L,
    0xe891e45L,0x62f95417L,0xadca9a6fL,0xdfcf173eL,
    0x6e30fc5eL,0x10739f0aL,0x84be8a38L,0xeb38f603L,
    0x19c47038L,0x7ea3c8d8L,0xaa1e77d1L,0xc5fb976fL,
    0x9b6271e8L,0x22f0034L,0x1d6b17d5L,0xcaf493ceL,
    0x8b733869L,0xc66014edL,0xd1d8dca3L,0x644fb266L,
    0x57133c0dL,0xb209de4dL,0x345a90a2L,0x635c9b4eL,
    0x77656423L,0x70c8eacfL,0xe50a9e1L,0x87b99988L,
    0x1c9373bcL,0xaf6a2d6fL,0xf7447c4fL,0x126a8981L,
    0x92ebddf2L,0xedb5da1dL,0xae62e4c8L,0xc2dfa6d4L,
    0xe07f1bc0L,0xecb633d7L,0xde028e22L,0xc7b2382eL,
    0x7402d4a4L,0x2cba6587L,0xd60d5264L,0x18c6e3fbL,
    0xd0b78309L,0x63c63966L,0xd4e76636L,0xaa75c099L,
    0x21310edaL,0xdbd0ea36L,0x3359d9f8L,0x4d47cd0aL,
    0xab3552dbL,0xcd319dbeL,0xe6313100L,0xe9732d85L,
    0x61fbf6cL,0xdad8ce74L,0xad2c6e3aL,0x1ab638deL,
    0x63be8a61L,0x13e6a5d3L,0xf4f59b7aL,0xfdd6f45cL,
    0xd15da343L,0x3b29fab6L,0x1a9e5cd7L,0x1cf73e21L,
    0xa7961661L,0x33a14d01L,0x3c4af42eL,0xe98e9363L,
    0x94b940e9L,0xe64066cfL,0xeef87472L,0xc0801abfL,
    0xb8631e31L,0x6792264fL,0x1c476801L,0x9507273fL,
    0xdb1d490aL,0x86d60213L,0x577bb440L,0x91ec0ba5L,
    0x2df04e94L,0x2bdee7c0L,0xd008b6e2L,0x83a3c904L,
    0x9c6f1698L,0x6a09a7a5L,0x352a83f9L,0x10544dd5L,
    0x1cc302ebL,0x3e25907eL,0x8789fcbL,0x6d45287cL,
    0xc380e7a2L,0x94aec9faL,0x666b6dccL,0x59018325L,
    0x5d3053f4L,0x96e1be26L,0xe228bf63L,0xe20462e0L,
    0xd51b76b7L,0x19362be2L,0xd8ece7d9L,0xfd52f724L,
```

```
0xebd37f7cL,0x754d92e0L,0x851ea15bL,0x5987a099L,
0x6333f757L,0xc83abbf3L,0x3422357aL,0xc7897290L,
0xcd2240c4L,0x3e08364bL,0x27175e7eL,0x479a2d03L,
0x98926b95L,0xfab2bcd9L,0xedef42f7L,0xc9327da5L,
0x320ffeabL,0xfee530c5L,0x82a6841aL,0xd7c2cefaL,
0xc8e79d23L,0x8d5e64ccL,0x9e86160eL,0xea121852L,
0x800910c1L,0x29233b88L,0x110e721fL,0x63294562L,
0xfee39b36L,0x118e9548L,0x18b3b9abL,0xbdb0ea9fL,
0xebd38ac2L,0x2e4c3e7dL,0x6986b0c0L,0x7d21c3e4L,
0x488c7d00L,0x1172714aL },
{ 0x3fc5a6ffL,0x8709717aL,
0x1aa40de6L,0x7d22cf2L,0xc728e2c1L,0xb0fe456cL,
0x7e4e559cL,0x5dfeafa9L,0x959a4a0cL,0xc6c2d485L,
0x9f211443L,0x98c9fe62L,0x65e972e0L,0xe962d94bL,
0xe2eb808aL,0x2d591b04L,0x5c831cdbL,0xabbb666fL,
0xb074afd9L,0x62e8120bL,0x6cfd1132L,0x421d2d20L,
0x1d3df5a0L,0xb76fd739L,0x8f7ad4b6L,0x38d04183L,
0xe4029c92L,0xa6ed64c1L,0xb78e408fL,0x65e2391dL,
0x683d915aL,0xe0e51bc2L,0x450740b0L,0xff3d7197L,
0x67162c9L,0x26b39225L,0xece8c6d5L,0xeb760d33L,
0x5ed2f574L,0xfa827164L,0xebced2c9L,0xe517e279L,
0x5b277887L,0x5882a9deL,0x5fe821b9L,0xeefe37f2L,
0x776dc21L,0x235440cbL,0x979a6be1L,0x14fbddc8L,
0x58363ad0L,0xd7584902L,0xd417f4f5L,0x39cc7505L,
0x19314dc1L,0xe2549196L,0xfd10b09bL,0xcfdcb53cL,
0x5a7947a5L,0xd9ab5b83L,0x3a198831L,0xf74303eeL,
0xd8e4b1e5L,0x224808ebL,0x6f010103L,0xe0aa7908L,
0x8ab70f5cL,0x8dc6801bL,0x1f323451L,0xb6d2d280L,
0xa58eaab0L,0xe589ebadL,0x66739183L,0x80efd586L,
0x78a419e1L,0xd184d00L,0x18a5c93eL,0xdbb62973L,
0x9e2086e9L,0x10167894L,0xe91e44d3L,0xdc2c968bL,
0xbfa57f5L,0xfce6e336L,0x3789e0aaL,0x9a91b93cL,
0x503deaeeL,0x565e279fL,0x4002b47dL,0x1639095eL,
0xf563bc0L,0x5f2a29c8L,0xf15ed48fL,0xa7e8a7f4L,
0x5b285896L,0x14b54dd1L,0x5287f7b1L,0xa455d06dL,
0xaf703bf7L,0x8ab0f29fL,0x6e28352dL,0xcc4a52cfL,
0xaaebaa46L,0xc8b0aaeeL,0x78646c98L,0xfbda2429L,
0x1408f5c6L,0x15921b45L,0x447c55f1L,0xb6c35a55L,
0x7cc9a145L,0xa7f0a308L,0x64 9fe7cL,0x41a61a46L,
0x1cff45ddL,0x64739ab6L,0x18be80dbL,0x11882389L,
0xd414599aL,0x41afdcd6L,0xcde5c575L,0x81f6a521L,
0x1631485cL,0xeed828afL,0x86504c29L,0xcb6b1f07L,
0x6bab8dbaL,0xdc34da4L,0x8a9da3acL,0xc85dd56dL,
0xd36cc745L,0x8047e985L,0xb4dfc0b9L,0xd55dd328L,
0xbc7c3538L,0xadef84adL,0x7e31fbd7L,0xdaa9128L,
0x5d106340L,0x882f225fL,0x4f047190L,0xd9e800b8L,
0x55c1de1fL,0x631c30a3L,0x608b99a8L,0xf00a08daL,
0xe5248750L,0xcf19e8aeL,0x26f580a6L,0xb3e2cd22L,
0xe70786dbL,0xecef31d6L,0xf6f06817L,0xae88bc2fL,
0xa3d6412L,0x3022c10eL,0x296262d7L,0x759b8136L,
0x857d4678L,0xfc7dbc8eL,0x78a83591L,0x88cc20e1L,
0x3bd6412fL,0xcadaf7bbL,0xf946c057L,0x5a00353dL,
0x185e4999L,0x92 dc6caaL,0x96843430L,0x56e17059L,
0x748d7b82L,0x7b1d4877L,0xbd3eb7bbL,0x1f87af6dL,
0xc87a9d7eL,0x9db2b51fL,0xf3e3da2aL,0x18d25e51L,
0xf5558798L,0xbb40d10fL,0x45c6db2fL,0x7aea9e18L,
0x562e2ce6L,0x6ab6638fL,0xbdacc235L,0xf29494dcL,
0xbd5bed1aL,0x6bcb7e33L,0x21bb8190L,0x92dc7eecL,
0xb8b7f9f9L,0x9be7980L,0x2a689a90L,0xcb81623eL,
0xac92dccL,0x415a8c3cL,0xe2c3284dL,0x18b3d10bL,
0xac6225f6L,0xb8aa30c6L,0x140f6f89L,0x2061dffbL,
0xc2f6db19L,0x2de02cedL,0x5b8127bbL,0x4b1f91f4L,
0x3a119b77L,0x66df5403L,0x13228fbbL,0x48885cdbL,
0xbc705039L,0x797f808eL,0x3c4f916fL,0x2518cdf9L,
0x7706ced5L,0x8ecbf034L,0x119bda3aL,0x755357ecL,
0x2f764ff0L,0xe0fd288aL,0x710ac39L,0xf5be39afL,
0xda5094cbL,0x179027a2L,0x40a03c23L,0x33f98ea0L,
0x21e4b3a1L,0xfd2539b0L,0x26c39dfcL,0x6f4ac98aL,
0xb89c80f3L,0x66112140L,0x9892cebaL,0x16b24fcdL,
0x677ab1e8L,0x272b7985L,0x3243b32fL,0xcad52353L,
0x4709d7aeL,0xb652c5ddL,0x760fa9dL,0x1b63c60bL,
0x7ea96659L,0x7084c536L,0xd7ce61f6L,0xee985b53L,
0x65f4af5eL,0x71ae39b6L,0xe9137692L,0x9cc9e188L,
0x3b4a00e6L,0xb3805defL,0xb4d12050L,0x75f44444L,
0xb4fef803L,0x4098832eL },
{ 0xd3f1ea8fL,0xf63425a4L,
0x89faee01L,0xf38dfc2dL,0x1748b5aaL,0xce3dbc44L,
0xfbeb7a26L,0xe958e9aaL,0x235910c3L,0x1eb5416cL,
0x7dc96656L,0x81ff1015L,0x6720a991L,0x54649476L,
0x153f9843L,0xe4c4bd5dL,0xccd633edL,0x26003c6cL,
0x447c31b3L,0x3a7cb644L,0xb90c852dL,0x82edc3baL,
0xd72c9102L,0xd3dba126L,0x2ee92b5aL,0x970871f4L,
0xce9037edL,0xf82896f6L,0xda81c500L,0x55f249edL,
0xcfbb35ceL,0xd2a2eec9L,0x5d05cadeL,0x81916975L,
0xe6657196L,0x2d7d4ee0L,0x27a15cfbL,0xf42377edL,
0xa136e0a9L,0x9f639ba5L,0x3520920aL,0xc397f840L,
0xfc03caf2L,0x4807565dL,0x4249802L,0x5209b531L,
0xc8897645L,0x483397a2L,0xdda49db6L,0x2f328fedL,
0x6dee5becL,0x44ebbafaL,0xc3211a35L,0xf2517606L,
0xeaf66d90L,0x7ac55108L,0x29bfe345L,0x399 2d1cL,
0xc3268e1aL,0x745c8a33L,0xf1ea8d0cL,0x454bfcf6L,
0x8ca91ddeL,0x8501da1L,0x30677283L,0x4b25d50fL,
0xf03b99baL,0xdacb0db6L,0xea5f377L,0x5b16b675L,
0x93678590L,0x6240eb7eL,0x9f331db2L,0x5836387dL,
0xb2ce6d8L,0x6f50ce7eL,0xdd2d51d4L,0xd162590bL,
0x1cd11642L,0x644dccc4L,0x7d9e1dbaL,0x6256a955L,
0xd61e2eeeL,0x2b5270aeL,0x34265a1cL,0x81b334a5L,
0x97e1f4beL,0xe1aa51e2L,0x6910f60fL,0x3c257895L,
0xb9d2419eL,0xdc0f230eL,0x84c2f0efL,0xd27b5b0L,
0xf37b7b0L,0x9c4e76dcL,0x1c5881d8L,0x461f075dL,
0xff1d213eL,0x35c8fb54L,0xc483f5b4L,0xc3b020fbL,
0x27f498f2L,0x60b0af8L,0xe38aaccL,0x69ad04e9L,
0xe021ee83L,0xe9cf0dbfL,0xfead1313L,0x1d38e4b5L,
0x9d27efffL,0x8877c66eL,0x9d8ea2b7L,0x6a850c27L,
0x105a9ab7L,0x78c20510L,0xc4391518L,0x36f9b7d3L,
0xece314f7L,0x29acf203L,0x12aff291L,0xe6731871L,
0x35f671beL,0xae93de54L,0xc7d2936L,0x9c94b098L,
0x73e76b7eL,0x73d953c5L,0xec23ecbfL,0x54f3a3f7L,
0x9a2922aaL,0x22078d81L,0xa94d69dbL,0xb2bed43bL,
0x5cc0ea3L,0xf76a5a31L,0x485cb16L,0xab94abf5L,
0xa466bfbL,0x6477e764L,0x3f4fd932L,0xb82068a8L,
0x4276192cL,0x4309af7L,0x84768938L,0xf71fecc2L,
0x3fbf30f3L,0x9f8d609aL,0x663261d3L,0xf8d9ff9dL,
0xef5c7c6bL,0x2268aad0L,0xa667e099L,0x69d44ee0L,
0x1e3f7de0L,0x2716707dL,0x49fe19d7L,0xc8403212L,
0x443a32edL,0xb92d7ce4L,0x7c738bfcL,0x28a18349L,
0x5b1d4c6aL,0xe0fd3fccL,0xbf695c2aL,0xa97a15a6L,
0x6d2db423L,0x99d6b1e2L,0xafba5204L,0xac075bccL,
0xa87c134dL,0x7aab0feL,0xb0d399f7L,0xbe5cbbd8L,
0x68c0f268L,0xfae5aed6L,0x1b197192L,0xf273c537L,
0x11ea01a1L,0x464faddaL,0xa3ff62abL,0xcb45c872L,
0x9fd5f8d1L,0x7d951d3fL,0xd7fad15eL,0xab6d3cb4L,
0x4746e5a7L,0x83a067d3L,0x2a73ad06L,0x4a38f00eL,
0x1747a182L,0x32a09244L,0x2dcf304fL,0x17544f4eL,
0xa638b6dbL,0xa8a5153cL,0x4103b568L,0x9c5c3043L,
0xbbb63b43L,0xb12fcb8eL,0x51e804c1L,0x390b7d2bL,
0x2e1fd144L,0x5a47e956L,0xabb82541L,0xd399092fL,
0x814bb184L,0x9990eedfL,0x1d2ef552L,0x91c9160aL,
0x668e6c07L,0x4cb0f201L,0x6debb729L,0x5fd61718L,
0xa9f955beL,0x30deac7aL,0x2c204d09L,0x8fb916d4L,
0x53e522c3L,0x785ad77cL,0x3d138642L,0x5eafff98L,
0xe393151L,0x9c20533aL,0x18b5c5adL,0x87c2ef56L,
0x55c58b95L,0x65d34afaL,0x804232d8L,0xaa47471cL,
0x6d749fd1L,0xb035af9L,0xb6ea908L,0x266c93daL,
0x40b91a72L,0x10cb42e1L,0x41cc1b86L,0x79b447c3L,
0x378848d0L,0xdde54aa7L,0xd0fd342cL,0x4d44a1dfL,
0xdfec50beL,0xe943a38aL,0xbfcbe4a5L,0xd2e2f64cL,
0xbb0a24dL,0x1808d539L,0x71be2908L,0x2a53bebL,
0x333f503bL,0x38d7bf08L,0x2418ceb3L,0x21cd89efL,
0xc3608d1L,0x65b9c57bL } };
```

```
typedef struct {
  unsigned long P[16 + 2];
  unsigned long S[4][256];
} BLOWFISH;


static unsigned long F(BLOWFISH *ctx, unsigned long x) {
   unsigned short a, b, c, d;
   unsigned long  y;
  d = (unsigned short)(x & 0xFF);   x >>= 8;
   c = (unsigned short)(x & 0xFF);
   x >>= 8;
   b = (unsigned short)(x & 0xFF);
   x >>= 8;
   a = (unsigned short)(x & 0xFF);
   y = ctx->S[0][a] + ctx->S[1][b];
   y = y ^ ctx->S[2][c];
   y = y + ctx->S[3][d];
   return y;
}

void Cifra_Blowfish(BLOWFISH *ctx, unsigned long *xl,
unsigned long *xr){
  unsigned long  Xl;
  unsigned long  Xr;
  unsigned long  temp;
  short i;

  Xl = *xl;
  Xr = *xr;
  for (i = 0; i < N; ++i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;

    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }

  temp = Xl;
  Xl = Xr;
  Xr = temp;
  Xr = Xr ^ ctx->P[N];   //P[16]
  Xl = Xl ^ ctx->P[N + 1];  //P[17]
  *xl = Xl;
  *xr = Xr;
}

void Decifra_Blowfish(BLOWFISH *ctx, unsigned long *xl,
unsigned long *xr){
  unsigned long  Xl;
  unsigned long  Xr;
  unsigned long  temp;
  short       i;

  Xl = *xl;
  Xr = *xr;
  for (i = N + 1; i > 1; --i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;
    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }
  temp = Xl;
  Xl = Xr;
  Xr = temp;
  Xr = Xr ^ ctx->P[1];
  Xl = Xl ^ ctx->P[0];
  *xl = Xl;
  *xr = Xr;
}

void Inicia_Blowfish(BLOWFISH *ctx) {
  int i, j, k;
  unsigned long data, datal, datar;
  for (i = 0; i < 4; i++) {
    for (j = 0; j < 256; j++)
      ctx->S[i][j] = ORIG_S[i][j];
  }

  for (i = 0; i < 18; i++ )
      ctx->P[i] = ORIG_P[i];
}


int main( int argc, char *argv[] )
{
  BLOWFISH ctx;
  unsigned long l, r;

  FILE *arquivo;
  FILE *cifrado;
  FILE *decifrado;
  char nome[30];
  char chave[30];
  long int inicio, fim;
  int i;

    Inicia_Blowfish (&ctx);

    printf("Digite o nome do arquivo a ser encriptado: ");
    gets(nome);

   arquivo = fopen(nome, "rb");
   if(arquivo == NULL) {
     printf("\nErro ao abrir arquivo! <%s>\n", nome);
     system("PAUSE");
     exit(1);
    }

   cifrado = fopen("cifrado", "wb");
   if(cifrado == NULL) {
     printf("\nErro ao abrir arquivo! <cifrado1>");
     system("PAUSE");
     exit(1);
    }

   inicio = clock();
   while(!feof(arquivo)) {
     fread(&l, sizeof(unsigned long), 1, arquivo);
     fread(&r, sizeof(unsigned long), 1, arquivo);
     Cifra_Blowfish(&ctx, &l, &r);
     fwrite(&l, sizeof(unsigned long), 1, cifrado);
     fwrite(&r, sizeof(unsigned long), 1, cifrado);
    }

   fim = clock();
   printf("\nTempo de encriptacao: %ld", fim  -  inicio);
   fclose(arquivo);
```

