UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

MODELO SIMPLIFICADO DO CIFRADOR IDEA

HENRIQUE TOMASI PIRES

BLUMENAU
2006

2006/1-17

HENRIQUE TOMASI PIRES

MODELO SIMPLIFICADO DO CIFRADOR IDEA

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Paulo Fernando da Silva, MsC. - Orientador

BLUMENAU
2006

2006/1-17

MODELO SIMPLIFICADO DO CIFRADOR IDEA

Por

HENRIQUE TOMASI PIRES

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo Fernando da Silva, MsC. – Orientador, FURB
Membro:	Prof. Francisco Adell Péricas, MsC – FURB
Membro:	Prof. Antonio Carlos Tavares – FURB

Blumenau, 03 de julho de 2006

Dedico este trabalho a meus pais, João Roberto Pires e Zenaide Aparecida Tomasi Pires e a minha irmã, Roberta Tomasi Pires. Pessoas que amo e respeito tanto. Esbanjo respeito e amor perante a esta família tão unida cujo tem riquezas impagáveis conquistadas com muito esforço e humildade, riquezas como: companheirismo, dedicação, vontade de vencer e ajudar o próximo. Pessoas das quais sempre estiveram presentes em todos os momentos de minha vida, nem sempre presentes fisicamente, porém sempre presentes espiritualmente dando-me força para vencer batalhas maiores de que minha própria pessoa, que sozinho não teria condições de superá-las.

AGRADECIMENTOS

Aos meus pais pela confiança, suporte e principalmente por acreditar em minha capacidade de realização.

À minha irmã, que me mostrou que a distância é apenas uma simples palavra do vocabulário da língua portuguesa, devido ao seu apoio e preocupação com seu irmão mais novo.

À minha família, que sempre incentivou-me a buscar conhecimento.

Aos meus amigos, pela compreensão da minha ausência nos últimos eventos festivos.

Ao meu orientador, Paulo Fernando da Silva, por acreditar em minha pessoa e em minha vontade de tornar este trabalho uma realidade.

À Emanoel Koslosky, pela ajuda prestada na implementação do protótipo em linguagem C.

À Mediacrypt, que obtêm a patente do cifrador IDEA, e que disponibilizou seu código para a confecção deste trabalho.

Há duas maneiras de viver a vida: Uma, é como se nada fosse milagre. A outra, como se tudo fosse milagre.

Albert Einstein

RESUMO

Este trabalho consiste na proposta de um modelo simplificado para o cifrador de bloco simétrico IDEA, a ser denominado SIDEA. Um modelo simplificado pode auxiliar no entendimento de um cifrador, pois ele mantém as operações primitivas, enquanto diminui os parâmetros, facilitando seu uso. O meio adotado foi o estudo dos princípios matemáticos/algébricos para que se consiga realizar a maior redução possível do número de parâmetros para a compreensão da estrutura básica. Durante o desenvolvimento do protótipo, faz-se o estudo do cifrador DES e seu modelo simplificado, o S-DES, verificando-se a contribuição que o S-DES traz no entendimento do DES. No decorrer do trabalho é apresentada a metodologia empregada para as reduções, as técnicas utilizadas na implementação do protótipo, tal como os testes e resultados do mesmo. Este trabalho visa facilitar a explicação tanto dos mecanismos que provêem forte resistência, quanto do algoritmo completo do IDEA. Outro ponto importante deste trabalho é tornar possível uma implementação manual do modelo simplificado, o qual tem correlação com o IDEA e que também poderá servir como referência para a proposição de outros modelos simplificados de cifradores simétricos.

Palavras-chave: Cifrador de bloco. Cifrador simétrico. Modelo simplificado. IDEA.

ABSTRACT

The development of a simplified model for the IDEA symmetrical block cipher, denominated SIDEA is proposed. A simplified model can help one to understand a cipher, for it keeps the primitive operations, while decreases the parameters, making it easier to use it. The adopted way was the study of algebraic and mathematical principles to obtain the maximum possible reduction of the number of parameters for the understanding of the basic structure. During the development of the prototype, DES cipher and its simplified model, S-DES, are analyzed, to verify if S-DES contributes to the understanding of DES. In elapsing of the work the methodology used for the reductions, the techniques used in the implementation of the archetype is presented, as the tests and results. This work aims at an easier way to explain not only the mechanisms that provide strong resistance, but also to explain the real one. Another important point of this work is to make possible a manual implementation from the simplified model that has correlation with IDEA and can serve like reference to another proposal simplified model of symmetrical block ciphers.