```
fclose(cifrado);

cifrado = fopen("cifrado", "rb");
if(cifrado == NULL) {
  printf("\nErro ao abrir arquivo! <cifrado>");
  system("PAUSE");
  exit(1);
 }

decifrado = fopen("decifrado", "wb");
if(decifrado == NULL) {
  printf("\nErro ao abrir arquivo! <decifrado>");
  system("PAUSE");
  exit(1);
}
inicio = clock();
  while(!feof(cifrado)) {
    fread(&l, sizeof(unsigned long), 1, cifrado);
    fread(&r, sizeof(unsigned long), 1, cifrado);
    Decifra_Blowfish(&ctx, &l, &r);
    fwrite(&l, sizeof(unsigned long), 1, decifrado);
    fwrite(&r, sizeof(unsigned long), 1, decifrado);
   }

   fim = clock();
   printf("\nTempo de decriptacao: %ld \n", fim - inicio);
   fclose(cifrado);
   fclose(decifrado);

  system("PAUSE");
  return 0;
}
```

Código C Chave Estática 128 bits

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 16

static const unsigned long ORIG_P[16 + 2] = {
      0x2f2016b0L, 0xa1a53b53L, 0xc31af3fcL,
      0x85fd0de1L, 0x9c4bec8eL,
      0x162e1e81L, 0x8a011542L, 0xe9620430L,
      0x3b466b49L, 0xd3effe12L,
      0xeabb7803L, 0x4f02d249L, 0x7f1a9667L,
      0x351b0f0cL, 0x92c6fadbL,
      0x125dbbb8L, 0xea2a1bbdL, 0x131257a5L
};

//S-Boxes geradas a partir da chave fixa zxcvasdfqwerzxcv
static const unsigned long ORIG_S[4][256] = {
{ 0xe7414d90L,0x94da48abL,
0x8bbda9f1L,0x97678e80L,0x60d3b952L,0x8565579eL,
0xd8b37ea9L,0xb52cb35eL,0x84a94af3L,0x1b390caeL,
0xa92a8248L,0x887364e0L,0xec154ce2L,0xd6c25491L,
0x66b7d180L,0x6cc30a8dL,0x3e8a95f1L,0xfa218559L,
0x26b2363fL,0xc583e8d5L,0x9dd2ea3dL,0x5f2d2abfL,
0xb15a6bd4L,0xf5db4dbaL,0x278aa5d2L,0x162ae6cbL,
0xdd5d8922L,0x29c4b86cL,0xa00e8886L,0xc236e5a0L,
0xb0710c0aL,0x3b7ecb7L,0xf32a6254L,0x5dd55773L,
0x5cf6ab9dL,0x30d9856L,0x1581a390L,0x96bfea75L,
0x97cc625dL,0x657a203dL,0xfb338695L,0x7defec8eL,
0x9c458a2cL,0xc6871da4L,0xcdf25a06L,0x563da82eL,
0x39bb9cbfL,0x8d789abbL,0x9abde736L,0x3c33c547L,
0x48dea62aL,0x32292037L,0x5c75af6aL,0xb9f8cb81L,
0x3cb90a3L,0x7e504a7bL,0x e773b1d6L,0x9e023df0L,
0x78cda127L,0x8672086dL,0x37f8bbd8L,0x61c31441L,
0x740886b4L,0x257a62e3L,0x880f88d1L,0x1b1877bbL,
0xb7096146L,0x5a2ee9eaL,0x9c9d5bdfL,0x97d79bbfL,
0x7726c5c7L,0xc7e9a00dL,0x2c6a0426L,0x9568a3bdL,
0x176b2209L,0x44c9f648L,0xae1c6e81L,0xbdad1034L,
0x8d1aa7aeL,0x9c594e0fL,0xdc6158feL,0x50ce6e89L,
0x9821a9feL,0xecec2781L,0xc3809842L,0x1f5a2662L,
0x243efbd6L,0x64fa5b81L,0xdc4b7a9bL,0x71316f0cL,
0x6e24a960L,0x57ea8284L,0x6d18d445L,0xfbdb0669L,
0x94608727L,0x20436b08L,0x20f832afL,0x4716382aL,
0x63c8a019L,0x537e818aL,0x6b02699eL,0x272f5383L,
0x2e69e25aL,0x2da7e17aL,0xce13ee08L,0x2a1934feL,
0xef4b28b2L,0xe2616703L,0xb51e45cbL,0x6e970327L,
0x22829c42L,0x77d4c92aL,0x7d219ebeL,0x5803406cL,
0x76b0a722L,0xf035bf0fL,0xf0da98dcL,0x1ebbcf8aL,
0x3e86a5dL ,0xf1257dd9L,0x483f88e2L,0x8130fe5dL,
0xa97c6e28L,0x642f5485L,0x18b0bc76L,0xfee0f518L,
0x5c0be54eL,0x874b423eL,0x8a9e4802L,0xe0550739L,
0x867dd933L,0xd694b05bL,0x75ec46d6L,0x7da6c4e3L,
0x33151080L,0xa7bbfc6eL,0x4fce45c8L,0x8e64390dL,
0xc7535666L,0xce945764L,0x56751edcL,0x76681bc3L,
0x248e42L,0x545fffdfL,0x352f4547L,0x5386aba9L,
0xb76470b3L,0x684cbecdL,0x93d7df1cL,0xa885e732L,
0xfbcab017L,0xa7b0d30bL,0xfd9c7f75L,0xc63b84e1L,
0x125b57fdL,0x62d2786bL,0xb866a16fL,0xa30f2153L,
0xda5a8379L,0xed33d16dL,0xdf56f396L,0x9bb0e3ceL,
0x62b6d32cL,0x969a2b79L,0x3a132d1L,0x5b93878cL,
0xb939e755L,0x58a3e2e3L,0x590bd26eL,0xabc8d4bL,
0x2c8d20c0L,0x136d9a1bL,0x5c2590dL,0xaf31878dL,
0xd0299064L,0x80e05e4cL,0x6ecd3f7eL,0xb145bde4L,
0x13f59178L,0xce07f36fL,0xcb1c08ddL,0x9174eb56L,
0xa8fa2706L,0x4c5e4adeL,0xe1482d64L,0x9e099b48L,
0x1ff50777L,0xb7384fafL,0xd3bfce38L,0x22a9f7a1L,
0x2e9e0d84L,0xcdbce58aL,0x649c0879L,0xdf77b812L,
0xff940c7dL,0x51f2270cL,0x4549e476L,0x595a7eb9L,
0x269c3f4L,0xb9027da9L,0x473796aaL,0x38407cfeL,
0x433a59a0L, 0xa60d348cL,0xb1cab934L,0x8ce5ce27L,
0xef7cba51L,0x56468566L,0x95b1cf36L,0x5b04c604L,
0x66e5a94dL,0x3728448eL,0xda0df75eL,0x7c2f5939L,
0x679a33ceL,0x3861d190L,0x2cb69976L,0xf7ba8f2bL,
0xb2b21cffL,0x96644c3cL,0xc87984dbL,0x1195dbc6L,
0xbd4e9ccaL,0x394ca61aL ,0x5840d2f0L,0xddf436daL,
0x1e6168cL,0x84dde74eL,0xa5afdccdL,0xabc92bc1L,
0x26a4f2faL,0xeed4033eL,0x4edcf8f5L,0xb64d8586L,
0x60f07470L,0x658e6e76L,0xf698b825L,0xd4d827efL,
0x5ae3e72fL,0x1ac783bbL,0x769c1db3L,0xb976a4ffL,
0xf9170aa4L,0x831263d5L,0x4ecc7682L ,0xe3b3382fL,
0xafa16637L,0x7daa2eL,0xc638b05L,0x3726eb2eL,
0xbb6a826dL,0x10492c6aL,0xcd771d34L,0xd25a52feL,
0x24130e11L,0xae9af4eeL },
{ 0x1b78c790L,0x78012c3cL,
0x49d7339aL,0xf5414bf8L,0x265a6c3L,0x7f1a06d6L,
0x31dcd842L,0x11f46b97L,0xbc9d75beL,0xb4e584f6L,
0x84f1f967L,0x5d29375dL,0x514bd21eL,0xa0c2dcbbL,
0x40da8a7dL,0xfdbfd8b2L,0x5018a7e4L,0x87ce944cL,
0x623e5c96L,0x45c33684L,0x29de9376L,0x2869735fL,
0xe2e005ecL,0x1da0dafdL,0x56349b3fL,0x43a1aca8L,
0x3c1b4f29L,0x578a50f2L,0x865e6f62L,0x7fc49a4fL,
0x330e2f1bL,0x289ccedfL,0xc6638e25L,0x7300c74bL,
0xab60f54bL,0x46557689L,0x9e7676dbL,0xbf1fc6b2L,
0x17d23739L,0x2e4b0b0cL,0x91a0a008L,0xb8d74aabL,
0xca65db8aL,0x913675fdL,0x453ea489L,0x314a6b82L,
0xb794a65bL,0x58c04142L,0x856b28feL,0xf26c8349L,
0xeee08103L,0xefcbb4cfL,0xd4e1acfdL,0x247c6d5cL,
0x18c64d9dL,0x29599273L,0xf55bd69aL,0x307c8bfcL,
0x946981eL,0x70e30b76L,0x908462abL,0x13512cacL,
0x10d13b44L,0xe4ffad6L,0x44a7eeaL,0xb66ee291L,
0x90234b47L,0xd237514fL,0x772601cbL,0x2ecb252fL,
0x3dc230f4L,0xd4b0eccaL,0x762ace1aL,0xf4f2e04bL,
0xcc8fc3acL,0x20f60c0aL,0x8cc50908L,0xd9a4040fL,
0xf17bccecL,0xac765a37L,0x1a1511c7L,0xc0b67edaL,
0x5e831addL,0xe84e24eL,0x88aefd45L,0x600e6b48L,
0xca0c5a2bL,0xab168aefL,0xb2669e38L,0x3fe9da4dL,
0xfdcae7d5L,0xb8de9e0dL,0xa24e2522L,0x976e4c87L,
0x31863284L,0xe3606fc4L,0xa97c72a5L,0x7cfca29aL,
0x1480b825L,0xd3b314a8L,0x52ba66a7L,0xb8835b8dL,
0x572bded4L,0x85180718L,0x74eb58c8L,0xc6f261b8L,
0xba330a5eL,0x9b2970eL,0xbcc51debL,0x6b94e037L,
0xdf4ef53aL,0xfd3ce9fdL,0xf7fb05baL,0x377d9390L,
0x8eea7ab8L,0xe37c564cL,0x4569cd24L,0xa481be70L,
0xfaee63afL,0x2f3d3edcL,0xb6889d26L,0xaff548ceL,
0xc36c5d85L,0x277a51faL,0x17bf44f6L,0xa79074cdL,
0x7eee807bL,0x8d802adbL,0x23bae8b0L,0xeb4d02f9L,
0xd9773e50L,0x1195e82dL,0x510b6a41L,0x42414ba2L,
0xce8ca6ddL,0xa4a1749L,0x73c600f5L,0x61ccfb04L,
0xbde273b7L,0x13c39801L,0xa3b7b3fbL,0x7af874c6L,
0x93bf089L,0xe89e8c13L,0xec53ac9eL,0xc0c77c87L,
0xefeaf5dcL,0x1ee13106L,0x4ae70c66L,0x476249c8L,
0xd7e1aff0L,0x737859d8L,0xca65b020L,0x72aa9f50L,
0xf6d1d3ddL,0x8e033b8cL,0xf860c5dL,0x802f6547L,
0xdad1b593L,0x35092290L,0x4a7ec622L,0xec8f54b9L,
0xe6372cfeL,0xe324ac7bL,0xdb842777L,0x2ae7b30fL,
0x560f0123L,0x700eaed9L,0x53b202ecL,0xfa95438L,
0x49e391f4L,0x6071b3b5L,0x759d9bbdL,0x65835874L,
0x96162b97L,0x57cdaabL,0xb533a162L,0x904fae46L,
0x56a08d42L,0xd056e25fL,0xa6588186L,0x9521513L,
0xffd32503L,0x4efa223dL,0xd72bbd28L,0x86366c2dL,
0xf4c30d02L,0x3f6c8453L,0x3fc31978L,0x309a31adL,
0x2e0c0f1eL,0x28d7f502L,0x57add05aL,0x2418da4fL,
0x62d186f3L,0x3f9f14e8L,0xf6ff8645L,0xe2cf4c13L,
0x8e962fe3L,0x65ec2094L,0x8819f30bL,0x71a28dcbL,
0x3281cbcL,0x8e13395aL,0x4ba75e84L,0x422c86dcL,
0x4d3d7956L,0xaf4e2410L,0x90d4e0b6L,0xc0dec214L,
0xea849d25L,0x818722eaL,0xecbba513L,0x37189eecL,
0x9db9184fL,0xff8f6823L,0x7f7cfa10L,0xc05746b0L,
```

```
0x52085d87L,0x18f433L,0x4cfa7cb7L,0xbfe20989L,
0xf4ff1328L,0x8656868cL,0xc9d8e300L,0xed138a11L,
0x3c660693L,0x70cf22ebL,0x78c16920L,0xfb209ccfL,
0x9b005bf3L,0x78538817L,0x85a497f1L,0xc43bf776L,
0xddfc395bL,0xed092e4bL,0x5a626fedL,0x764d509cL,
0xe3c5b8bL,0xe5813006L,0x26fbd8a2L,0x4d2bda1cL,
0xd6d3936bL,0x2e60bb1eL,0x2e81fc5cL,0xf42b1ce8L,
0x1f3a5ad0L,0xbcf11c23L,0x2623904bL,0x948159b7L,
0x39075445L,0xedd627dL,0xadfcf2e4L,0x4f358fffL,
0x79a21910L,0xb8cfa77aL },
{ 0xbc6664e3L,0xe57bb6c5L,
0xac9ae207L,0x82cfab4aL,0x65134d19L,0x767 49638L,
0x6f3a6142L,0x967cc794L,0xf14bf7aaL,0xe11b3afaL,
0xfee0c32L,0x96026e4eL,0x8636a3b2L,0xd63077b1L,
0x3b98f6cbL,0xa80b73feL,0xbf39ca6cL,0x3339a00bL,
0x99953bd2L,0xafeb55b4L,0x345a3b14L,0xa3bb953fL,
0xc68d7afdL,0xd849931aL,0xee3eff51L,0x6dbd1150L,
0xaa299a54L,0x44809315L,0x2ff79ef3L,0x61f748dbL,
0x55792c79L,0x74e45143L,0xe35ceeaaL,0xabe7872cL,
0x17118880L,0x92602689L,0x44c60cb3L,0x91a9d8fbL,
0x17f043ebL,0x9323fe21L,0x694fa0fL,0x2275ac66L,
0xf8070046L,0xf3a4d165L,0x561bf39aL,0x26a2d69fL,
0x4a9b6ca4L,0x85ab70bcL,0x5737e043L,0x353b33f3L,
0x327e791aL,0x3da28176L,0x1c12a961L,0xbc782daL,
0x29886974L,0x77b5ac8dL,0x73f87b51L,0x1fead657L,
0x68179c2dL,0x878371c3L,0x6be40bb7L,0x80700b71L,
0xbe8e53e2L,0x848077b9L,0x926bfba2L,0xdfe01f9aL,
0x4453af6fL,0x9ec7ef34L,0x79ac839cL,0x8f9bad11L,
0xe8453f4L,0xf3264856L,0xfc2c56f0L,0x49910608L,
0x20a577c4L,0xbe4f5fe1L,0x6f1fdf8cL,0xdfdeb72bL,
0xcc40cd74L,0x7a2810f8L,0xb9abeefeL,0xc5553d23L,
0xe01a5192L,0x49fd58d5L,0xd031be6cL,0x29153c34L,
0xa36d02ecL,0xa191710dL,0xf46c42ebL,0x5021646bL,
0xa0a71f8eL,0x29d0901eL,0xe2bbf2dcL,0x1dd2f86cL,
0xe9ac4a10L,0xd64eecccL,0x6bb6a89fL,0x1b80148aL,
0x3fe5f023L,0x34205abfL,0x49b6869eL,0xc19e23b5L,
0x82cf4536L,0x9e482da4L,0x6e5e2d50L,0x2c33cf62L,
0xaf6594ebL,0xd9bcd4fcL,0xe5df6e4cL,0xec54210eL,
0xfc2229afL,0x163c5a14L,0x9716bfd3L,0x9ac8911eL,
0x96d94219L,0xadbeb5d8L,0x28407226L,0xcc408dc9L,
0xdb610458L,0xcb021f11L,0xbb5fccf5L,0xe0d6afacL,
0x8cec70ebL,0xf09edbc8L,0x204cfebL,0x9453f270L,
0xbd7120a8L,0xdd8f8b6fL,0xf0f8091fL,0x2b27a683L,
0xfeaca1ddL,0xfca49344L,0xfdcea2efL,0x59be6f70L,
0x72d1168L,0x76c26561L,0x2a138807L,0xffd0b30aL,
0x6f10e1fbL,0xaf8f9de4L,0x67702aa0L,0xa7cfd9e2L,
0x9f99901bL,0xab8e5827L,0xcb1be286L,0xea238f41L,
0x411c5dc1L,0x295e3e23L,0x9bce7612L,0x132e1914L,
0x4b2c8b88L,0x2f761d5bL,0x61652217L,0x884efL,
0x3f89b9d3L,0xff97af67L,0xb5544c49L,0x33c94c0fL,
0x3bb2e360L,0x8e8ad71dL,0x5e9a7328L,0x8a4c1958L,
0xbd4a436eL,0xeeec5796L,0x92085e6eL,0x97c33cf3L,
0x65d41e7fL,0x4545f6c7L,0xb25c1582L,0x7d84f2d9L,
0x92708cb3L,0xbc156bb4L,0x97707007L,0x797f746eL,
0xeebe1216L,0x97f912cdL,0xed9019eL,0x3b5a3901L,
0xd893371L,0x2d3489f5L,0xfb8f3377L,0xb06dec49L,
0x2f9a663eL,0x5384436fL,0xc90715e5L,0x60b3e7f8L,
0x82e2f7dfL,0xa7bd0f6L,0x8ed8abb7L,0xae3de477L,
0x9166cc9bL,0x5396938eL,0xeea6ac23L,0xd5d2261bL,
0xed7bca7bL,0x58142550L,0x9fd11762L,0x2ec037f2L,
0x2e1ee132L,0x6750bbfeL,0xb12dcdb9L,0x6cbb1decL,
0x8391af46L,0xc554d8c9L,0xde21fcebL,0x67c3d873L,
0xbeb757baL,0x59a1fcabL,0x62683d7cL,0xdf716a07L,
0x9f11fae1L,0xea50abc1L,0x7348ec22L,0x996ee17dL,
0x977627e2L,0xc898baf5L,0x859aafffL,0x3e3bf421L,
0x7c0b8d8bL,0xbc9be192L,0x2ef242e4L,0x3f71560fL,
0xf7be6a6fL,0x2132fddL,0xc6b55885L,0x727d798bL,
0xb16b7ccfL,0xfbba88fcL,0xd7644ffbL,0x6d7b0971L,
0x69b02444L,0x3274ddccL,0xfbd404bL,0x7676c7beL,
0xbff2857aL,0x3e7f3eadL,0xf094a3edL,0x41bec1ecL,
0xd8e2d03aL,0xed3d562bL,0x73ae081L,0x41636733L,
0x39a12525L,0x8bdc5d6L,0xbd9f334L,0x2cd943d0L,
0x1860c644L,0x15e262a0L,0xc7251b8eL,0x85041065L,
0xf3b3ef68L,0xd2338e54L,0xb426e6d6L,0xd7936724L,
0x29a3103dL,0xbfc51978L },
{ 0x4f81787cL,0x14f0c763L,
0xa6ffcb2cL,0x7fac756fL,0x5dc31817L,0x35f0e2a1L,
0x644babecL,0xb5c8e69eL,0x2c8517edL,0x791cd6aeL,
0x55690b55L,0xf45ab43dL,0x2a535ce4L,0x6bd6d80eL,
0x1686f240L,0xba56cbaL,0x66d60a2aL,0x15102a14L,
0xc9af0aL,0x96ddff49L,0xe903c103L,0xc8ac6b46L,
0x7eebf3b2L,0x64c3f806L,0x9c9a5ab2L,0x60df99f2L,
0x147194b3L,0x6a5b0e32L,0xa8868242L,0x2c3c6195L,
0xe21d630eL,0x3d38754fL,0xa8b991afL,0x2e773d0eL,
0xe8e81d36L,0xff5317caL,0x15017bf4L,0xde8498ceL,
0xc85e9db6L,0x8c197392L,0xc9120804L,0x5291e21L,
0x585f3d67L,0x8da73fccL,0xa177c878L,0x4ea3d464L,
0xcad9aab0L,0x69455468L,0xe497c86fL,0x6a919052L,
0xc83eaf0aL,0xe0be95b7L,0x2c7dc8edL,0x22c4c99aL,
0xde971987L,0x33f93de6L,0xa347765cL,0x93882219L,
0xa22e8834L,0xba319e1dL,0x6814c715L,0xf75d2094L,
0xe4c25f62L,0x3e51b7dcL,0xa91c967L,0x9f819d31L,
0xb49660aL,0x9e1981a4L,0x39a0dc1fL,0x378c48f7L,
0x9f4301a7L,0x825a0124L,0x5dc32ff6L,0x5f10c3b1L,
0xe54d1b86L,0xc9ae9d1cL,0x646a0c93L,0xdb085203L,
0xe0688fe9L,0xb85a4cL,0x8e191aefL,0xb8f484eL,
0x1d591f92L,0x477a0d81L,0x6265885fL,0x9babf86cL,
0x25e34d64L,0x7057a0e9L,0xa723004aL,0x36bb927eL,
0x38cf96b7L,0xcc2a1f20L,0xa7bb5415L,0xf1cd2d16L,
0x15fdba5eL,0x7ce62b25L,0xf6d059e7L,0x3a017e12L,
0x1dbd86a0L,0xbfe60e1fL,0x17d9560eL,0xbfe61111L,
0xc42c10a1L,0x2ccc1eefL,0x702f970L,0x7d297839L,
0x62acf631L,0x12bafaaaL,0x8045af0fL,0x3f02a41aL,
0x98a68394L,0xcd51a8b6L,0x8b8ac7bL,0x6d16e525L,
0xfc68f8e7L,0x844bfff6L,0x497506d2L,0xb6a33a9eL,
0xc46c8c86L,0xfcbe2e3dL,0x1d1fecbaL,0x3b4580c1L,
0x5a08da12L,0x5c48dfecL,0x910380a0L,0x698df76dL,
0x2ef0bd79L,0x93d6bdc2L,0x7ffbc905L,0xee35cb4dL,
0x4042fe8dL,0x5b6a3e16L,0xdc6d4d6cL,0x695020d6L,
0x356b3054L,0x10e279d2L,0x25cfa576L,0x23ec09acL,
0xe47dacdaL,0xff5250b2L,0x12b9ce1dL,0x753d853aL,
0xc933236L,0x58a4e3afL,0x8b094229L,0xc1ba67d4L,
0x31069503L,0x61e419caL,0x397aee63L,0xbe749df5L,
0xdd38b226L,0xe58a2cb4L,0x9f6ce13cL,0xf9032ccbL,
0x5dad49c1L,0x5cf7219dL,0xdf140a2L,0x5a7a871fL,
0xb9d21b41L,0x3ee5a17fL,0xebd0c742L,0x1bd93fd8L,
0x323d75c4L,0x22f1a4d9L,0x544d9750L,0x29a9f0a5L,
0xa4bf7223L,0xd95e9195L,0x68987086L,0x66ce2d08L,
0xa6c4f2acL,0xdd1fcfc1L,0x3c4bc67fL,0xf11a35ccL,
0x75c8c96bL,0xa6982896L,0xd3401a47L,0x5eba42eeL,
0xcbf05287L,0x710c10dbL,0x3128fc34L,0x1958214fL,
0x93d0191eL,0x46d89b86L,0x3aaa664aL,0x81f0e646L,
0x23377780L,0xd3d54e63L,0x2a5495d2L,0x37a74c7eL,
0x57a4819bL,0x2c93ecf2L,0xc0fd17a3L,0x7fc3bc42L,
0x52d48060L,0x36eeb716L,0x78890da9L,0x91762fdaL,
0x97d408ebL,0xe10228e3L,0xc6f57c5bL,0xfa5ed8d5L,
0x2bf4bcdfL,0xced71372L,0x2297a82dL,0x49c032a5L,
0xcf6ad92dL,0x6643d0ccL,0x2f98ee3bL,0x90d8d053L,
0xf914b761L,0x25e13cf4L,0xab8e170dL,0xdd811886L,
0xc4ae867eL,0x56bbe86bL,0xa5a04fc3L,0x6ebb1561L,
0x54ca841aL,0xea651978L,0x233dbbc3L,0x3c69861dL,
0xea621ff7L,0xf6b713aeL,0x30c49438L,0x865028eL,
0xc787035L,0xe5ee7cb3L,0x4e443bc2L,0xd133f3a5L,
0x58127715L,0xf013a607L,0xb1358255L,0xa3185c4eL,
0xa1e159f5L,0xe8c09b89L,0xeb296857L,0xb6e99b7bL,
0xe61f3085L,0xccfd3ffdL,0xa33b4754L,0x3b456d07L,
0x3595c1c1L,0x12de6ae3L,0xa56ba7a2L,0x68be08e7L,
0x9061a3dfL,0xc2084b1dL,0x9aaa2b27L,0xbaa1fdbdL,
0xc12dc397L,0xc0ae34e9L,0x85c35bf3L,0x5b6d8865L,
0xec7cb92eL,0x1170691aL } };
```

```
typedef struct {
  unsigned long P[16 + 2];
  unsigned long S[4][256];
} BLOWFISH;

static unsigned long F(BLOWFISH *ctx, unsigned long x) {
   unsigned short a, b, c, d;
   unsigned long  y;

   d = (unsigned short)(x & 0xFF); //mascara
   x >>= 8;
   c = (unsigned short)(x & 0xFF);
   x >>= 8;
   b = (unsigned short)(x & 0xFF);
   x >>= 8;
   a = (unsigned short)(x & 0xFF);
   y = ctx->S[0][a] + ctx->S[1][b];
   y = y ^ ctx->S[2][c];
   y = y + ctx->S[3][d];
   return y;
}

void Cifra_Blowfish(BLOWFISH *ctx, unsigned long *xl,
unsigned long *xr){
  unsigned long  Xl;
  unsigned long  Xr;
  unsigned long  temp;
  short i;

  Xl = *xl;
  Xr = *xr;

  for (i = 0; i < N; ++i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;
    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }
  temp = Xl;
  Xl = Xr;
  Xr = temp;
  Xr = Xr ^ ctx->P[N];
  Xl = Xl ^ ctx->P[N + 1];
  *xl = Xl;
  *xr = Xr;
}

void Decifra_Blowfish(BLOWFISH *ctx, unsigned long *xl,
unsigned long *xr){
  unsigned long  Xl;
  unsigned long  Xr;
  unsigned long  temp;
  short i;

  Xl = *xl;
  Xr = *xr;

  for (i = N + 1; i > 1; --i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;
    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }
  temp = Xl;
  Xl = Xr;
  Xr = temp;
  Xr = Xr ^ ctx->P[1];
  Xl = Xl ^ ctx->P[0];
  *xl = Xl;
  *xr = Xr;
}

void Inicia_Blowfish(BLOWFISH *ctx) {
  int i, j, k;
  unsigned long data, datal, datar;

  //inicializa S-Boxes
  for (i = 0; i < 4; i++) {
   for (j = 0; j < 256; j++)
     ctx->S[i][j] = ORIG_S[i][j];
  }
  for (i = 0; i < 18; i++ )
     ctx->P[i] = ORIG_P[i];
}

int main( int argc, char *argv[] )
{
  BLOWFISH ctx;
  unsigned long l, r;

  FILE *arquivo;
  FILE *cifrado;
  FILE *decifrado;
  char nome[30];
  char chave[30];

  long int inicio, fim;
  int i;

    Inic_Blowfish(&ctx);
    printf("Digite o nome do arquivo a ser encriptado: ");
    gets(nome);
    arquivo = fopen(nome, "rb");
    if(arquivo == NULL) {
      printf("\nErro ao abrir arquivo! <%s>\n", nome);
      system("PAUSE");
      exit(1);
    }

    cifrado = fopen("cifrado", "wb");
    if(cifrado == NULL) {
      printf("\nErro ao abrir arquivo! <cifrado1>");
      system("PAUSE");
      exit(1);
    }

    inicio = clock();
    while(!feof(arquivo)) {
      fread(&l, sizeof(unsigned long), 1, arquivo);
      fread(&r, sizeof(unsigned long), 1, arquivo);
      Cifra_Blowfish(&ctx, &l, &r);
      fwrite(&l, sizeof(unsigned long), 1, cifrado);
      fwrite(&r, sizeof(unsigned long), 1, cifrado);
    }

    fim = clock();
    printf("\nTempo de encriptacao: %ld", fim - inicio);
    fclose(arquivo);
    fclose(cifrado);

    cifrado = fopen("cifrado", "rb");
    if(cifrado == NULL) {
      printf("\nErro ao abrir arquivo! <cifrado>");
      system("PAUSE");
      exit(1);
    }

    decifrado = fopen("decifrado", "wb");
```

```
    if(decifrado == NULL) {
      printf("\nErro ao abrir arquivo! <decifrado>");
      system("PAUSE");
      exit(1);
    }
    inicio = clock();
    inicio = clock();

    while(!feof(cifrado)) {
      fread(&l, sizeof(unsigned long), 1, cifrado);
      fread(&r, sizeof(unsigned long), 1, cifrado);
      Decifra_Blowfish(&ctx, &l, &r);
      fwrite(&l, sizeof(unsigned long), 1, decifrado);
      fwrite(&r, sizeof(unsigned long), 1, decifrado);
    }

    fim = clock();
    printf("\nTempo de decriptacao: %ld \n", fim - inicio);
    fclose(cifrado);
    fclose(decifrado);

   system("PAUSE");
   return 0;
}
```

Código Java Chave Dinâmica

Classe Blowfish

```
public class Blowfish {

  class BlowStrut {
    int S[ ][ ] = new int[4][256];
    int P[ ] = new int[18];
  }

  private int ks0[ ] = {
    0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7,
    0xb8e1afed, 0x6a267e96, 0xba7c9045, 0xf12c7f99,
    0x24a19947, 0xb3916cf7, 0x0801f2e2, 0x858efc16,
    0x636920d8, 0x71574e69, 0xa458fea3, 0xf4933d7e,
    0x0d95748f, 0x728eb658, 0x718bcd58, 0x82154aee,
    0x7b54a41d, 0xc25a59b5, 0x9c30d539, 0x2af26013,
    0xc5d1b023, 0x286085f0, 0xca417918, 0xb8db38ef,
    0x8e79dcb0, 0x603a180e, 0x6c9e0e8b, 0xb01e8a3e,
    0xd71577c1, 0xbd314b27, 0x78af2fda, 0x55605c60,
    0xe65525f3, 0xaa55ab94, 0x57489862, 0x63e81440,
    0x55ca396a, 0x2aab10b6, 0xb4cc5c34, 0x1141e8ce,
    0xa15486af, 0x7c72e993, 0xb3ee1411, 0x636fbc2a,
    0x2ba9c55d, 0x741831f6, 0xce5c3e16, 0x9b87931e,
    0xafd6ba33, 0x6c24cf5c, 0x7a325381, 0x28958677,
    0x3b8f4898, 0x6b4bb9af, 0xc4bfe81b, 0x66282193,
    0x61d809cc, 0xfb21a991, 0x487cac60, 0x5dec8032,
    0xef845d5d, 0xe98575b1, 0xdc262302, 0xeb651b88,
    0x23893e81, 0xd396acc5, 0x0f6d6ff3, 0x83f44239,
    0x2e0b4482, 0xa4842004, 0x69c8f04a, 0x9e1f9b5e,
    0x21c66842, 0xf6e96c9a, 0x670c9c61, 0xabd388f0,
    0x6a51a0d2, 0xd8542f68, 0x960fa728, 0xab5133a3,
    0x6eef0b6c, 0x137a3be4, 0xba3bf050, 0x7efb2a98,
    0xa1f1651d, 0x39af0176, 0x66ca593e, 0x82430e88,
    0x8cee8619, 0x456f9fb4, 0x7d84a5c3, 0x3b8b5ebe,
    0xe06f75d8, 0x85c12073, 0x401a449f, 0x56c16aa6,
    0x4ed3aa62, 0x363f7706, 0x1bfedf72, 0x429b023d,
    0x37d0d724, 0xd00a1248, 0xdb0fead3, 0x49f1c09b,
    0x075372c9, 0x80991b7b, 0x25d479d8, 0xf6e8def7,
    0xe3fe501a, 0xb6794c3b, 0x976ce0bd, 0x04c006ba,
    0xc1a94fb6, 0x409f60c4, 0x5e5c9ec2, 0x196a2463,
    0x68fb6faf, 0x3e6c53b5, 0x1339b2eb, 0x3b52ec6f,
    0x6dfc511f, 0x9b30952c, 0xcc814544, 0xaf5ebd09,
    0xbee3d004, 0xde334afd, 0x660f2807, 0x192e4bb3,
    0xc0cba857, 0x45c8740f, 0xd20b5f39, 0xb9d3fbdb,
    0x5579c0bd, 0x1a60320a, 0xd6a100c6, 0x402c7279,
    0x679f25fe, 0xfb1fa3cc, 0x8ea5e9f8, 0xdb3222f8,
    0x3c7516df, 0xfd616b15, 0x2f501ec8, 0xad0552ab,
    0x323db5fa, 0xfd238760, 0x53317b48, 0x3e00df82,
    0x9e5c57bb, 0xca6f8ca0, 0x1a87562e, 0xdf1769db,
    0xd542a8f6, 0x287effc3, 0xac6732c6, 0x8c4f5573,
    0x695b27b0, 0xbbca58c8, 0xe1ffa35d, 0xb8f011a0,
    0x10fa3d98, 0xfd2183b8, 0x4afcb56c, 0x2dd1d35b,
    0x9a53e479, 0xb6f84565, 0xd28e49bc, 0x4bfb9790,
    0xe1ddf2da, 0xa4cb7e33, 0x62fb1341, 0xcee4c6e8,
    0xef20cada, 0x36774c01, 0xd07e9efe, 0x2bf11fb4,
    0x95dbda4d, 0xae909198, 0xeaad8e71, 0x6b93d5a0,
    0xd08ed1d0, 0xafc725e0, 0x8e3c5b2f, 0x8e7594b7,
    0x8ff6e2fb, 0xf2122b64, 0x8888b812, 0x900df01c,
    0x4fad5ea0, 0x688fc31c, 0xd1cff191, 0xb3a8c1ad,
    0x2f2f2218, 0xbe0e1777, 0xea752dfe, 0x8b021fa1,
    0xe5a0cc0f, 0xb56f74e8, 0x18acf3d6, 0xce89e299,
    0xb4a84fe0, 0xfd13e0b7, 0x7cc43b81, 0xd2ada8d9,
    0x165fa266, 0x80957705, 0x93cc7314, 0x211a1477,
    0xe6ad2065, 0x77b5fa86, 0xc75442f5, 0xfb9d35cf,
    0xebcdaf0c, 0x7b3e89a0, 0xd6411bd3, 0xae1e7e49,
    0x00250e2d, 0x2071b35e, 0x226800bb, 0x57b8e0af,
    0x2464369b, 0xf009b91e, 0x5563911d, 0x59dfa6aa,
    0x78c14389, 0xd95a537f, 0x207d5ba2, 0x02e5b9c5,
    0x83260376, 0x6295cfa9, 0x11c81968, 0x4e734a41,
    0xb3472dca, 0x7b14a94a, 0x1b510052, 0x9a532915,
    0xd60f573f, 0xbc9bc6e4, 0x2b60a476, 0x81e67400,
    0x08ba6fb5, 0x571be91f, 0xf296ec6b, 0x2a0dd915,
    0xb6636521, 0xe7b9f9b6, 0xff34052e, 0xc5855664,
    0x53b02d5d, 0xa99f8fa1, 0x08ba4799, 0x6e85076a,
  };

  private int[ ] ks1 = {
    0x4b7a70e9, 0xb5b32944, 0xdb75092e, 0xc4192623,
    0xad6ea6b0, 0x49a7df7d, 0x9cee60b8, 0x8fedb266,
    0xecaa8c71, 0x699a17ff, 0x5664526c, 0xc2b19ee1,
    0x193602a5, 0x75094c29, 0xa0591340, 0xe4183a3e,
    0x3f54989a, 0x5b429d65, 0x6b8fe4d6, 0x99f73fd6,
    0xa1d29c07, 0xefe830f5, 0x4d2d38e6, 0xf0255dc1,
    0x4cdd2086, 0x8470eb26, 0x6382e9c6, 0x021ecc5e,
    0x09686b3f, 0x3ebaefc9, 0x3c971814, 0x6b6a70a1,
    0x687f3584, 0x52a0e286, 0xb79c5305, 0xaa500737,
    0x3e07841c, 0x7fdeae5c, 0x8e7d44ec, 0x5716f2b8,
    0xb03ada37, 0xf0500c0d, 0xf01c1f04, 0x0200b3ff,
    0xae0cf51a, 0x3cb574b2, 0x25837a58, 0xdc0921bd,
    0xd19113f9, 0x7ca92ff6, 0x94324773, 0x22f54701,
    0x3ae5e581, 0x37c2dadc, 0xc8b57634, 0x9af3dda7,
    0xa9446146, 0x0fd0030e, 0xecc8c73e, 0xa4751e41,
    0xe238cd99, 0x3bea0e2f, 0x3280bba1, 0x183eb331,
    0x4e548b38, 0x4f6db908, 0x6f420d03, 0xf60a04bf,
    0x2cb81290, 0x24977c79, 0x5679b072, 0xbcaf89af,
    0xde9a771f, 0xd9930810, 0xb38bae12, 0xdccf3f2e,
    0x5512721f, 0x2e6b7124, 0x501adde6, 0x9f84cd87,
    0x7a584718, 0x7408da17, 0xbc9f9abc, 0xe94b7d8c,
    0xec7aec3a, 0xdb851dfa, 0x63094366, 0xc464c3d2,
    0xef1c1847, 0x3215d908, 0xdd433b37, 0x24c2ba16,
    0x12a14d43, 0x2a65c451, 0x50940002, 0x133ae4dd,
    0x71dff89e, 0x10314e55, 0x81ac77d6, 0x5f11199b,
    0x043556f1, 0xd7a3c76b, 0x3c11183b, 0x5924a509,
    0xf28fe6ed, 0x97f1fbfa, 0x9ebabf2c, 0x1e153c6e,
    0x86e34570, 0xeae96fb1, 0x860e5e0a, 0x5a3e2ab3,
    0x771fe71c, 0x4e3d06fa, 0x2965dcb9, 0x99e71d0f,
    0x803e89d6, 0x5266c825, 0x2e4cc978, 0x9c10b36a,
    0xc6150eba, 0x94e2ea78, 0xa5fc3c53, 0x1e0a2df4,
    0xf2f74ea7, 0x361d2b3d, 0x1939260f, 0x19c27960,
    0x5223a708, 0xf71312b6, 0xebadfe6e, 0xeac31f66,
    0xe3bc4595, 0xa67bc883, 0xb17f37d1, 0x018cff28,
    0xc332ddef, 0xbe6c5aa5, 0x65582185, 0x68ab9802,
    0xeecea50f, 0xdb2f953b, 0x2aef7dad, 0x5b6e2f84,
    0x1521b628, 0x29076170, 0xecdd4775, 0x619f1510,
    0x13cca830, 0xeb61bd96, 0x0334fe1e, 0xaa0363cf,
    0xb5735c90, 0x4c70a239, 0xd59e9e0b, 0xcbaade14,
    0xeecc86bc, 0x60622ca7, 0x9cab5cab, 0xb2f3846e,
    0x648b1eaf, 0x19bdf0ca, 0xa02369b9, 0x655abb50,
    0x40685a32, 0x3c2ab4b3, 0x319ee9d5, 0xc021b8f7,
    0x9b540b19, 0x875fa099, 0x95f7997e, 0x623d7da8,
    0xf837889a, 0x97e32d77, 0x11ed935f, 0x16681281,
    0x0e358829, 0xc7e61fd6, 0x96dedfa1, 0x7858ba99,
    0x57f584a5, 0x1b227263, 0x9b83c3ff, 0x1ac24696,
    0xcdb30aeb, 0x532e3054, 0x8fd948e4, 0x6dbc3128,
    0x58ebf2ef, 0x34c6ffea, 0xfe28ed61, 0xee7c3c73,
    0x5d4a14d9, 0xe864b7e3, 0x42105d14, 0x203e13e0,
    0x45eee2b6, 0xa3aaabea, 0xdb6c4f15, 0xfacb4fd0,
    0xc742f442, 0xef6abbb5, 0x654f3b1d, 0x41cd2105,
    0xd81e799e, 0x86854dc7, 0xe44b476a, 0x3d816250,
    0xcf62a1f2, 0x5b8d2646, 0xfc8883a0, 0xc1c7b6a3,
    0x7f1524c3, 0x69cb7492, 0x47848a0b, 0x5692b285,
    0x095bbf00, 0xad19489d, 0x1462b174, 0x23820e00,
    0x58428d2a, 0x0c55f5ea, 0x1dadf43e, 0x233f7061,
    0x3372f092, 0x8d937e41, 0xd65fecf1, 0x6c223bdb,
    0x7cde3759, 0xcbee7460, 0x4085f2a7, 0xce77326e,
    0xa6078084, 0x19f8509e, 0xe8efd855, 0x61d99735,
```

```
    0xa969a7aa, 0xc50c06c2, 0x5a04abfc, 0x800bcadc,
    0x9e447a2e, 0xc3453484, 0xfdd56705, 0x0e1e9ec9,
    0xdb73dbd3, 0x105588cd, 0x675fda79, 0xe3674340,
    0xc5c43465, 0x713e38d8, 0x3d28f89e, 0xf16dff20,
    0x153e21e7, 0x8fb03d4a, 0xe6e39f2b, 0xdb83adf7,
};

  private int ks2[ ] = {
    0xe93d5a68, 0x948140f7, 0xf64c261c, 0x94692934,
    0x411520f7, 0x7602d4f7, 0xbcf46b2e, 0xd4a20068,
    0xd4082471, 0x3320f46a, 0x43b7d4b7, 0x500061af,
    0x1e39f62e, 0x97244546, 0x14214f74, 0xbf8b8840,
    0x4d95fc1d, 0x96b591af, 0x70f4ddd3, 0x66a02f45,
    0xbfbc09ec, 0x03bd9785, 0x7fac6dd0, 0x31cb8504,
    0x96eb27b3, 0x55fd3941, 0xda2547e6, 0xabca0a9a,
    0x28507825, 0x530429f4, 0x0a2c86da, 0xe9b66dfb,
    0x68dc1462, 0xd7486900, 0x680ec0a4, 0x27a18dee,
    0x4f3ffea2, 0xe887ad8c, 0xb58ce006, 0x7af4d6b6,
    0xaace1e7c, 0xd3375fec, 0xce78a399, 0x406b2a42,
    0x20fe9e35, 0xd9f385b9, 0xee39d7ab, 0x3b124e8b,
    0x1dc9faf7, 0x4b6d1856, 0x26a36631, 0xeae397b2,
    0x3a6efa74, 0xdd5b4332, 0x6841e7f7, 0xca7820fb,
    0xfb0af54e, 0xd8feb397, 0x454056ac, 0xba489527,
    0x55533a3a, 0x20838d87, 0xfe6ba9b7, 0xd096954b,
    0x55a867bc, 0xa1159a58, 0xcca92963, 0x99e1db33,
    0xa62a4a56, 0x3f3125f9, 0x5ef47e1c, 0x9029317c,
    0xfdf8e802, 0x04272f70, 0x80bb155c, 0x05282ce3,
    0x95c11548, 0xe4c66d22, 0x48c1133f, 0xc70f86dc,
    0x07f9c9ee, 0x41041f0f, 0x404779a4, 0x5d886e17,
    0x325f51eb, 0xd59bc0d1, 0xf2bcc18f, 0x41113564,
    0x257b7834, 0x602a9c60, 0xdff8e8a3, 0x1f636c1b,
    0x0e12b4c2, 0x02e1329e, 0xaf664fd1, 0xcad18115,
    0x6b2395e0, 0x333e92e1, 0x3b240b62, 0xeebeb922,
    0x85b2a20e, 0xe6ba0d99, 0xde720c8c, 0x2da2f728,
    0xd0127845, 0x95b794fd, 0x647d0862, 0xe7ccf5f0,
    0x5449a36f, 0x877d48fa, 0xc39dfd27, 0xf33e8d1e,
    0x0a476341, 0x992eff74, 0x3a6f6eab, 0xf4f8fd37,
    0xa812dc60, 0xa1ebddf8, 0x991be14c, 0xdb6e6b0d,
    0xc67b5510, 0x6d672c37, 0x2765d43b, 0xdcd0e804,
    0xf1290dc7, 0xcc00ffa3, 0xb5390f92, 0x690fed0b,
    0x667b9ffb, 0xcedb7d9c, 0xa091cf0b, 0xd9155ea3,
    0xbb132f88, 0x515bad24, 0x7b9479bf, 0x763bd6eb,
    0x37392eb3, 0xcc115979, 0x8026e297, 0xf42e312d,
    0x6842ada7, 0xc66a2b3b, 0x12754ccc, 0x782ef11c,
    0x6a124237, 0xb79251e7, 0x06a1bbe6, 0x4bfb6350,
    0x1a6b1018, 0x11caedfa, 0x3d25bdd8, 0xe2e1c3c9,
    0x44421659, 0x0a121386, 0xd90cec6e, 0xd5abea2a,
    0x64af674e, 0xda86a85f, 0xbebfe988, 0x64e4c3fe,
    0x9dbc8057, 0xf0f7c086, 0x60787bf8, 0x6003604d,
    0xd1fd8346, 0xf6381fb0, 0x7745ae04, 0xd736fccc,
    0x83426b33, 0xf01eab71, 0xb0804187, 0x3c005e5f,
    0x77a057be, 0xbde8ae24, 0x55464299, 0xbf582e61,
    0x4e58f48f, 0xf2ddfda2, 0xf474ef38, 0x8789bdc2,
    0x5366f9c3, 0xc8b38e74, 0xb475f255, 0x46fcd9b9,
    0x7aeb2661, 0x8b1ddf84, 0x846a0e79, 0x915f95e2,
    0x466e598e, 0x20b45770, 0x8cd55591, 0xc902de4c,
    0xb90bace1, 0xbb8205d0, 0x11a86248, 0x7574a99e,
    0xb77f19b6, 0xe0a9dc09, 0x662d09a1, 0xc4324633,
    0xe85a1f02, 0x09f0be8c, 0x4a99a025, 0x1d6efe10,
    0x1ab93d1d, 0x0ba5a4df, 0xa186f20f, 0x2868f169,
    0xdcb7da83, 0x573906fe, 0xa1e2ce9b, 0x4fcd7f52,
    0x50115e01, 0xa70683fa, 0xa002b5c4, 0x0de6d027,
    0x9af88c27, 0x773f8641, 0xc3604c06, 0x61a806b5,
    0xf0177a28, 0xc0f586e0, 0x006058aa, 0x30dc7d62,
    0x11e69ed7, 0x2338ea63, 0x53c2dd94, 0xc2c21634,
    0xbbcbee56, 0x90bcb6de, 0xebfc7da1, 0xce591d76,
    0x6f05e409, 0x4b7c0188, 0x39720a3d, 0x7c927c24,
    0x86e3725f, 0x724d9db9, 0x1ac15bb4, 0xd39eb8fc,
    0xed545578, 0x08fca5b5, 0xd83d7cd3, 0x4dad0fc4,