Key-words: Block cipher. Symmetrical cipher. Simplified model. IDEA.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

3DES – *Triple Data Encryption Standard*

AES – *Advanced Encryption Standard*

ANSI – *American National Standards Institute*

CBC – *Cipher Block Chaining*

CFB – *Ciphertext Feedback*

DEA – *International Data Encryption Algorithm*

DES – *Data Encryption Standard*

DSA – *Digital Signature Algorithm*

DOS – Sistema operacional em modo caracter

ECB – *Electronic Code Book*

ECC – *Elliptic Curve Crytpgraphy*

NBS – *National Bureau of Standards*

NIST – *National Institute of Standards and Technology*

IBM – *International Business Machines*

IDEA – *International Data Encryption Algorithm*

IP – Permutação Inicial

IPES – *Improved Proposed Encryption Algorithm*

IV – Vetor de Inicialização

LSHIFT – Deslocamento circular à esquerda de bits

LSW4 – *Low Significant 4-bit Word*

MA – Multiplicação/Adição

MOD – *Mission Operations Division*

MSW4 – *Most Significant 4-bit Word*

NXT – *Next Generation Encryption*

OFB – *Outpot Feedback*

PES – *Proposed Encryption Algorithm*

PGP – *Pretty Good Privacy*

RC5 – *Ron`s Code 5*

RC6 – *Ron`s Code 6*

RSA – *Rivest, Shamir, Adleman*

SAES – *Simplified Advanced Encryption Standard*

S-DES – *Simplified Data Encryption Standard*

SIDEA – *Simplified International Data Encryption Algorithm*

SRC6 – *Simplified Ron`s Code 6*

XOR – OU Exclusivo

SUMÁRIO

1 INTRODUÇÃO

O conceito de criptografia é tão antigo quanto a escrita e consiste na arte e na ciência de comunicar-se secretamente. Antigamente, os romanos usavam a criptografia para envio de mensagens durante as guerras. Depois da Segunda Guerra Mundial, com a invenção do computador, a área realmente emergiu e também fortaleceu estudos sobre a computação (BERNSTEIN, 1997).

Segundo Mendes (2001), a segurança da informação está se tornando mais importante à medida que mais dados são enviados através das redes. Uma vez que os dados trafegam por uma rede, eles podem ser interceptados em vários pontos. Uma forma de proteger a informação, sendo ela transmitida ou mesmo estando dentro de um computador, é cifrá-la (criptografá-la), isto é, torná-la ilegível a todos, exceto àqueles que realmente podem ter acesso a ela. Para o adequado uso e o aperfeiçoamento dos algoritmos de criptografia, faz-se necessário o estudo e a análise dos mesmos, como também o seu perfeito entendimento, para a procura de possíveis vulnerabilidades.

Conforme mencionado em Miers (2002), o primeiro cifrador a tornar-se padrão no mundo foi uma variante do Lucifer da IBM, que tornou-se então conhecido como *Data Encryption Standard* (DES) em 1977. O DES foi usado por muitos anos pelo governo americano e com isso tornou-se padrão na maioria das aplicações desenvolvidas nesta época. Com o passar dos anos e o aumento exponencial do poder computacional, o cifrador tornou-se frágil. Em 1998 foi decidido que deveria ser escolhido um substituto para o DES. Com a necessidade de algoritmos de criptografia mais fortes, foram surgindo novos cifradores como o *Triple Data Encryption Standard* (3DES) e o *International Data Encryption Algorithm* (IDEA), os quais têm a arquitetura muito semelhante ao DES original, porém com maior eficiência e segurança.

Tendo em vista o cenário apresentado, este trabalho visa à compreensão do cifrador IDEA através do desenvolvimento de um modelo simplificado. Devido ao alto nível de complexidade do algoritmo original, o modelo simplificado foi elaborado através de estudos sobre os processos matemáticos que compõem o cifrador. Após a redução destes processos matemáticos, a simplificação foi denominada *Simplified International Data Encryption Algorithm* (SIDEA). Esta simplificação ajuda de forma significativa a explicação tanto dos mecanismos que provêem forte resistência, quanto do algoritmo completo do IDEA. Através da simplificação, o SIDEA pode ser utilizado de forma didática nos cursos de graduação ou pós-graduação e por pesquisadores que atuam na área de segurança em redes.

A implementação deste trabalho tem uma documentação que aborda as semelhanças entre o cifrador original e o simplificado, tal como a comparação entre os mesmos levando em consideração o tamanho das chaves e blocos, velocidade do algoritmo e a segurança entre eles. A aplicação gerada pode ser utilizada para criptografar dados ou também para auxiliar no ensino do cifrador IDEA.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é a especificação e implementação do protótipo de um modelo simplificado do cifrador IDEA, que tem como finalidade auxiliar o ensino e facilitar a compreensão dos mecanismos que compõem tal cifrador.

1.1.1 Objetivos específicos

Os objetivos específicos do presente trabalho são:

a) propor um protótipo de um modelo simplificado do cifrador IDEA sem

descaracterizar suas propriedades matemáticas;

b) expor os mecanismos de geração de chaves e de cifragem do IDEA de forma didática;
c) identificar os pontos de redução no IDEA;
d) disponibilizar versão didática em linguagem C do SIDEA;
e) servir como referência para futuros desenvolvimentos de modelos simplificados de cifradores.