    0x1e50ef5e, 0xb161e6f8, 0xa28514d9, 0x6c51133c,
    0x6fd5c7e7, 0x56e14ec4, 0x362abfce, 0xddc6c837,
    0xd79a3234, 0x92638212, 0x670efa8e, 0x406000e0,
};

private int ks3[ ] = {
    0x3a39ce37, 0xd3faf5cf, 0xabc27737, 0x5ac52d1b,
    0x5cb0679e, 0x4fa33742, 0xd3822740, 0x99bc9bbe,
    0xd5118e9d, 0xbf0f7315, 0xd62d1c7e, 0xc700c47b,
    0xb78c1b6b, 0x21a19045, 0xb26eb1be, 0x6a366eb4,
    0x5748ab2f, 0xbc946e79, 0xc6a376d2, 0x6549c2c8,
    0x530ff8ee, 0x468dde7d, 0xd5730a1d, 0x4cd04dc6,
    0x2939bbdb, 0xa9ba4650, 0xac9526e8, 0xbe5ee304,
    0xa1fad5f0, 0x6a2d519a, 0x63ef8ce2, 0x9a86ee22,
    0xc089c2b8, 0x43242ef6, 0xa51e03aa, 0x9cf2d0a4,
    0x83c061ba, 0x9be96a4d, 0x8fe51550, 0xba645bd6,
    0x2826a2f9, 0xa73a3ae1, 0x4ba99586, 0xef5562e9,
    0xc72fefd3, 0xf752f7da, 0x3f046f69, 0x77fa0a59,
    0x80e4a915, 0x87b08601, 0x9b09e6ad, 0x3b3ee593,
    0xe990fd5a, 0x9e34d797, 0x2cf0b7d9, 0x022b8b51,
    0x96d5ac3a, 0x017da67d, 0xd1cf3ed6, 0x7c7d2d28,
    0x1f9f25cf, 0xadf2b89b, 0x5ad6b472, 0x5a88f54c,
    0xe029ac71, 0xe019a5e6, 0x47b0acfd, 0xed93fa9b,
    0xe8d3c48d, 0x283b57cc, 0xf8d56629, 0x79132e28,
    0x785f0191, 0xed756055, 0xf7960e44, 0xe3d35e8c,
    0x15056dd4, 0x88f46dba, 0x03a16125, 0x0564f0bd,
    0xc3eb9e15, 0x3c9057a2, 0x97271aec, 0xa93a072a,
    0x1b3f6d9b, 0x1e6321f5, 0xf59c66fb, 0x26dcf319,
    0x7533d928, 0xb155fdf5, 0x03563482, 0x8aba3cbb,
    0x28517711, 0xc20ad9f8, 0xabcc5167, 0xccad925f,
    0x4de81751, 0x3830dc8e, 0x379d5862, 0x9320f991,
    0xea7a90c2, 0xfb3e7bce, 0x5121ce64, 0x774fbe32,
    0xa8b6e37e, 0xc3293d46, 0x48de5369, 0x6413e680,
    0xa2ae0810, 0xdd6db224, 0x69852dfd, 0x09072166,
    0xb39a460a, 0x6445c0dd, 0x586cdecf, 0x1c20c8ae,
    0x5bbef7dd, 0x1b588d40, 0xccd2017f, 0x6bb4e3bb,
    0xdda26a7e, 0x3a59ff45, 0x3e350a44, 0xbcb4cdd5,
    0x72eacea8, 0xfa6484bb, 0x8d6612ae, 0xbf3c6f47,
    0xd29be463, 0x542f5d9e, 0xaec2771b, 0xf64e6370,
    0x740e0d8d, 0xe75b1357, 0xf8721671, 0xaf537d5d,
    0x4040cb08, 0x4eb4e2cc, 0x34d2466a, 0x0115af84,
    0xe1b00428, 0x95983a1d, 0x06b89fb4, 0xce6ea048,
    0x6f3f3b82, 0x3520ab82, 0x011a1d4b, 0x277227f8,
    0x611560b1, 0xe7933fdc, 0xbb3a792b, 0x344525bd,
    0xa08839e1, 0x51ce794b, 0x2f32c9b7, 0xa01fbac9,
    0xe01cc87e, 0xbcc7d1f6, 0xcf0111c3, 0xa1e8aac7,
    0x1a908749, 0xd44fbd9a, 0xd0dadecb, 0xd50ada38,
    0x0339c32a, 0xc6913667, 0x8df9317c, 0xe0b12b4f,
    0xf79e59b7, 0x43f5bb3a, 0xf2d519ff, 0x27d9459c,
    0xbf97222c, 0x15e6fc2a, 0x0f91fc71, 0x9b941525,
    0xfae59361, 0xceb69ceb, 0xc2a86459, 0x12baa8d1,
    0xb6c1075e, 0xe3056a0c, 0x10d25065, 0xcb03a442,
    0xe0ec6e0e, 0x1698db3b, 0x4c98a0be, 0x3278e964,
    0x9f1f9532, 0xe0d392df, 0xd3a0342b, 0x8971f21e,
    0x1b0a7441, 0x4ba3348c, 0xc5be7120, 0xc37632d8,
    0xdf359f8d, 0x9b992f2e, 0xe60b6f47, 0x0fe3f11d,
    0xe54cda54, 0x1edad891, 0xce6279cf, 0xcd3e7e6f,
    0x1618b166, 0xfd2c1d05, 0x848fd2c5, 0xf6fb2299,
    0xf523f357, 0xa6327623, 0x93a83531, 0x56cccd02,
    0xacf08162, 0x5a75ebb5, 0x6e163697, 0x88d273cc,
    0xde966292, 0x81b949d0, 0x4c50901b, 0x71c65614,
    0xe6c6c7bd, 0x327a140a, 0x45e1d006, 0xc3f27b9a,
    0xc9aa53fd, 0x62a80f00, 0xbb25bfe2, 0x35bdd2f6,
    0x71126905, 0xb2040222, 0xb6cbcf7c, 0xcd769c2b,
    0x53113ec0, 0x1640e3d3, 0x38abbd60, 0x2547adf0,
    0xba38209c, 0xf746ce76, 0x77afa1c5, 0x20756060,
    0x85cbfe4e, 0x8ae88dd8, 0x7aaaf9b0, 0x4cf9aa7e,
    0x1948c25c, 0x02fb8a8c, 0x01c36ae4, 0xd6ebe1f9,
    0x90d4f869, 0xa65cdea0, 0x3f09252d, 0xc208e69f,
    0xb74e6132, 0xce77e25b, 0x578fdfe3, 0x3ac372e6,
};
```

```
private int[ ] ps = {
  0x243f6a88, 0x85a308d3, 0x13198a2e, 0x03707344,
  0xa4093822, 0x299f31d0, 0x082efa98, 0xec4e6c89,
  0x452821e6, 0x38d01377, 0xbe5466cf, 0x34e90c6c,
  0xc0ac29b7, 0xc97c50dd, 0x3f84d5b5, 0xb5470917,
  0x9216d5d9, 0x8979fb1b,
};
private BlowStrut ctx;

public Blowfish ( byte[ ] key ) {
  this.ctx = new BlowStrut();
  System.arraycopy(ps, 0, this.ctx.P, 0, 18);
  System.arraycopy(ks0, 0, this.ctx.S[0], 0, 256);
  System.arraycopy(ks1, 0, this.ctx.S[1], 0, 256);
  System.arraycopy(ks2, 0, this.ctx.S[2], 0, 256);
   System.arraycopy(ks3, 0, this.ctx.S[3], 0, 256);
  int data;
  int j = 0, i;
  for (i = 0; i < 18; ++i) {
    data = 0x00000000;
    for (int k = 0; k < 4; ++k) {
      data = (data << 8) | (key[j] & 0xFF);
      j++;
      if (j >= key.length)
        j = 0;
    }
     this.ctx.P[i] ^= data;
  }
  byte[] b = new byte[8];
  for (i = 0; i < 18; i += 2) {
    cifrar(b);
    this.ctx.P[i] = b2d(b, 0);
    this.ctx.P[i + 1] = b2d(b, 4);
  }
  for (i = 0; i < 4; ++i) {
    for (j = 0; j < 256; j += 2) {
      decifrar(b);
      ctx.S[i][j] = b2d(b, 0);
      ctx.S[i][j + 1] = b2d(b, 4);
    }
  }
}

private int F(int x) {
  int a, b, c, d;
  d = x & 0xFF;
  x >>= 8;
  c = x & 0xFF;
  x >>= 8;
  b = x & 0xFF;
  x >>= 8;
  a = x & 0xFF;
  int y = this.ctx.S[0][a] + this.ctx.S[1][b];
  y ^= this.ctx.S[2][c];
  y += this.ctx.S[3][d];
  return y;
}

private int b2d(byte[ ] b, int p) {
  int r = 0;
  r |= b[p + 3] & 0xFF;
  r <<= 8;
  r |= b[p + 2] & 0xFF;
  r <<= 8;
  r |= b[p + 1] & 0xFF;
  r <<= 8;
  r |= b[p] & 0xFF;
  return r;
}

private void d2b(int a, byte[ ] b, int p) {
  b[p] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 1] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 2] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 3] = (byte) (a & 0xFF);
}

public void cifrar(byte[ ] data) {
  int blocks = data.length >> 3;
  for (int k = 0, p; k < blocks; k++) {
    p = k << 3;
    int Xl = b2 d(data, p);
    int Xr = b2d(data, p + 4);
    int tmp;
    for (int i = 0; i < 16; i++) {
      Xl = Xl ^ this.ctx.P[i];
      Xr = F(Xl) ^ Xr;
      tmp = Xl;
      Xl = Xr;
      Xr = tmp;
    }
    tmp = Xl;
    Xl = Xr;
    Xr = tmp;
    Xr ^= this.ctx.P[16];
    Xl ^= this.ctx.P[17];
    d2b(Xl, data, p);
    d2b(Xr, data, p + 4);
  }
}

public void decifrar(byte[ ] data) {
  int blocks = data.length >> 3;
  for (int k = 0, p; k < blocks; k++) {
    p = k << 3;
    int Xl = b2d(data, p);
    int Xr = b2d(data, p + 4);
    int tmp;
    for (int i = 17; i > 1; i--) {
      Xl = Xl ^ this.ctx.P[i];
      Xr = F(Xl) ^ Xr;
      tmp = Xl;
      Xl = Xr;
      Xr = tmp;
    }
    tmp = Xl;
    Xl = Xr;
    Xr = tmp;
    Xr ^= this.ctx.P[1];
    Xl ^= this.ctx.P[0];
    d2b(Xl, data, p);
    d2b(Xr, data, p + 4);
  }
}

public byte[ ] padding(byte[ ] a, int p) {
  int l = (a.length | 7) + 1;
  byte[] b = new byte[l];
```

```
    for (int i = 0; i < b.length; i++)
        b[i] = (byte) p;
    System.arraycopy(a, 0, b, 0, a.length);
    return b;
  }
}
```

Classe NewJFrame

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JSeparator;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JTextPane;

import javax.swing.WindowConstants;
import javax.swing.filechooser.FileFilter;

public class NewJFrame extends javax.swing.JFrame {
    private JTextField JTextChave;
    private JLabel jLabelChave;
    private JMenuItem jMenuItemOpen;
    private JMenuItem jMenuItemClose;
    private JButton jButtonOk;
    private JButton jButtonDecifrar;
    private JButton jButtonCifrar;
    private JSeparator jSeparator1;
    private JMenu jMenu1;
    private JMenuBar jMenuBar1;
    Blowfish key;
    File inputFile;
    File encripFile;
    private JTextPane jTextPane1;
    File decripFile;
    FileInputStream in = null;
    FileOutputStream out = null;

    public static void main(String[] args) {
      NewJFrame inst = new NewJFrame();
      inst.setVisible(true);
    }

    public NewJFrame() {
      super();
      initGUI();
      show();
     }

    private void initGUI() {
      try {
              {
                jMenuBar1 = new JMenuBar();
                 setJMenuBar(jMenuBar1);
                 {
                   jMenu1 = new JMenu();
                   jMenuBar1.add(jMenu1);
                   jMenu1.setText("File");
                   {
                     jMenuItemOpen = new JMenuItem();
                     jMenu1.add(jMenuItemOpen);
                     jMenuItemOpen.setText("Open File");
                     jMenuItemOpen.addActionListener(new
                           ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                           jMenuItemOpenActionPerformed(evt);
 }
});
      }
      {
       jSeparator1 = new JSeparator();
       jMenu1.add(jSeparator1);
      }
      {
       jMenuItemClose = new JMenuItem();
       jMenu1.add(jMenuItemClose);
       jMenuItemClose.setText("Close");
       jMenuItemClose.addActionListener(new
              ActionListener() {
                public void
                  actionPerformed(ActionEvent evt) {
                 jMenuItemCloseActionPerformed(evt);
                }
       });
      }
     }
}
setDefaultCloseOperation(
WindowConstants.DISPOSE_ON_CLOSE);
this.getContentPane().setLayout(null);
this.setResizable(false);
this.setTitle("Interface");
{
  JTextChave = new JTextField();
  this.getContentPane().add(JTextChave);
  JTextChave.setBounds(71, 85, 158, 18);
}
{
  jLabelChave = new JLabel();
  this.getContentPane().add(jLabelChave);
  jLabelChave.setText("Chave");
  jLabelChave.setBounds(23, 77, 60, 30);
}
{
  jButtonCifrar = new JButton();
  this.getContentPane().add(jButtonCifrar);
  jButtonCifrar.setT ext("Cifrar");
  jButtonCifrar.setBounds(58, 158, 94, 29);
  jButtonCifrar.addActionListener(new
              ActionListener() {
      public void actionPerformed(ActionEvent evt)
       {
              jButtonCifrarActionPerformed(evt);
             }
  });
}
{
  jButtonDecifrar = new JButton();
  this.getContentPane().add(jButtonDecifrar);
  jButtonDecifrar.setText("Decifrar");
  jButtonDecifrar.setBounds(194, 158, 94, 29);
  jButtonDecifrar.addActionListener(new
              ActionListener() {
      public void actio nPerformed(ActionEvent evt)
       {
            jButtonDecifrarActionPerformed(evt);
       }
  });
}
{
  jButtonOk = new JButton();
  this.getContentPane().add(jButtonOk);
  jButtonOk.setText("OK");
  jButtonOk.setBounds(242 , 83, 60, 20);
  jButtonOk.addActionListener(new
     ActionListener() {
```

```
                public void actionPerformed(ActionEvent evt)
                {
                    jButtonOkActionPerformed(evt);
                }
            });
        }
        {
            jTextPane1 = new JTextPane();
            this.getContentPane().add(jTextPane1);
            jTextPane1.setBounds(3, 215, 388, 21);
            jTextPane1.setEditable(false);
        }
        pack();
        this.setSize(402, 300);
    } catch (Exception e) {
        e.printStackTrace();
    }
  }

  private void jMenuItemOpenActionPerformed(ActionEvent
evt) {
    JFileChooser chooser = new JFileChooser();
    chooser.setVisible(true);
    int returnVal = chooser.showOpenDialog(chooser);
      if (returnVal == JFileChooser.APPROVE_OPTION)
        inputFile = chooser.getSelectedFile();
      System.out.println("Nome do arquivo: " +
        inputFile.getName());
  }

  private void jMenuItemCloseActionPerformed(ActionEvent
evt) {
    System.exit(0);
  }

private void jButtonOkActionPerformed(ActionEvent evt)
{
  key = new Blowfish(JTextChave.getText().getBytes());
}

private void jButtonCifrarActionPerformed(ActionEvent evt) {
    double inicio, fim;
    int n;
    byte c[ ] = new byte[8];
    encripFile = new File("encriptado");

    try {
        in = new FileInputStream(inputFile);
    }
    catch (Exception e) {}

    try {
        out = new FileOutputStream(encripFile);
    }
    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];
          key.cifrar(lidos);
          out.write(lidos,0,lidos.length);
        }
        in.close();
        out.close();
    }
    catch (IOException e){}

    fim = System.currentTimeMillis();

    jTextPane1.setText("Tempo de Encriptação: " + (fim -
inicio));
  }

  private void jButtonDecifrarActionPerformed(ActionEvent
evt) {
    double inicio, fim;
    int n;
    byte c[ ] = new byte[8];
    encripFile = new File("encriptado");
    decripFile = new File("decriptado");

    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];
          key.decifrar(lidos);
          out.write(lidos,0,lidos.length);
        }
        in.close();
        out.close();
    }
    catch (IOException e){    }
    fim = System.currentTimeMillis();
    jTextPane1.setText("Tempo de Decriptação: " + (fim -
inicio));
  }
}
```

Código Java Chave Estática 32 bits

```
public class BlowfishChaveFixa32 {
    class BlowStrut {
        int S[ ][ ] = new int[4][256];
        int P[ ] = new int[18];
    }
    private int ks0[ ] = {
        0x2563ef98,0x4d9ee216,
        0xb6003691,0xd743587e,0x6c556e23,0xfe5a01e7,
        0x94fd298,0x5ada3678,0x48dc8203,0x688c884,
        0xefe74433,0x5739b6d4,0x4e55f038,0x5b112d6a,
        0x73818d9b,0x24f40946,0x39927ee0,0x98588592,
        0x5bbe064d,0xf6d7df53,0xf7d00343,0xab5924,
        0x543673f4,0x98952205,0xad2efddb,0x86d3c8d6,
        0xe5b3c0cd,0x2577b2a6,0xcc048a34,0x6c73f006,
        0xff216c39,0xfc357c1e,0xe0ac49d6,0x6562f9a9,
        0xa24605cf,0xe2507bd7,0xd8787e72,0x5fe5b04b,
        0xdf203a63,0x399ff53d,0xf45888d6,0x4 7c9565a,
        0x6d972b34,0x3cf1dc1e,0xa77de156,0x547c0033,
        0x1f652364,0x45c1cdab,0xf7716dad,0x343dd0ec,
        0xe2b25d93,0x7a9f71d6,0x90fcfd0,0xa7fef7eb,
        0x7641a772,0xff4de4e6,0x66766321,0x72ff8f4f,
        0xf3546786,0xd9a5d12b,0x99f0e138,0x8b71b680,
        0x18d1d64,0xbfc82e03,0xd08efd74,0x5a064a,
        0x265e2c4,0x6a29fef9,0x91c1968f,0xb5602d76,
        0x1aa425b5,0x5a2e515d,0x4095b414,0x566c3d1e,
        0x236221bd,0xe5dbe10c,0x23ea8591,0x49efdf0b,
        0xf7813fb0,0x1c86e8b4,0x6383c15b,0x3e3cce20,
        0x54d07067,0xbe4c151d,0x6a30653d,0xfb45beb1,
        0x6741ca45,0xd7e1b18,0x77b73ba9,0xb8f048f0,
        0x678676bf,0x1a4a4650,0x8b2b5963,0x9b9dbbeb,
        0x4c5cad64,0x7d6e6318,0x75164606,0x2338a791,
        0xf056dc6a,0x744584d,0xe7562d2e,0x3120aa7,
        0xe84917a6,0xf2ac5610,0x99820eaa,0xf2fefcc9,
        0xae005c51,0x53387dac,0x577edee5,0x36763e11,
        0xe08f195b,0xb6c91ef1,0x2c7299b1,0xa118ead5,
        0xb9ac69f1,0x6d7eac5d,0x5a7a4e18,0xd58e662c,
        0x8741d041,0x85c251d0,0x1715c829,0x229bc5d1,
        0x314d31eb,0xe441c239,0xf645ad9c,0xcd1c6e32,
        0x7272f8fc,0x3ca09e82,0x7da0c78a,0x5532c440,
        0x16cd9a80,0xeb317d36,0xb9f1ccc9,0x796485fb,
        0x1f925a82,0x1718e380,0x5b5eebef,0x72411723,
        0x1f5c0ca4,0xc54582fa,0xd8a26990,0xc669af9e,
        0x1599c3fe,0xc59f1b08,0xdf65702a,0x4e0453b4,
        0xcc7c8cbc,0xf086c86b,0x67c10136,0x868b5c63,
        0x13edc457,0xcbf94ecf,0x5509db5e,0x8 ceafb95,
        0xbd8ed38,0x243dff80,0xd050df08,0xae6df7f6,
        0xb43daa49,0x65f8bf8b,0xf27ca452,0xa78fbee2,
        0xc506605f,0x541790c0,0xa87ba0db,0x7f57c98d,
        0x639a0b66,0xdfd0f0b1,0x9d506d69,0x38e05852,
        0xc6371ed8,0x626a466f,0x892bb592,0x61a420fa,
        0xca43db5b,0x2512bdcc,0x687eff5a,0x60b81f41,
        0x88cb0cc,0xa0090888,0xcd6a0d95,0x1579efb7,
        0x70210cdc,0x2524d71d,0x3ff18db9,0xffc43e34,
        0x977515a8,0xea5d971,0xfd0527db,0x89751c30,
        0x4185c420,0x12a207ab,0x5e5bfe70,0x8a05cca7,
        0x6e19b5a5,0xb449dbbc,0x899f320f,0x19f00040,
        0xcf6b9d93,0xde840554,0xbecdfc25,0xbec56c53,
        0x25f3e298,0xe0d35e00,0x4337f116,0x7ba2e28c,
        0x6bc404b2,0xe20755fc,0x3ba37fa6,0x156fade5,
        0x512c46f,0xf0492f0d,0x966e5968,0xe095992e,
        0xdeef240c,0x68430c2d,0xea61a460,0x93c66934,
        0x7dc9d90c,0x8208d01e,0x44cfec46,0x9c9fb7c3,
        0xebf147fe,0x60802f83,0x8a637604,0xde7fb976,
        0x652ec336,0x33f43477,0x19e7361f,0x2f6fca1f,
        0x7a9ebc74,0xb2c7e414,0x664424bc,0x9590641f,
        0x5fa68f5c,0xc1b8067e,0x4be31b2c,0xb489f46c,
        0xc7c7b1d7,0x2cb007f9,0x50cc7dd2,0xeae2dc6c,
        0xb5ff84e6,0x9c9c5231,0xea5472ec,0xf3dbe0a4,
        0x3d02174,0xc115627e,0x60f2054d,0x8304e4fc,
        0xcfedca27,0x64010003,0x21cf5e1d,0x7b79cfcb,
        0x1551a100,0xfbdee2a9 };
    private int[ ] ks1 = {
        0x78fc09a0,0xa727b2ea,
        0x272ea1f7,0x274fdf99,0xc6f0b1d9,0x56b4d41d,
        0x4828fd61,0x4dfe24ee,0x5c626d5a,0xde68e1c2,
        0x119fd658,0xf3f48ee,0x5152994c,0x6dfc67bd,
        0xcea534d5,0x981facb,0x1868c88e,0x353896b3,
        0x7ff22b13,0x84dc9f43,0x2663c951,0x92ac59e4,
        0x3b24190a,0xa1d7636,0x39e381c5,0x394dc1a1,
        0x7b8c1fba,0xc00a0626,0x4f1bc88e,0x93660ad9,
        0xdd57bc0d,0xadd1214d,0xa3a4bdf,0x843fc8be,
        0x6b30882d,0x6214a9c2,0x5c817a2e,0x25f43c3,
        0x664f7632,0xfceb3ce1,0xda1fb3b3,0xaa6f3e09,
        0xf46d7c78,0x401e0084,0x5814a348,0x55474d25,
        0xb9910aed,0xb1242417,0xb05adaea,0x339693e9,
        0x5e3bc946,0x96a8fbd9,0x9525aa5b,0xd401718,
        0x3d536838,0x2fe06715,0xeed26717,0x75c09f81,
        0x7e266bf3,0x7deba5a,0x2dbc0c0b,0x1f0fe4c,
        0xb8246d57,0x783fd8f6,0x2defd5d6,0x9c5e45d6,
        0x14124357,0x8a44d360,0xbacbaa5d,0x27c91a7f,
        0xab9a35b7,0x186c3770,0x565829f0,0x11fb483f,
        0xb9f0951f,0x7cca442a,0x97bbd48b,0x73445c0f,
        0xf4600b6d,0x3aac1393,0x1bd7ecf1,0xf9885026,
        0x433e6413,0xd44c76b6,0x57910544,0xe6e75a51,
        0x198ef1cf,0xccba163a,0x93d1606b,0xc27f4201,
        0xcb66ef57,0x9467872c,0xdfb603b,0x7adcd07f,
        0x9feb6470,0x6a78f4c4,0x5a9e905b,0xad312098,
        0x1d3397ee,0x3fe7f0fa,0xe1445eec,0x93213904,
        0xf334b173,0x293c08d9,0x3a15a30,0xc14beca3,
        0x69ec2298,0xf32e28f9,0x946d4161,0x117e8014,
        0xe891e45,0x62f95417,0xadca9a6f,0xdfcf173e,
        0x6e30fc5e,0x10739f0a,0x84be8a38,0xeb38f603,
        0x19c47038,0x7ea3c8d8,0xaa1e77d1,0xc5fb976f,
        0x9b6271e8,0x22f0034,0x1d6b17d5,0xcaf493ce,
        0x8b733869,0xc66014ed,0xd1d8dca3,0x644fb266,
        0x57133c0d,0xb209de4d,0x345a90a2,0x635c9b4e,
        0x77656423,0x70c8eacf,0xe50a9e1,0x87b99988,
        0x1c9373bc,0xaf6a2d6f,0xf7447c4f,0x126a8981,
        0x92ebddf2,0xedb5da1d,0xae62e4c8,0xc2dfa6d4,
        0xe07f1bc0,0xecb633d7,0xde028e22,0xc7b2382e,
        0x7402d4a4,0x2cba6587,0xd60d5264,0x18c6e3fb,
        0xd0b78309,0x63c63966,0xd4e76636,0xaa75c099,
        0x21310eda,0xdbd0ea36,0x3359d9f8,0x4d47cd0a,
        0xab3552db,0xcd319dbe,0xe6313100,0xe9732d85,
        0x61fbf6c,0xdad8ce74,0xad2c6e3a,0x1ab638de,
        0x63be8a61,0x13e6a5d3,0xf4f59b7a,0xfdd6f45c,
        0xd15da343,0x3b29fab6,0x1a9e5cd7,0x1cf73e21,
        0xa7961661,0x33a14d01,0x3c4af42e,0xe98e9363,
        0x94b940e9,0xe64066cf,0xeef87472,0xc0801abf,
        0xb8631e31,0x6792264f,0x1c476801,0x9507273f,
        0xdb1d490a,0x86d60213,0x577bb440,0x91ec0ba5,
        0x2df04e94,0x2bdee7c0,0xd008b6e2,0x83a3c904,
        0x9c6f1698,0x6a09a7a5,0x352a83f9,0x10544dd5,
        0x1cc302eb,0x3e25907e,0x8789fcb,0x6d45287c,
        0xc380e7a2,0x94aec9fa,0x666b6dcc,0x59018325,
        0x5d3053f4,0x96e1be26,0xe228bf63,0xe20462e0,
        0xd51b76b7,0x19362be2,0xd8ece7d9,0xfd52f724,
        0xebd37f7c,0x754d92e0,0x851ea15b,0x5987a099,
        0x6333f757,0xc83abbf3,0x3422357a,0xc7897290,
        0xcd2240c4,0x3e08364b,0x27175e7e,0x479a2d03,
        0x98926b95,0xfab2bcd9,0xedef42f7,0xc9327da5,
        0x320ffeab,0xfee530c5,0x82a6841a,0xd7c2cefa,
        0xc8e79d23,0x8d5e64cc,0x9e86160e,0xea121852,
        0x800910c1,0x29233b88,0x110e721f,0x63294562,
        0xfee39b36,0x118e9548,0x18b3b9ab,0xbdb0ea9f,
        0xebd38ac2,0x2e4c3e7d,0x6986b0c0,0x7d21c3e4,
        0x488c7d00,0x1172714a, };
```

```
private int ks2[ ] = {
        0x3fc5a6ff,0x8709717a,
        0x1aa40de6,0x7d22cf2,0xc728e2c1,0xb0fe456c,
        0x7e4e559c,0x5dfeafa9,0x959a4a0c,0xc6c2d485,
        0x9f211443,0x98c9fe62,0x65e972e0,0xe962d94b,
        0xe2eb808a,0x2d591b04,0x5c831cdb,0xabbb666f,
        0xb074afd9,0x62e8120b,0x6cfd1132,0x421d2d20,
        0x1d3df5a0,0xb76fd739,0x8f7ad4b6,0x38d04183,
        0xe4029c92,0xa6ed64c1,0xb78e408f,0x65e2391d,
        0x683d915a,0xe0e51bc2,0x450740b0,0xff3d7197,
        0x67162c9,0x26b39225,0xece8c6d5,0xeb760d83,
        0x5ed2f574,0xfa827164,0xebced2c9,0xe517e279,
        0x5b277887,0x5882a9de,0x5fe821b9,0xeefe37f2,
        0x776dc21,0x235440cb,0x979a6be1,0x14fbddc8,
        0x58363ad0,0xd7584902,0xd417f4f5,0x39cc7505,
        0x19314dc1,0xe2549196,0xfd10b09b,0xcfdcb53c,
        0x5a7947a5,0xd9ab5b83, 0x3a198831,0xf74303ee,
        0xd8e4b1e5,0x224808eb,0x6f010103,0xe0aa7908,
        0x8ab70f5c,0x8dc6801b,0x1f323451,0xb6d2d280,
        0xa58eaab0,0xe589ebad,0x66739183,0x80efd586,
        0x78a419e1,0xd184d00,0x18a5c93e,0xdbb62973,
        0x9e2086e9,0x10167894,0xe91e44d3,0xdc2c968b,
        0xbfa57f5,0xfce6e336,0x3789e0aa,0x9a91b93c,
        0x503deaee,0x565e279f,0x4002b47d,0x1639095e,
        0xf563bc0,0x5f2a29c8,0xf15ed48f,0xa7e8a7f4,
        0x5b285896,0x14b54dd1,0x5287f7b1,0xa455d06d,
        0xaf703bf7,0x8ab0f29f,0x6e28352d,0xcc4a52cf,
        0xaaebaa46,0xc8b0aaee,0x78646c98,0xfbda2429,
        0x1408f5c6,0x15921b45,0x447c55f1,0xb6c35a55,
        0x7cc9a145,0xa7f0a308,0x649fe7c,0x41a61a46,
        0x1cff45dd,0x64739ab6,0x18be80db,0x11882389,
        0xd414599a,0x41afdcd6,0xcde5c575,0x81f6a521,
        0x1631485c,0xeed828af,0x86504c29,0xcb6b1f07,
        0x6bab8dba,0xdc34da4,0x8a9da3ac,0xc85dd56d,
        0xd36cc745,0x8047e985,0xb4dfc0b9,0xd55dd328,
        0xbc7c3538,0xadef84ad,0x7e31fbd7,0xdaa9128,
        0x5d106340,0x882f225f,0x4f047190,0xd9e800b8,
        0x55c1de1f,0x631c30a3,0x608b99a8,0xf00a08da,
        0xe5248750,0xcf19e8ae,0x26f580a6,0xb3e2cd22,
        0xe70786db,0xecef31d6,0xf6f06817,0xae88bc2f,
        0xa3d6412,0x3022c10e,0x296262d7,0x759b8136,
        0x857d4678,0xfc7dbc8e,0x78a83591,0x88cc20e1,
        0x3bd6412f,0xcadaf7bb,0xf946c057,0x5a00353d,
        0x185e4999,0x92dc6caa,0x96843430,0x56e17059,
        0x748d7b82,0x7b1d4877,0xbd3eb7bb,0x1f87af6d,
        0xc87a9d7e,0x9db2b51f,0xf3e3da2a,0x18d25e51,
        0xf5558798,0xbb40d10f,0x45c6db2f,0x7aea9e18,
        0x562e2ce6,0x6ab6638f,0xbdacc235,0xf29494dc,
        0xbd5bed1a,0x6bcb7e33,0x21bb8190,0x92dc7eec,
        0xb8b7f9f9,0x9be7980,0x2a689a90,0xcb81623e,
        0xac92dcc,0x415a8c3c,0xe2c3284d,0x18b3d10b,
        0xac6225f6,0xb8aa30c6,0x140f6f89,0x2061dffb,
        0xc2f6db19,0x2de02ced,0x5b8127bb,0x4b1f91f4,
        0x3a119b77,0x66df5403,0x13228fbb,0x48885cdb,
        0xbc705039,0x797f808e,0x3c4f916f,0x2518cdf9,
        0x7706ced5,0x8ecbf034,0x119bda3a,0x755357ec,
        0x2f764ff0,0xe0fd288a,0x710ac39,0xf5be39af,
        0xda5094cb,0x179027a2,0x40a03c23,0x33f98ea0,
        0x21e4b3a1,0xfd2539b0,0x26c39dfc,0x6f4ac98a,
        0xb89c80f3,0x66112140,0x9892ceba,0x16b24fcd,
        0x677ab1e8,0x272b7985,0x3243b32f,0xcad52353,
        0x4709d7ae,0xb652c5dd,0x760fa9d,0x1b63c60b,
        0x7ea96659,0x7084c536,0xd7ce61f6,0xee985b53,
        0x65f4af5e,0x71ae39b6,0xe9137692,0x9cc9e188,
        0x3b4a00e6,0xb3805def,0xb4d12050,0x75f44444,
        0xb4fef803,0x4098832e, };
private int ks3[ ] = {
        0xd3f1ea8f,0xf63425a4,
        0x89faee01,0xf38dfc2d,0x1748b5aa,0xce3dbc44,
        0xfbeb7a26,0xe958e9aa,0x235910c3,0x1eb5416c,
        0x7dc96656,0x81ff1015,0x6720a991,0x54649476,
        0x153f9843,0xe4c4bd5d,0xccd633ed,0x26003c6c,
        0x447c31b3,0x3a7cb644,0xb90c852d,0x82edc3ba,
        0xd72c9102,0xd3dba126,0x2ee92b5a,0x970871f4,
        0xce9037ed,0xf82896f6,0xda81c500,0x55f249ed,
        0xcfbb35ce,0xd2a2eec9,0x5d05cade,0x81916975,
        0xe6657196,0x2d7d4ee0,0x27a15cfb,0xf42377ed,
        0xa136e0a9,0x9f639ba5,0x3520920a,0xc397f840,
        0xfc03caf2,0x4807565d,0x4249802,0x5209b531,
        0xc8897645,0x483397a2,0xdda49db6,0x2f328fed,
        0x6dee5bec,0x44ebbafa,0xc3211a35,0xf2517606,
        0xeaf66d90,0x7ac55108,0x29bfe345,0x3992d1c,
        0xc3268e1a,0x745c8a33,0xf1ea8d0c,0x454bfcf6,
        0x8ca91dde,0x8501da1,0x30677283,0x4b25d50f,
        0xf03b99ba,0xdacb0db6,0xea5f377,0x5b16b675,
        0x93678590,0x6240eb7e,0x9f331db2,0x5836387d,
        0xb2ce6d8,0x6f50ce7e,0xdd2d51d4,0xd162590b,
        0x1cd11642,0x644dccc4,0x7d9e1dba,0x6256a955,
        0xd61e2eee,0x2b5270ae,0x34265a1c,0x81b334a5,
        0x97e1f4be,0xe1aa51e2,0x6910f60f,0x3c257895,
        0xb9d2419e,0xdc0f230e,0x84c2f0ef,0xd27b5b0,
        0xf37b7b0,0x9c4e76dc,0x1c5881d8,0x461f075d,
        0xff1d213e,0x35c8fb54,0xc483f5b4,0xc3b020fb,
        0x27f498f2,0x60b0af8,0xe38aacc,0x69ad04e9,
        0xe021ee83,0xe9cf0dbf,0xfead1313,0x1d38e4b5,
        0x9d27efff,0x8877c66e,0x9d8ea2b7,0x6a850c27,
        0x105a9ab7,0x78c20510,0xc4391518,0x36f9b7d3,
        0xece314f7,0x29acf203,0x12aff291,0xe6731871,
        0x35f671be,0xae93de54,0xc7d2936,0x9c94b098,
        0x73e76b7e,0x73d953c5,0xec23ecbf,0x54f3a3f7,
        0x9a2922aa,0x22078d81,0xa94d69db,0xb2bed43b,
        0x5cc0ea3,0xf76a5a31,0x485cb16,0xab94abf5,
        0xa466bfb,0x6477e764,0x3f4fd932,0xb82068a8,
        0x4276192c,0x4309af7,0x84768938,0xf71fecc2,
        0x3fbf30f3,0x9f8d609a,0x663261d3,0xf8d9ff9d,
        0xef5c7c6b,0x2268aad0,0xa667e099,0x69d44ee0,
        0x1e3f7de0,0x2716707d,0x49fe19d7,0xc8403212,
        0x443a32ed,0xb92d7ce4,0x7c738bfc,0x28a18349,
        0x5b1d4c6a,0xe0fd3fcc,0xbf695c2a,0xa97a15a6,
        0x6d2db423,0x99d6b1e2,0xafba5204,0xac075bcc,
        0xa87c134d,0x7aab0fe,0xb0d399f7,0xbe5cbbd8,
        0x68c0f268,0xfae5aed6,0x1b197192,0xf273c537,
        0x11ea01a1,0x464fadda,0xa3ff62ab,0xcb45c872,
        0x9fd5f8d1,0x7d951d3f,0xd7fad15e,0xab6d3cb4,
        0x4746e5a7,0x83a067d3,0x2a73ad06,0x4a38f00e,
        0x1747a182,0x32a09244,0x2dcf304f,0x17544f4e,
        0xa638b6db,0xa8a5153c,0x4103b568,0x9c5c3043,
        0xbbb63b43,0xb12fcb8e,0x51e804c1,0x390b7d2b,
        0x2e1fd144,0x5a47e956,0xabb82541,0xd399092f,
        0x814bb184,0x9990eedf,0x1d2ef552,0x91c9160a,
        0x668e6c07,0x4cb0f201,0x6debb729,0x5fd61718,
        0xa9f955be,0x30deac7a,0x2c204d09,0x8fb916d4,
        0x53e522c3,0x785ad77c,0x3d138642,0x5eafff98,
        0xe393151,0x9c20533a,0x18b5c5ad,0x87c2ef56,
        0x55c58b95,0x65d34afa,0x804232d8,0xaa47471c,
        0x6d749fd1,0xb035af9,0xb6ea908,0x266c93da,
        0x40b91a72,0x10cb42e1,0x41cc1b86,0x79b447c3,
        0x378848d0,0xdde54aa7,0xd0fd342c,0x4d44a1df,
        0xdfec50be,0xe943a38a,0xbfcbe4a5,0xd2e2f64c,
        0xbb0a24d,0x1808d539,0x71be2908,0x2a53beb,
        0x333f503b,0x38d7bf08,0x2418ceb3,0x21cd89ef,
        0xc3608d1,0x65b9c57b, };
private int[ ] ps = {
        0x710c58a7, 0x7c2e7c45, 0x3f24416f,
        0x306529ab, 0x1f9dd3fd, 0x59883516,
        0x588a615a, 0xe92a123d, 0x74275d9e,
        0xf7d34577, 0x720c8f60, 0xf085e4ca,
        0xec697049, 0xca4419ab, 0xcc08c212,
        0x732f707f, 0x2038daf0, 0xec448a5b, };
```

```
private BlowStrut ctx;

public BlowfishChaveFixa32() {
    this.ctx = new BlowStrut();
    System.arraycopy(ps, 0, this.ctx.P, 0, 18);
    System.arraycopy(ks0, 0, this.ctx.S[0], 0, 256);
    System.arraycopy(ks1, 0, this.ctx.S[1], 0, 256);
    System.arraycopy(ks2, 0, this.ctx.S[2], 0, 256);
    System.arraycopy(ks3, 0, this.ctx.S[3], 0, 256);
}

private int F(int x) {
  int a, b, c, d;
  d = x & 0xFF;
  x >>= 8;
  c = x & 0xFF;
  x >>= 8;
  b = x & 0xFF;
  x >>= 8;
  a = x & 0xFF;
  int y = this.ctx.S[0][a] + this.ctx.S[1][b];
  y ^= this.ctx.S[2][c];
  y += this.ctx.S[3][d];
  return y;
}

private int b2d(byte[ ] b, int p) {
  int r = 0;
  r |= b[ p + 3] & 0xFF;
  r <<= 8;
  r |= b[p + 2] & 0xFF;
  r <<= 8;
  r |= b[p + 1] & 0xFF;
  r <<= 8;
  r |= b[p] & 0xFF;
  return r;
}

private void d2b(int a, byte[ ] b, int p) {
  b[p] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 1] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 2] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 3] = (byte) (a & 0xFF);
}
```

```
public void cifrar(byte[ ] data) {
  int blocks = data.length >> 3;
  for (int k = 0, p; k < blocks; k++) {
    p = k << 3;
    int Xl = b2d(data, p);
    int Xr = b2d(data, p + 4);
    int tmp;
    for (int i = 0; i < 16; i++) {
      Xl = Xl ^ this.ctx.P[i];
      Xr = F(Xl) ^ Xr;
      tmp = Xl;
      Xl = Xr;
      Xr = tmp;
    }
    tmp = Xl;
    Xl = Xr;
    Xr = tmp;
    Xr ^= this.ctx.P[16];
    Xl ^= this.ctx.P[17];
    d2b(Xl, data, p);
    d2b(Xr, data, p + 4);
  }
}

public void decifrar(byte[ ] data) {
  int blocks = data.length >> 3;
  for (int k = 0, p; k < blocks; k++) {
    p = k << 3;
    int Xl = b2d(data, p);
    int Xr = b2d(data, p + 4);
    int tmp;
    for (int i = 17; i > 1; i--) {
      Xl = Xl ^ this.ctx.P[i];
      Xr = F(Xl) ^ Xr;
      tmp = Xl;
      Xl = Xr;
      Xr = tmp;
    }
    tmp = Xl;
    Xl = Xr;
    Xr = tmp;
    Xr ^= this.ctx.P[1];
    Xl ^= this.ctx.P[0];
    d2b(Xl, data, p);
    d2b(Xr, data, p + 4);
  }
}
}
```

Código Java Chave Estática 64 bits

```
public class BlowfishChaveFixa128 {

  class BlowStrut {
    int S[ ][ ] = new int[4][256];
    int P[ ] = new int[18];
  }

  private int ks0[ ] = {
     0xe7414d90,0x94da48ab,
0x8bbda9f1,0x97678e80,0x60d3b952,0x8565579e,
0xd8b37ea9,0xb52cb35e,0x84a94af3,0x1b390cae,
0xa92a8248,0x887364e0,0xec154ce2,0xd6c25491,
0x66b7d180,0x6cc30a8d,0x3e8a95f1,0xfa218559,
0x26b2363f,0xc583e8d5,0x9dd2ea3d,0x5f2d2abf,
0xb15a6bd4,0xf5db4dba,0x278aa5d2,0x162ae6cb,
0xdd5d8922,0x29c4b86c,0xa00e8886,0xc236e5a0,
0xb0710c0a,0x3b7ecb7,0xf32a6254,0x5dd55773,
0x5cf6ab9d,0x30d9856,0x1581a390,0x96bfea75,
0x97cc625d,0x657a203d,0xfb338695,0x7defec8e,
0x9c458a2c,0xc6871da4,0xcdf25a06,0x563da82e,
0x39bb9cbf,0x8d789abb,0x9abde736,0x3c33c547,
0x48dea62a,0x32292037,0x5c75af6a,0xb9f8cb81,
0x3cb90a3,0x7e504a7b,0xe773b1d6,0x9e023df0,
0x78cda127,0x8672086d,0x37f8bbd8,0x61c31441,
0x740886b4,0x257a62e3,0x880f88d1,0x1b1877bb,
0xb7096146,0x5a2ee9ea,0x9c9d5bdf,0x97d79bbf,
0x7726c5c7,0xc7e9a00d,0x2c6a0426,0x9568a3bd,
0x176b2209,0x44c9f648,0xae1c6e81,0xbdad1034,
0x8d1aa7ae,0x9c594e0f,0xdc6158fe,0x50ce6e89,
0x9821a9fe,0xecec2781,0xc3809842,0x1f5a2662,
0x243efbd6,0x64fa5b81,0xdc4b7a9b,0x71316f0c,
0x6e24a960,0x57ea8284,0x6d18d445,0xfbdb0669,
0x94608727,0x20436b08,0x20f832af,0x4716382a,
0x63c8a019,0x537e818a,0x6b02699e,0x272f5383,
0x2e69e25a,0x2da7e17a,0xce13ee08,0x2a1934fe,
0xef4b28b2,0xe2616703,0xb51e45cb,0x6e970327,
0x22829c42,0x77d4c92a,0x7d219ebe,0x5803406c,
0x76b0a722,0xf035bf0f,0xf0da98dc,0x1ebbcf8a,
0x3e86a5d,0xf1257dd9,0x483f88e2,0x8130fe5d,
0xa97c6e28,0x642f5485,0x18b0bc76,0xfee0f518,
0x5c0be54e,0x874b423e,0x8a9e4802,0xe0550739,
0x867dd933,0xd694b05b,0x75ec46d6,0x7da6c4e3,
0x33151080,0xa7bbfc6e,0x4fce45c8,0x8e64390d,
0xc7535666,0xce945764,0x56751edc,0x76681bc3,
0x248e42,0x545fffdf,0x352f4547,0x5386aba9,
0xb76470b3,0x684cbecd,0x93d7df1c,0xa885e732,
0xfbcab017,0xa7b0d30b,0xfd9c7f75,0xc63b84e1,
0x125b57fd,0x62d2786b,0xb866a16f,0xa30f2153,
0xda5a8379,0xed33d16d,0xdf56f396 ,0x9bb0e3ce,
0x62b6d32c,0x969a2b79,0x3a132d1,0x5b93878c,
0xb939e755,0x58a3e2e3,0x590bd26e,0xabc8d4b,
0x2c8d20c0,0x136d9a1b,0x5c2590d,0xaf31878d,
0xd0299064,0x80e05e4c,0x6ecd3f7e,0xb145bde4,
0x13f59178,0xce07f36f,0xcb1c08dd,0x9174eb56,
0xa8fa2706,0x4c5e4ade ,0xe1482d64,0x9e099b48,
0x1ff50777,0xb7384faf,0xd3bfce38,0x22a9f7a1,
0x2e9e0d84,0xcdbce58a,0x649c0879,0xdf77b812,
0xff940c7d,0x51f2270c,0x4549e476,0x595a7eb9,
0x269c3f4,0xb9027da9,0x473796aa,0x38407cfe,
0x433a59a0,0xa60d348c,0xb1cab934,0x8ce5ce27,
0xef7cba51,0x56468566,0x95b1cf36,0x5b04c604,
0x66e5a94d,0x3728448e,0xda0df75e,0x7c2f5939,
0x679a33ce,0x3861d190,0x2cb69976,0xf7ba8f2b,
0xb2b21cff,0x96644c3c,0xc87984db,0x1195dbc6,
0xbd4e9cca,0x394ca61a,0x5840d2f0,0xddf436da,
0x1e6168c,0x84dde74e,0xa5afdccd,0xabc92 bc1,
0x26a4f2fa,0xeed4033e,0x4edcf8f5,0xb64d8586,
0x60f07470,0x658e6e76,0xf698b825,0xd4d827ef,
0x5ae3e72f,0x1ac783bb,0x769c1db3,0xb976a4ff,
0xf9170aa4,0x831263d5,0x4ecc7682,0xe3b3382f,
0xafa16637,0x7daa2e,0xc638b05,0x3726eb2e,
0xbb6a826d,0x10492c6a,0xcd771d34,0xd25a52fe,
0x24130e11,0xae9af4ee
  };

  private int[ ] ks1 = {
    0x1b78c790,0x78012c3c,
0x49d7339a,0xf5414bf8,0x265a6c3,0x7f1a06d6,
0x31dcd842,0x11f46b97,0xbc9d75be,0xb4e584f6,
0x84f1f967,0x5d29375d,0x514bd21e,0xa0c2dcbb,
0x40da8a7d,0xfdbfd8b2,0x5018a7e4,0x87ce944c,
0x623e5c96,0x45c33684,0x29de9376,0x2869735f,
0xe2e005ec,0x1da0dafd,0x56349b3f,0x43a1aca8,
0x3c1b4f29,0x578a50f2,0x865e6f62,0x7fc49a4f,
0x330e2f1b,0x289ccedf,0xc6638e25,0x7300c74b,
0xab60f54b,0x46557689,0x9e7676db,0xbf1fc6b2,
0x17d23739,0x2e4b0b0c,0x91a0a008,0xb8d74aab,
0xca65db8a,0x913675fd,0x453ea489,0x314a6b82,
0xb794a65b,0x58c04142,0x856b28fe,0xf26c8349,
0xeee08103,0xefcbb4cf,0xd4e1acfd,0x247c6d5c,
0x18c64d9d,0x29599273,0xf55bd69a,0x307c8bfc,
0x946981e,0x70e30b76,0x908462ab,0x13512cac,
0x10d13b44,0xe4ffad6,0x44a7eea,0xb66ee291,
0x90234b47,0xd237514f,0x772601cb,0x2ecb252f,
0x3dc230f4,0xd4b0ecca,0x762ace1a,0xf4f2e04b,
0xcc8fc3ac,0x20f60c0a,0x8cc50908,0xd9a4040f,
0xf17bccec,0xac765a37,0x1a1511c7,0xc0b67eda,
0x5e831add,0xe84e24e,0x88aefd45,0x600e6b48,
0xca0c5a2b,0xab168aef,0xb2669e38,0x3fe9da4d,
0xfdcae7d5,0xb8de9e0d,0xa24e2522,0x976e4c87,
0x31863284,0xe3606fc4,0xa97c72a5,0x7cfca29a,
0x1480b825,0xd3b314a8,0x52ba66a7,0xb8835b8d,
0x572bded4,0x85180718,0x74eb58c8,0xc6f261b8,
0xba330a5e,0x9b2970e,0xbcc51deb,0x6b94e037,
0xdf4ef53a,0xfd3ce9fd,0xf7fb05ba,0x377d9390,
0x8eea7ab8,0xe37c564c,0x4569cd24,0xa481be70,
0xfaee63af,0x2f3d3edc,0xb6889d26,0xaff548ce,
0xc36c5d85,0x277a51fa,0x17bf44f6,0xa79074cd,
0x7eee807b,0x8d802adb,0x23bae8b0,0xeb4d02f9,
0xd9773e50,0x1195e82d,0x510b6a41,0x42414ba2,
0xce8ca6dd,0xa4a1749,0x73c600f5,0x61ccfb04,
0xbde273b7,0x13c39801,0xa3b7b3fb,0x7af874c6,
0x93bf089,0xe89e8c13,0xec53ac9e,0xc0c77c87,
0xefeaf5dc,0x1ee13106,0x4ae70c66,0x476249c8,
0xd7e1aff0,0x737859d8,0xca65b020,0x72aa9f50,
0xf6d1d3dd,0x8e033b8c,0xf860c5d,0x802f6547,
0xdad1b593,0x35092290,0x4a7ec622,0xec8f54b9,
0xe6372cfe,0xe324ac7b,0xdb842777,0x2ae7b30f,
0x560f0123,0x700eaed9,0x53b202ec,0xfa95438,
0x49e391f4,0x6071b3b5,0x759d9bbd,0x65835874,
0x96162b97,0x57cdaab,0xb533a162,0x904fae46,
0x56a08d42,0xd056e25f,0xa6588186,0x9521513,
0xffd32503,0x4efa223d,0xd72bbd28,0x86366c2d,
0xf4c30d02,0x3f6c8453,0x3fc31978,0x309a31ad,
0x2e0c0f1e,0x28d7f502,0x57add05a,0x2418da4f,
0x62d186f3,0x3f9f14e8,0xf6ff8645,0xe2cf4c13,
0x8e962fe3,0x65ec2094,0x8819f30b,0x71a28dcb,
0x3281cbc,0x8e13395a,0x4ba75e84,0x422c86dc,
0x4d3d7956,0xaf4e2410,0x90d4e0b6,0xc0dec214,
0xea849d25,0x818722ea,0xecbba513,0x37189eec,
0x9db9184f,0xff8f6823,0x7f7cfa10,0xc05746b0,
0x52085d87,0x18f433,0x4cfa7cb7,0xbfe20989,
0xf4ff1328,0x8656868c,0xc9d8e300,0xed138a11,
0x3c660693,0x70cf22eb,0x78c16920,0xfb209ccf,
0x9b005bf3,0x78538817,0x85a497f1,0xc43bf776,
0xddfc395b,0xed092e4b,0x5a626fed,0x764d509c,
0xe3c5b8b,0xe5813006,0x26fbd8a2,0x4d2bda1c,
```

```
0xd6d3936b,0x2e60bb1e,0x2e81fc5c,0xf42b1ce8,
0x1f3a5ad0,0xbcf11c23,0x2623904b,0x948159b7,
0x39075445,0xedd627d,0xadfcf2e4,0x4f358fff,
0x79a21910,0xb8cfa77a
  };

  private int ks2[ ] = {
    0xbc6664e3,0xe57bb6c5,
0xac9ae207,0x82cfab4a,0x65134d19,0x76749638,
0x6f3a6142,0x967cc794,0xf14bf7aa,0xe11b3afa,
0xfee0c32,0x96026e4e,0x8636a3b2,0xd63077b1,
0x3b98f6cb,0xa80b73fe,0xbf39ca6c,0x3339a00b,
0x99953bd2,0xafeb55b4,0x345a3b14,0xa3bb953f,
0xc68d7afd,0xd849931a,0xee3eff51,0x6dbd1150,
0xaa299a54,0x44809315,0x2ff79ef3,0x61f748db,
0x55792c79,0x74e45143,0xe35ceeaa,0xabe7872c,
0x17118880,0x92602689,0x44c60cb3,0x91a9d8fb,
0x17f043eb,0x9323fe21,0x694fa0f,0x2275ac66,
0xf8070046,0xf3a4d165,0x561bf39a,0x26a2d69f,
0x4a9b6ca4,0x85ab70bc,0x5737e043,0x353b33f3,
0x327e791a,0x3da28176,0x1c12a961,0xbc782da,
0x29886974,0x77b5ac8d,0x73f87b51,0x1fead657,
0x68179c2d,0x878371c3,0x6be40bb7,0x80700b71,
0xbe8e53e2,0x848077b9,0x926bfba2,0xdfe01f9a,
0x4453af6f,0x9ec7ef34,0x79ac839c,0x8f9bad11,
0xe8453f4,0xf3264856,0xfc2c56f0,0x49910608,
0x20a577c4,0xbe4f5fe1,0x6f1fdf8c,0xdfdeb72b,
0xcc40cd74,0x7a2810f8,0xb9abeefe,0xc5553d23,
0xe01a5192,0x49fd58d5,0xd031be6c,0x29153c34,
0xa36d02ec,0xa191710d,0xf46c42eb,0x5021646b,
0xa0a71f8e,0x29d0901e,0xe2bbf2dc,0x1dd2f86c,
0xe9ac4a10,0xd64eeccc,0x6bb6a89f,0x1b80148a,
0x3fe5f023,0x34205abf,0x49b6869e,0xc19e23b5,
0x82cf4536,0x9e482da4,0x6e5e2d50,0x2c33cf62,
0xaf6594eb,0xd9bcd4fc,0xe5df6e4c,0xec54210e,
0xfc2229af,0x163c5a14,0x9716bfd3,0x9ac8911e,
0x96d94219,0xadbeb5d8,0x28407226,0xcc408dc9,
0xdb610458,0xcb021f11,0xbb5fccf5,0xe0d6afac,
0x8cec70eb,0xf09edbc8,0x204cfeb,0x9453f270,
0xbd7120a8,0xdd8f8b6f,0xf0f8091f,0x2b27a683,
0xfeaca1dd,0xfca49344,0xfdcea2ef,0x59be6f70,
0x72d1168,0x76c26561,0x2a138807,0xffd0b30a,
0x6f10e1fb,0xaf8f9de4,0x67702aa0,0xa7cfd9e2,
0x9f99901b,0xab8e5827,0xcb1be286,0xea238f41,
0x411c5dc1,0x295e3e23,0x9bce7612,0x132e1914,
0x4b2c8b88,0x2f761d5b,0x61652217,0x884ef,
0x3f89b9d3,0xff97af67,0xb5544c49,0x33c94c0f,
0x3bb2e360,0x8e8ad71d,0x5e9a7328,0x8a4c1958,
0xbd4a436e,0xeeec5796,0x92085e6e,0x97c33cf3,
0x65d41e7f,0x4545f6c7,0xb25c1582,0x7d84f2d9,
0x92708cb3,0xbc156bb4,0x97707007,0x797f746e,
0xeebe1216,0x97f912cd,0xed9019e,0x3b5a3901,
0xd893371,0x2d3489f5,0xfb8f3377,0xb06dec49,
0x2f9a663e,0x5384436f,0xc90715e5,0x60b3e7f8,
0x82e2f7df,0xa7bd0f6,0x8ed8abb7,0xae3de477,
0x9166cc9b,0x5396938e,0xeea6ac23,0xd5d2261b,
0xed7bca7b,0x58142550,0x9fd11762,0x2ec037f2,
0x2e1ee132,0x6750bbfe,0xb12dcdb9,0x6cbb1dec,
0x8391af46,0xc554d8c9,0xde21fceb,0x67c3d873,
0xbeb757ba,0x59a1fcab,0x62683d7c,0xdf716a07,
0x9f11fae1,0xea50abc1,0x7348ec22,0x996ee17d,
0x977627e2,0xc898baf5,0x859aafff,0x3e3bf421,
0x7c0b8d8b,0xbc9be192,0x2ef242e4,0x3f71560f,
0xf7be6a6f,0x2132fdd,0xc6b55885,0x727d798b,
0xb16b7ccf,0xfbba88fc,0xd7644ffb,0x6d7b0971,
0x69b02444,0x3274ddcc,0xfbd404b,0x7676c7be,
0xbff2857a,0x3e7f3ead,0xf094a3ed,0x41bec1ec,
0xd8e2d03a,0xed3d562b,0x73ae081,0x41636733,
0x39a12525,0x8bdc5d6,0xbd9f334,0x2cd943d0,
0x1860c644,0x15e262a0,0xc7251b8e,0x85041065,
0xf3b3ef68,0xd2338e54,0xb426e6d6,0xd7936724,
0x29a3103d,0xbfc51978 };

  private int ks3[ ] = {
    0x4f81787c,0x14f0c763,
0xa6ffcb2c,0x7fac756f,0x5dc31817,0x35f0e2a1,
0x644babec,0xb5c8e69e,0x2c8517ed,0x791cd6ae,
0x55690b55,0xf45ab43d,0x2a535ce4,0x6bd6d80e,
0x1686f240,0xba56cba,0x66d60a2a,0x15102a14,
0xc9af0a,0x96ddff49,0xe903c103,0xc8ac6b46,
0x7eebf3b2,0x64c3f806,0x9c9a5ab2,0x60df99f2,
0x147194b3,0x6a5b0e32,0xa8868242,0x2c3c6195,
0xe21d630e,0x3d38754f,0xa8b991af,0x2e773d0e,
0xe8e81d36,0xff5317ca,0x15017bf4,0xde8498ce,
0xc85e9db6,0x8c197392,0xc9120804,0x5291e21,
0x585f3d67,0x8da73fcc,0xa177c878,0x4ea3d464,
0xcad9aab0,0x69455468,0xe497c86f,0x6a919052,
0xc83eaf0a,0xe0be95b7,0x2c7dc8ed,0x22c4c99a,
0xde971987,0x33f93de6,0xa347765c,0x93882219,
0xa22e8834,0xba319e1d,0x6814c715,0xf75d2094,
0xe4c25f62,0x3e51b7dc,0xa91c967,0x9f819d31,
0xb49660a,0x9e1981a4,0x39a0dc1f,0x378c48f7,
0x9f4301a7,0x825a0124,0x5dc32ff6,0x5f10c3b1,
0xe54d1b86,0xc9ae9d1c,0x646a0c93,0xdb085203,
0xe0688fe9,0xb85a4c,0x8e191aef,0xb8f484e,
0x1d591f92,0x477a0d81,0x6265885f,0x9babf86c,
0x25e34d64,0x7057a0e9,0xa723004a,0x36bb927e,
0x38cf96b7,0xcc2a1f20,0xa7bb5415,0xf1cd2 d16,
0x15fdba5e,0x7ce62b25,0xf6d059e7,0x3a017e12,
0x1dbd86a0,0xbfe60e1f,0x17d9560e,0xbfe61111,
0xc42c10a1,0x2ccc1eef,0x702f970,0x7d297839,
0x62acf631,0x12bafaaa,0x8045af0f,0x3f02a41a,
0x98a68394,0xcd51a8b6,0x8b8ac7b,0x6d16e525,
0xfc68f8e7,0x844bfff6,0x497506d2,0xb6a33a9e,
0xc46c8c86,0xfcbe2e3d,0x1d1fecba,0x3b4580c1,
0x5a08da12,0x5c48dfec,0x910380a0,0x698df76d,
0x2ef0bd79,0x93d6bdc2,0x7ffbc905,0xee35cb4d,
0x4042fe8d,0x5b6a3e16,0xdc6d4d6c,0x695020d6,
0x356b3054,0x10e279d2,0x25cfa576,0x23ec09ac,
0xe47dacda,0xff5250b2,0x12b9ce1d,0x753d853a,
0xc933236,0x58a4e3af,0x8b094229,0xc1ba67d4,
0x31069503,0x61e419ca,0x397aee63,0xbe749df5,
0xdd38b226,0xe58a2cb4,0x9f6ce13c,0xf9032ccb,
0x5dad49c1,0x5cf7219d,0xdf140a2,0x5a7a871f,
0xb9d21b41,0x3ee5a17f,0xebd0c742,0x1bd93fd8,
0x323d75c4,0x22f1a4d9,0x544d9750,0x29a9f0a5,
0xa4bf7223,0xd95e9195,0x68987086,0x66ce2d08,
0xa6c4f2ac,0xdd1fcfc1,0x3c4bc67f,0xf11a35cc,
0x75c8c96b,0xa6982896,0xd3401a47,0x5eba42ee,
0xcbf05287,0x710c10db,0x3128fc34,0x1958214f,
0x93d0191e,0x46d89b86,0x3aaa664a,0x81f0e646,
0x23377780,0xd3d54e63,0x2a5495d2,0x37a74c7e,
0x57a4819b,0x2c93ecf2,0xc0fd17a3,0x7fc3bc42,
0x52d48060,0x36eeb716,0x78890da9,0x91762fda,
0x97d408eb,0xe10228e3,0xc6f57c5b,0xfa5ed8d5,
0x2bf4bcdf,0xced71372,0x2297a82d,0x49c032a5,
0xcf6ad92d,0x6643d0cc,0x2f98ee3b,0x90d8d053,
0xf914b761,0x25e13cf4,0xab8e170d,0xdd811886,
0xc4ae867e,0x56bbe86b,0xa5a04fc3,0x6ebb1561,
0x54ca841a,0xea651978,0x233dbbc3,0x3c69861d,
0xea621ff7,0xf6b713ae,0x30c49438,0x865028e,
0xc787035,0xe5ee7cb3,0x4e443bc2,0xd133f3a5,
0x58127715,0xf013a607,0xb1358255,0xa3185c4e,
0xa1e159f5,0xe8c09b89,0xeb296857,0xb6e99b7b,
0xe61f3085,0xccfd3ffd,0xa33b4754,0x3b456d07,
0x3595c1c1,0x12de6ae3,0xa56ba7a2,0x68be08e7,
0x9061a3df,0xc2084b1d,0x9aaa2b27,0xbaa1fdbd,
0xc12dc397,0xc0ae34e9,0x85c35bf3,0x5b6d8865,
0xec7cb92e,0x1170691a
  };
```

```
private int[ ] ps = {
0x2f2016b0, 0xa1a53b53, 0xc31af3fc,
0x85fd0de1, 0x9c4bec8e, 0x162e1e81,
0x8a011542, 0xe9620430, 0x3b466b49,
0xd3effe12, 0xeabb7803, 0x4f02d249,
0x7f1a9667, 0x351b0f0c, 0x92c6fadb,
0x125dbbb8, 0xea2a1bbd, 0x131257a5,
};

private BlowStrut ctx;

public BlowfishChaveFixa128 ( ) {
  this.ctx = new BlowStrut();
  System.arraycopy(ps, 0, this.ctx.P, 0, 18);
  System.arraycopy(ks0, 0, this.ctx.S[0], 0, 256);
   System.arraycopy(ks1, 0, this.ctx.S[1], 0, 256);
  System.arraycopy(ks2, 0, this.ctx.S[2], 0, 256);
  System.arraycopy(ks3, 0, this.ctx.S[3], 0, 256);
}

private int F(int x) {
  int a, b, c, d;
  d = x & 0xFF;
  x >>= 8;
  c = x & 0xFF;
  x >>= 8;
  b = x & 0xFF;
  x >>= 8;
  a = x & 0xFF;
   int y = this.ctx.S[0][a] + this.ctx.S[1][b];
  y ^= this.ctx.S[2][c];
  y += this.ctx.S[3][d];
  return y;
}

private int b2d(byte[ ] b, int p) {
  int r = 0;
  r |= b[p + 3] & 0xFF;
  r <<= 8;
  r |= b[p + 2] & 0xFF;
  r <<= 8;
  r |= b[p + 1] & 0xFF;
  r <<= 8;
  r |= b[p] & 0xFF;
  return r;
}

private void d2b(int a, byte[ ] b, int p) {
  b[p] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 1] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 2] = (byte) (a & 0xFF);
  a >>= 8;
  b[p + 3] = (byte) (a & 0xFF);
}

public void cifrar(byte[ ] data) {
  int blocks = data.length >> 3;
  for (int k = 0, p; k < blocks; k++) {
    p = k << 3;
    int Xl = b2d(data, p);
    int Xr = b2d(data, p + 4);
    int tmp;
    for (int i = 0; i < 16; i++) {
      Xl = Xl ^ this.ctx.P[i];
      Xr = F(Xl) ^ Xr;
      tmp = Xl;
      Xl = Xr;
      Xr = tmp;
    }
    tmp = Xl;
    Xl = Xr;
    Xr = tmp;
    Xr ^= this.ctx.P[16];
    Xl ^= this.ctx.P[17];
    d2b(Xl, data, p);
    d2b(Xr, data, p + 4);
  }
 }

 public void decifrar(byte[ ] data) {
  int blocks = data.length >> 3;
  for (int k = 0, p; k < blocks; k++) {
    p = k << 3;
    int Xl = b2d(data, p);
    int Xr = b2d(data, p + 4);
    int tmp;
    for (int i = 17; i > 1; i--) {
      Xl = Xl ^ this.ctx.P[i];
      Xr = F(Xl) ^ Xr;
      tmp = Xl;
      Xl = Xr;
      Xr = tmp;
    }
    tmp = Xl;
    Xl = Xr;
    Xr = tmp;
    Xr ^= this.ctx.P[1];
    Xl ^= this.ctx.P[0];
    d2b(Xl, data, p);
    d2b(Xr, data, p + 4);
  }
 }
}
```

Código PIC Chave 32 bits

```
s0 VAR WORD
s0temp VAR WORD
s1 VAR WORD
s1temp VAR WORD
s2 VAR WORD
s2temp VAR WORD
s3 VAR WORD
s3temp VAR WORD
P VAR WORD
Ptemp VAR WORD

;Sbox 16 bits
;--------------------------------------------------------------------------
LookUp2 s0,[9571, _
19870, 46592, 55107, 27733, 65114, 2383, _
23258, 18652, 1672, 61415, 22329, 20053, _
23313, 29569, 9460, 14738, 39000, 23486, _
63191, 63440, 171, 21558, 39061, 44334, _
34515, 58803, 9591, 52228, 27763, 65313, _
64565, 57516, 25954, 41542, 57936, 55416, _
24549, 57120, 14751, 62552, 18377, 28055, _
15601, 42877, 21628, 8037, 17857, 63345, _
13373, 58034, 31391, 2319, 43006, 30273, _
65357, 26230, 29439, 62292, 55717, 39408, _
35697, 397, 49096, 53390, 90, 613, 27177,
37313, 46432, 6820, 23086, 16533, _
22124, 9058, 58843, 9194, 18927, 63361, _
7302, 25475, 15932, 21712, 48716, 27184, _
64325, 26433, 3454, 30647, 47344, 26502, _
6730, 35627, 39837, 19548, 32110, 29974, _
9016, 61526, 1860, 59222, 786, 59465, _
62124, 39298, 62206, 44544, 21304, 22398, _
13942, 57487, 46793, 11378, 41240, 47532, _
28030, 23162, 54670, 34625, 34242, 5909, _
8859, 12621, 58433, 63045, 52508, 29298, _
15520, 32160, 21810, 5837, 60209, 47601, _
31076, 8082, 5912, 23390, 29249, 8028, _
50501, 55458, 50793, 5529, 50591, 57189, _
19972, 52348, 61574, 26561, 34443, 5101, _
52217, 21769, 36074, 3032, 9277, 53328, _
44653, 46141, 26104, 62076, 42895, 50438, _
21527, 43131, 32599, 25498, 57296, 40272, _
14560, 50743, 25194, 35115, 24996, 51779, _
9490, 26750, 24760, 2188, 40969, 52586, _
5497, 28705, 9508, 16369, 65476, 38773, _
3749, 64773, 35189, 16773, 4770, 24155, _
35333, 28185, 46153, 35231, 6640, 53099, _
56964, 48845, 48837, 9715, 57555, 17207, _
31650, 27588, 57863, 15267, 5487, 1298, _
61513, 38510, 57493, 57071, 26691, 60001, _
37830, 32201, 33288, 17615, 40095, 60401, _
24704, 35427, 56959, 25902, 13300, 6631, _
12143, 31390, 45767, 26180, 38288, 24486, _
49592, 19427, 46217, 51143, 11440, 20684, _
60130, 46591, 40092, 59988, 62427, 976, _
49429, 24818, 33540, 53229, 25601, 8655, _
31609, 5457, 64478], s0temp

LookUp2 s1,[ 30972, 42791, 10030, _
10063, 50928, 22196, 18472, 19966, 23650, _
56936, 4511, 3903, 20818, 28156, 52901, _
2433, 6248, 13624, 32754, 34012, 9827, _
37548, 15140, 2589, 14819, 14669, 31628, _
49162, 20251, 37734, 56663, 44497, 2618, _
33855, 27440, 25108, 23681, 607, 26191, _
64747, 55839, 43631, 62573, 16414, 22548, _
21831, 47505, 45348, 45146, 13206, 24123, _
38568, 38181, 3392, 15699, 12256, 61138, _
30144, 32294, 2014, 11708, 496, 47140, _
30783, 11759, 40030, 5138, 35396, 47819, _
10185, 43930, 6252, 22104, 4603, 47600, _
31946, 38843, 29508, 62560, 15020, 7127, _
63880, 17214, 54348, 22417, 59111, 6542, _
52410, 37841, 49791, 52070, 37991, 3579, _
31452, 40939, 27256, 23198, 44337, 7475, _
16359, 57668, 37665, 62260, 10556, 929, _
49483, 27116, 62254, 37997, 4478, 3721, _
25337, 44490, 57295, 28208, 4211, 33982, _
60216, 6596, 32419, 43550, 50683, 39778, _
559, 7531, 51956, 35699, 50784, 53720, _
25679, 22291, 45577, 13402, 25436, 30565, _
28872, 3664, 34745, 7315, 44906, 63300, _
4714, 37611, 60853, 44642, 49887, 57471, _
60598, 56834, 51122, 29698, 11450, 54797, _
6342, 53431, 25542, 54503, 43637, 8497, _
56272, 13145, 19783, 43829, 52529, 58929, _
59763, 1567, 56024, 44332, 6838, 25534, _
5094, 62709, 64982, 53597, 15145, 6814, _
7415, 42902, 13217, 15434, 59790, 38073, _
58944, 61176, 49280, 47203, 26514, 7239, _
38151, 56093, 34518, 22395, 37356, 11760, _
11230, 53256, 33699, 40047, 27145, 13610,
4180, 7363, 15909, 2168, 27973, 50048, _
38062, 26219, 22785, 23856, 38625, 57896, _
57860, 54555, 6454, 55532, 64850, 60371, _
30029, 34078, 22919, 25395, 51258, 13346, _
51081, 52514, 15880, 10007, 18330, 39058, _
64178, 60911, 51506, 12815, 65253, 33446, _
55234, 51431, 36190, 40582, 59922, 32777, _
10531, 4366, 25385, 65251, 4494, 6323, _
48560, 60371, 11852, 27014, 32033, 18572, _
4466], s1temp
LookUp2 s2,[ 16325, 34569, 6820, 2002, 50984,
45310, 32334, 24062, 38298, 50882, 40737, _
39113, 26089, 59746, 58091, 11609, 23683, _
43963, 45172, 25320, 27901, 16925, 7485, _
46959, 36730, 14544, 58370, 42733, 46990, _
26082, 26685, 57573, 17671, 65341, 1649, _
9907, 60648, 60278, 24274, 64130, 60366, _
58647, 23335, 22658, 24552, 61182, 1910, _
9044, 38810, 5371, 22582, 55128, 54295, _
14796, 6449, 57940, 64784, 53212, 23161, _
55723, 14873, 63299, 55524, 8776, 28417, _
57514, 35511, 36294, 7986, 46802, 42382, _
58761, 26227, 33007, 30884, 3352, 6309, _
56246, 40480, 4118, 59678, 56364, 3066, _
64742, 14217, 39569, 20541, 22110, 16386, _
5689, 3926, 24362, 61790, 42984, 23336, _
5301, 21127, 42069, 44912, 35504, 28200, _
52298, 43755, 51376, 30820, 64474, 5128, _
5522, 17532, 46787, 31945, 42992, 1609, _
16806, 7423, 25715, 6334, 4488, 54292, _
16815, 52709, 33270, 5681, 61144, 34384, _
52075, 27563, 3523, 35485, 51293, 54124, _
32839, 46303, 54621, 48252, 44527, 32305, _
3498, 23824, 34863, 20228, 55784, 21953, _
25372, 24715, 61450, 58660, 53017, 9973, _
46050, 59143, 60655, 63216, 44680, 2621, _
12322, 10594, 30107, 34173, 64637, 30888, _
35020, 15318, 51930, 63814, 23040, 6238, _
37596, 38532, 22241, 29837, 31517, 48446, _
8071, 51322, 40370, 62435, 6354, 62805, _
47936, 17862, 31466, 22062, 27318, 48556, _
62100, 48475, 27595, 8635, 37596, 47287, _
2494, 10856, 52097, 2761, 16730, 58051, _
6323, 44130, 47274, 5135, 8289, 49910, _
11744, 23425, 19231, 14865, 26335, 4898, _
```

```
18568, 48240, 31103, 15439, 9496, 30470, _
36555, 4507, 30035, 12150, 57597, 1808, _
62910, 55888, 6032, 16544, 13305, 8676, _
64805, 9923, 28490, 47260, 26129, 39058, _
5810, 26490, 10027, 12867, 51925, 18185, _
46674, 1888, 7011, 32425, 28804, 55246, _
61080, 26100, 29102, 59667, 40137, 15178, _
45952, 46289, 30196, 46334, 16536], s2temp
LookUp2 s3,[ 54257, _
63028, 35322, 62349, 5960, 52797, 64491, _
59736, 9049, 7861, 32201, 33279, 26400, _
21604, 5439, 58564, 52438, 9728, 17532, _
14972, 47372, 33517, 55084, 54235, 12009, _
38664, 52880, 63528, 55937, 22002, 53179,_
53922, 23813, 33169, 58981, 11645, 10145, _
62499, 41270, 40803, 13600, 50071, 64515, _
18439, 1060, 21001, 51337, 18483, 56740, _
12082, 28142, 17643, 49953, 62033, 60150,_
31429, 10687, 921, 49958, 29788, 61930, _
17739, 36009, 2128, 12391, 19237, 61499,_
56011, 3749, 23318, 37735, 25152, 40755, _
22582, 2860, 28496, 56621, 53602, 7377, _
25677, 32158, 25174, 54814, 11090, 13350, _
33203, 38881, 57770, 26896, 15397, 47570,_
56335, 33986, 3367, 3895, 40014, 7256, _
17951, 65309, 13768, 50307, 50096, 10228,_
1547, 3640, 27053, 57377, 59855, 65197, _
7480, 40231, 34935, 40334, 27269, 4186, _
30914, 50233, 14073, 60643, 10668, 4783, _
58995, 13814, 44691, 3197, 40084, 29671, _
29657, 60451, 21747, 39465, 8711, 43341, _
45758, 1484, 63338, 1157, 43924, 2630, _
25719, 16207, 47136, 17014, 1072, 33910, _
63263, 16319, 40845, 26162, 63705, 61276, _
8808, 42599, 27092, 7743, 10006, 18942, _
51264, 17466, 47405, 31859, 10401, 23325,_
57597, 49001, 43386, 27949, 39382, 44986, _
44039, 43132, 1962, 45267, 48732, 26816, _
64229, 6937, 62067, 4586, 17999, 41983, _
52037, 40917, 32149, 55290, 43885, 18246,_
33696, 10867, 19000, 5959, 12960, 11727, _
5972, 42552, 43173, 16643, 40028, 48054, _
45359, 20968, 14603, 11807, 23111, 43960, _
54169, 33099, 39312, 7470, 37321, 26254, _
19632, 28139, 24534, 43513, 12510, 11296, _
36793, 21477, 30810, 15635, 24239, 3641, _
39968, 6325, 34754, 21957, 26067, 32834, _
43591, 28020, 2819, 2926, 9836, 16569, _
4299, 16844, 31156, 14216, 56805, 53501, _
19780, 57324, 59715, 49099, 53986, 2992,_
6152, 29118, 677, 13119, 14551, 9240, _
8653, 3126, 26041], s3temp

LookUp P,[28940, 22695, 31790, 31813, 16164, _
          16751, 12389, 10667, 8093, 54269, _
       22920, 13590, 22666, 24922, 59690, _
        4669, 29735, 23966, 63443, 17783, _
       29196, 36704, 61573, 58570, 60521, _
        28745, 51780, 6571, 52232, 49682, _
        29487, 28799, 8248, 56048, 60484, 35419 ],Ptemp

y VAR WORD[2]

;Programa Principal
;-------------------------------------------------------------
L VAR WORD[2]
R VAR WORD[2]
L[0] = 65535
L[1] = 32768
R[0] = 65535
R[1] = 32768
GoSub Cifrar
GoSub Decifrar
End
Cifrar:

 i VAR WORD

 ;variáveis de 32bits
 XL VAR WORD[2]
 XR VAR WORD[2]
 dadoXL VAR WORD[2]
 dadoXR VAR WORD[2]


 XL[0] = L[0]
 XL[1] = L[1]
 XR[0] = R[0]
 XR[1] = R[1]

 For i = 0 TO i < 32 STEP 2
   P = i
   XL[0] = XL[0] ^ Ptemp
   P = i + 1
   XL[1] = XL[0] ^ Ptemp
   GoSub functionF
   XR[0] = y ^ XR[0]
   GoSub functionF
   XR[1] = y ^ XR[1]
   Swap XL[0], XR[0]
   Swap XL[1], XR[1]
 Next i
 Swap XL[0], XR[0]
 Swap XL[1], XR[1]
 P = i
 XR[0] = XR[0] ^ Ptemp
 P = i + 1
 XR[1] = XR[1] ^ Ptemp
 P = i + 2
 XL[0] = XL[0] ^ Ptemp
 P = i + 3
 XL[1] = XL[1] ^ Ptemp
 L[0] = XL[0]
 L[1] = XL[1]
 R[0] = XR[0]
 R[1] = XR[1]
Return


Decifrar:
 XL[0] = L[0]
 XL[1] = L[1]
 XR[0] = R[0]
 XR[1] = R[1]

 For i = 35 TO i > 3 STEP-2
   P = i
   XL[0] = XL[0] ^ Ptemp
   P = i - 1
   XL[1] = XL[0] ^ Ptemp
   GoSub functionF
   XR[0] = y ^ XR[0]
   GoSub functionF
   XR[1] = y ^ XR[1]
   Swap XL[0], XR[0]
   Swap XL[1], XR[1]
 Next i

 Swap XL[0], XR[0]
 Swap XL[1], XR[1]
```

```
 P = 3
 XR[0] = XR[0] ^ Ptemp
 P = 2
 XR[1] = XR[1] ^ Ptemp
 P = 1
 XL[0] = XL[0] ^ Ptemp
 P = 0
 XL[1] = XL[1] ^ Ptemp
 L[0] = XL[0]
 L[1] = XL[1]
 R[0] = XR[0]
 R[1] = XR[1]
Return


functionF:
 d VAR BYTE
 c VAR BYTE
 b VAR BYTE
 a VAR BYTE
 temp VAR BYTE
 tempA VAR WORD
 tempB VAR WORD
 tempC VAR WORD
 tempD VAR WORD

 d = XL[0] & %11111111
 temp = (XL[0] >> 8) & %11111111
 c = temp
 b = XL[1] & %11111111
 temp = (Xl[1] >> 8) & %11111111
 a = temp
 s0 = a
 tempA = s0temp
 s1 = b
 tempB = s1temp
 s2 = c
 tempC = s2temp
 s3 = d
 tempD = s3temp

 y = tempA + tempB
 y = y ^ tempC
 y = y + tempD
```

Código PIC Chave 128 bits

```
s0 VAR WORD
s0temp VAR WORD

s1 VAR WORD
s1temp VAR WORD

s2 VAR WORD
s2temp VAR WORD

s3 VAR WORD
s3temp VAR WORD

P VAR WORD
Ptemp VAR WORD

;Sbox 16 bits
;-----------------------------------------
LookUp2 s0,[59201, _
38106, 35773, 38759, 24787, 34149, 55475, _
46380, 33961, 6969, 43306, 34931, 60437, _
54978, 26295, 27843, 16010, 64033, 9906, _
50563, 40402, 24365, 45402, 62939, 10122, _
5674, 56669, 10692, 40974, 49718, 45169, _
951, 62250, 24021, 23798, 781, 5505, _
38591, 38860, 25978, 64307, 32239, 40005, _
50823, 52722, 22077, 14779, 36216, 39613, _
15411, 18654, 12841, 23669, 47608, 971, _
32336, 59251, 40450, 30925, 34418, 14328, _
25027, 29704, 9594, 34831, 6936, 46857, _
23086, 40093, 38871, 30502, 51177, 11370, _
38248, 5995, 17609, 44572, 48557, 36122, _
40025, 56417, 20686, 38945, 60652, 50048, _
8026, 9278, 25850, 56395, 28977, 28196, _
22506, 27928, 64475, 37984, 8259, 8440, _
18198, 25544, 21374, 27394, 10031, 11881, _
11687, 52755, 10777, 61259, 57953, 46366, _
28311, 8834, 30676, 32033, 22531, 30384, _
61493, 61658, 7867, 1000, 61733, 18495, _
33072, 43388, 25647, 6320, 65248, 23563, _
34635, 35486, 57429, 34429, 54932, 30188, _
32166, 13077, 42939, 20430, 36452, 51027, _
52884, 22133, 30312, 36, 21599, 13615, _
21382, 46948, 26700, 37847, 43141, 64458, _
42928, 64924, 50747, 4699, 25298, 47206, _
41743, 55898, 60723, 57174, 39856, 25270, _
38554, 929, 23443, 47417, 22691, 22795, _
2748, 11405, 4973, 1474, 44849, 53289, _
32992, 28365, 45381, 5109, 52743, 51996, _
37236, 43258, 19550, 57672, 40457, 8181, _
46904, 54207, 8873, 11934, 52668, 25756, _
57207, 65428, 20978, 17737, 22874, 617, _
47362, 18231, 14400, 17210, 42509, 45514, _
36069, 61308, 22086, 38321, 23300, 26341, _
14120, 55821, 31791, 26522, 14433, 11446, _
63418, 45746, 38500, 51321, 4501, 48462, _
14668, 22592, 56820, 486, 34013, 42415, _
43977, 9892, 61140, 20188, 46669, 24816, _
25998, 63128, 54488, 23267, 6855, 30364, _
47478, 63767, 33554, 20172, 58291, 44961, _
125, 3171, 14118, 47978, 4169, 52599, _
53850, 9235, 44698 ], s0temp

LookUp2 s1,[ 7032,30721, 18903, _
62785, 613, 32538, 12764, 4596, 48285, _
46309, 34033, 23849, 20811, 41154, 16602, _
64959, 20504, 34766, 25150, 17859, 10718, _
10345, 58080, 7584, 22068, 17313, 15387, _
22410, 34398, 32708, 13070, 10396, 50787, _
29440, 43872, 18005, 40566, 48927, 6098, _
11851, 37280, 47319, 51813, 37174, 17726, _
12618, 46996, 22720, 34155, 62060, 61152, _
61387, 54497, 9340, 6342, 10585, 62811, _
12412, 2374, 28899, 36996, 4945, 4305, _
3663, 1098, 46702, 36899, 53815, 30502, _
11979, 15810, 54448, 30250, 62706, 52367, _
8438, 36037, 55716, 61819, 44150, 6677, _
49334, 24195, 3716, 34990, 24590, 51724, _
43798, 45670, 16361, 64970, 47326, 41550, _
38766, 12678, 58208, 43388, 31996, 5248, _
54195, 21178, 47235, 22315, 34072, 29931, _
50930, 47667, 2482, 48325, 27540, 57166, _
64828, 63483, 14205, 36586, 58236, 17769, _
42113, 64238, 12093, 46728, 45045, 50028, _
10106, 6079, 42896, 32494, 36224, 9146, _
60237, 55671, 4501, 20747, 16961, 52876, _
2634, 29638,25036, 48610, 5059, 41911, _
31480, 2363, 59550, 60499, 49351, 61418, _
7905, 19175, 18274, 55265, 29560, 51813, _
29354, 63185, 36355, 3974, 32815, 56017, _
13577, 19070, 60559, 58935, 58148, 56196, _
10983, 22031, 28686, 21426, 4009, 18915, _
24689, 30109, 25987, 38422, 1404, 46387, _
36943, 22176, 53334, 42584, 2386, 65491, _
20218, 55083, 34358, 62659, 16236, 16323, _
12442, 11788, 10455, 22445, 9240, 25297, _
16287, 63231, 58063, 36502, 26092, 34841, _
29090, 808, 36371, 19367, 16940, 19773, _
44878, 37076, 49374, 60036, 33159, 60603, _
14104, 40377, 65423, 32636, 49239, 21000, _
24, 19706, 49122, 62719, 34390, 51672, _
60691, 15462, 28879, 30913, 64288, 39680, _
30803, 34212, 50235, 56828, 60681, 23138, _
30285, 3644, 58753, 9979, 19755, 54995, _
11872, 11905, 62507, 7994, 48369, 9763, _
38017, 14599, 3805, 44540, 20277, 31138, 47311], s1temp

LookUp2 s2,[ 48230, _
58747, 44186, 33487, 25875, _
30324, 28474, 38524, 61771, 57627, 4078, _
38402, 34358, 54832, 15256, 43019, 48953, _
13113, 39317, 45035, 13402, 41915, 50829, _
55369, 60990, 28093, 43561, 17536, 12279, _
25079, 21881, 29924, 58204, 44007, 5905, _
37472, 17606, 37289, 6128, 37667, 1684, _
8821, 63495, 62372, 22043, 9890, 19099, _
34219, 22327, 13627, 12926, 15778, 7186, _
3015, 10632, 30645, 29688, 8170, 26647, _
34691, 27620, 32880, 48782, 33920, 37483, _
57312, 17491, 40647, 31148, 36763, 3716, _
62246, 64556, 18833, 8357, 48719, 28447, _
57310, 52288, 31272, 47531, 50517, 57370, _
18941, 53297, 10517, 41837, 41361, 62572, _
20513, 41127, 10704, 58043, 7634, 59820, _
54862, 27574, 7040, 16357, 13344, 18870, _
49566, 33487, 40520, 28254, 11315, 44901, _
55740, 58847, 60500, 64546, 5692, 38678, _
39624, 38617, 44478, 10304, 52288, 56161, _
51970, 47967, 57558, 36076, 61598, 516, _
37971, 48497, 56719, 61688, 11047, 65196, _
64676, 64974, 22974, 1837, 30402, 10771, _
65488, 28432, 44943, 26480, 42959, 40857, _
43918, 51995, 59939, 16668, 10590, 39886, _
4910, 19244, 12150, 24933, 8, 16265, _
65431, 46420, 13257, 15282, 36490, 24218, _
35404, 48458, 61164, 37384, 38851, 26068, _
17733, 45660, 32132, 37488, 48149, 38768, _
31103, 61118, 38905, 3801, 15194, 3465, _
```

```
11572, 64399, 45165, 12186, 21380, 51463, _
24755, 33506, 2683, 36568, 44605, 37222, _
21398, 61094, 54738, 60795, 22548, 40913, _
11968, 11806, 26448, 45357, 27835, 33681, _
50516, 56865, 26563, 48823, 22945, 25192, _
57201, 40721, 59984, 29512, 39278, 38774, _
51352, 34202, 15931, 31755, 48283, 12018, _
16241, 63422, 531, 50869, 29309, 45419, _
64442, 55140, 28027, 27056, 12916, 4029, _
30326, 49138, 15999, 61588, 16830, 55522, _
60733, 1850, 16739, 14753, 2237, 3033, _
11481, 6240, 5602, 50981, 34052, 62387, _
53811, 46118, 55187, 10659, 49093], s2temp


LookUp2 s3,[ 20353, _
5360, 42751, 32684, 24003, 13808, 25675, _
46536, 11397, 31004,  21865, 62554, 10835, _
27606, 5766, 2981, 26326, 5392, 201, _
38621, 59651, 51372, 32491, 25795, 40090, _
24799, 5233, 27227, 43142, 11324, 57885, _
15672, 43193, 11895, 59624, 65363, 5377, _
56964, 51294, 35865, 51474, 1321, 22623, _
36263, 41335, 20131,  51929, 26949, 58519, _
27281, 51262, 57534, 11389, 8900, 56983, _
13305, 41799, 37768, 41518, 47665, 26644, _
63325, 58562, 15953, 2705, 40833, 2889, _
40473, 14752, 14220, 40771, 33370, 24003, _
24336, 58701, 51630, 25706, 56072, 57448, _
184, 36377, 2959, 7513, 18298, 25189, _
39851, 9699, 28759, 42787, 14011, 14543, _
52266, 42939, 61901, 5629, 31974, 63184, _
14849, 7613, 49126, 6105, 49126, 50220, _
11468, 1794, 32041, 25260, 4794, 32837, _
16130, 39078, 52561, 2232, 27926, 64616, _
33867, 18805, 46755, 50284, 64702, 7455, _
15173, 23048, 23624, 37123, 27021, 12016, _
37846, 32763, 60981, 16450, 23402, 56429, _
26960, 13675, 4322, 9679, 9196, 58493, _
65362, 4793, 30013, 3219, 22692, 35593, _
49594, 12550, 25060, 14714, 48756, 56632, _
58762, 40812, 63747, 23981, 23799, 3569, _
23162, 47570, 16101, 60368, 7129, 12861, _
8945, 21581, 10665, 42175, 55646, 26776, _
26318, 42692, 56607, 15435, 61722, 30152, _
42648, 54080, 24250, 52208, 28940, 12584, _
6488, 37840, 18136, 15018, 33264, 9015, _
54229, 10836,  14247, 22436, 11411, 49405, _
32707, 21204, 14062, 30857, 37238, 38868, _
57602, 50933, 64094, 11252, 52951, 8855, _
18880, 53098, 26179, 12184, 37080, 63764, _
9697, 43918, 56705, 50350, 22203, 42400, _
28347, 21706, 60005, 9021, 15465, 60002, _
63159, 12484, 2149, 3192, 58862, 20036, _
53555, 22546, 61459, 45365, 41752, 41441, _
59584, 60201, 46825, 58911, 52477, 41787, _
15173, 13717, 4830, 42347, 26814, 36961, _
49672, 39594, 47777, 49453, 49326, 34243, _
23405, 60540, 4464], s3temp


LookUp P,[ 12064, 5808, _
  41381, 15187, 49946, 62460, 34301, 3553, _
  40011, 60558, 5678, 7809, 35329, 5442, _
  59746, 1072, 15174, 27465, 54255, 65042, _
  60091, 30723, 20226, 53833, 32538, 38503, _
  13595, 3852, 37574, 64219, 4701, 48056, _
  59946, 7101, 4882,  22437],Ptemp


y VAR WORD[2]

;Programa Principal
;-------------------------------------------------------------
;chave de tamanho 128 bits

L VAR WORD[2]
R VAR WORD[2]
L[0] = 65535
L[1] = 32768
R[0] = 65535
R[1] = 32768

GoSub Cifrar

GoSub Decifrar

End

Cifrar:
  i VAR WORD

  ;variáveis de 32bits
  XL VAR WORD[2]
  XR VAR WORD[2]
  dadoXL VAR WORD[2]
  dadoXR VAR WORD[2]

  XL[0] = L[0]
  XL[1] = L[1]
  XR[0] = R[0]
  XR[1] = R[1]

  For i = 0 TO i < 32 STEP 2
    P = i
    XL[0] = XL[0] ^ Ptemp
    P = i + 1
    XL[1] = XL[0] ^ Ptemp
    GoSub functionF
    XR[0] = y ^ XR[0]
    GoSub functionF
    XR[1] = y ^ XR[1]
    Swap XL[0], XR[0]
    Swap XL[1], XR[1]
  Next i

  Swap XL[0], XR[0]
  Swap XL[1], XR[1]
  P = i
  XR[0] = XR[0] ^ Ptemp
  P = i + 1
  XR[1] = XR[1] ^ Ptemp
  P = i + 2
  XL[0] = XL[0] ^ Ptemp
  P = i + 3
  XL[1] = XL[1] ^ Ptemp
  L[0] = XL[0]
  L[1] = XL[1]
  R[0] = XR[0]
  R[1] = XR[1]

 Return

Decifrar:
  XL[0] = L[0]
  XL[1] = L[1]
  XR[0] = R[0]
  XR[1] = R[1]
```

```
  For i = 35 TO i > 3 STEP-2
    P = i
    XL[0] = XL[0] ^ Ptemp
    P = i - 1
    XL[1] = XL[0] ^ Ptemp
    GoSub functionF
    XR[0] = y ^ XR[0]
    GoSub functionF
    XR[1] = y ^ XR[1]
    Swap XL[0], XR[0]
    Swap XL[1], XR[1]
  Next i

  Swap XL[0], XR[0]
  Swap XL[1], XR[1]
  P = 3
  XR[0] = XR[0] ^ Ptemp
  P = 2
  XR[1] = XR[1] ^ Ptemp
  P = 1
  XL[0] = XL[0] ^ Ptemp
  P = 0
  XL[1] = XL[1] ^ Ptemp
  L[0] = XL[0]
  L[1] = XL[1]
  R[0] = XR[0]
  R[1] = XR[1]

  Return


functionF:
  d VAR BYTE
  c VAR BYTE
  b VAR BYTE
  a VAR BYTE
  temp VAR BYTE
  tempA VAR WORD
  tempB VAR WORD
  tempC VAR WORD
  tempD VAR WORD

  d = XL[0] & %11111111
  temp = (XL[0] >> 8) & %11111111
  c = temp
  b = XL[1] & %11111111
  temp = (Xl[1] >> 8) & %11111111
  a = temp

  s0 = a
  tempA = s0temp
  s1 = b
  tempB = s1temp
  s2 = c
  tempC = s2temp
  s3 = d
  tempD = s3temp

  y = tempA + tempB
  y = y ^ tempC
  y = y + tempD

  Return
```