1.2 ESTRUTURA DO TRABALHO

O trabalho está dividido em cinco capítulos. No capítulo seguinte é descrita a fundamentação teórica utilizada para embasar este trabalho. É apresentada a importância da presença da criptografia na transmissão de dados. São descritos os conceitos de criptografia assimétrica, simétrica, cifrador de bloco e o cifrador de Feistel. O capítulo 2 também aborda os cifradores DES, S-DES. No capítulo 3 é descrito de forma detalhada o cifrador IDEA, os processos matemáticos do cifrador, geração de chaves, modos de operação, desempenho, segurança e uma breve apresentação da próxima geração do IDEA. O capítulo é finalizado com os trabalhos correlatos.

O capítulo 4 traz a especificação e implementação do protótipo. É feita uma comparação entre o protótipo simplificado e o cifrador original levando em consideração as propriedades matemáticas, o tamanho das chaves e blocos, velocidade do algoritmo e segurança entre os mesmos. O capítulo é finalizado com os resultados alcançados com o protótipo.

O último capítulo contém a conclusão do trabalho, juntamente com sugestões para trabalhos futuros.

2 SEGURANÇA DA INFORMAÇÃO NA INFORMÁTICA

Neste capítulo são apresentados alguns aspectos teóricos relacionados com o trabalho, tais como: segurança em computação, criptografia, cifrador de bloco e cifrador de Feistel. No final deste capítulo são apresentados os cifradores DES e S-DES.

2.1 SEGURANÇA EM COMPUTAÇÃO

Com o crescente uso das redes de computadores por organizações para conduzir seus negócios e a massificação do uso da Internet, surgiu a necessidade de se utilizar melhores mecanismos para prover a segurança das transações de informações confidenciais. A questão segurança é bastante enfatizada, principalmente, quando se imagina a possibilidade de se ter informações expostas a atacantes ou intrusos da Internet, que surgem com meios cada vez mais sofisticados para violar a privacidade e a segurança das comunicações. Devido a estas preocupações, a proteção da informação tem se tornado um dos interesses primários dos administradores de sistemas (CUSTÓDIO, 2003).

Segundo Custódio (2003), a segurança em computação consiste na certeza de que as informações de uso restrito não devem ser acessadas, copiadas ou codificadas por pessoas não autorizadas. Para a garantia disto, é necessário o uso da criptografia.

2.2 CRIPTOGRAFIA

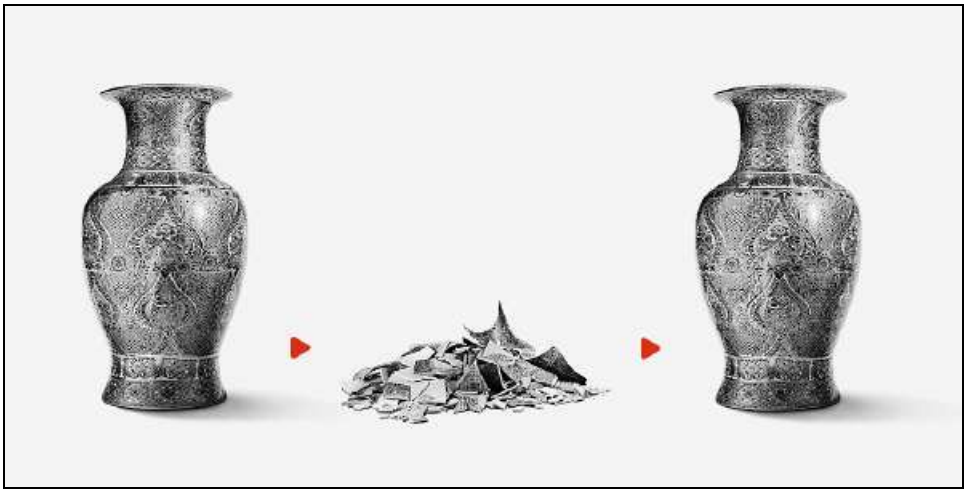
Para Miers (2002, p. 18), criptografia consiste na ciência e na arte de comunicar-se secretamente, ou seja, através de procedimentos matemáticos transformar algo legível, desde um simples texto até um arquivo por completo, em algo indecifrável. O avanço computacional

nos últimos anos e a necessidade da segurança na informação teve como conseqüência o grande destaque da segurança em redes, tal como a importância da criptografia em dados sigilosos.

Cifrar significa que a mensagem legível (texto claro) será transformada em uma mensagem ilegível (texto cifrado). A função de decifrar é o processo inverso da cifragem: a partir de um texto cifrado, obtêm-se o texto claro.

Um algoritmo criptográfico é uma função matemática usada para codificar e decodificar informações. Em geral, os algoritmos de criptografia modernos – para tornar a codificação realmente segura – utilizam uma chave, *K*. A chave criptográfica foi inventada pelo italiano Leon Battista Alberti em 1466 (COUNTERPANE, 1998). As chaves criptográficas podem variar de tamanho entre um cifrador e outro. O conceito de chaves criptográficas será trabalhado mais adiante na seção de fundamentação teórica.

Existem vários tipos de cifradores que utilizam vários procedimentos distintos para criptografar determinado texto ou dado.



Fonte: Mediacrypt(2005).

Figura 1 – Princípio básico de criptografia

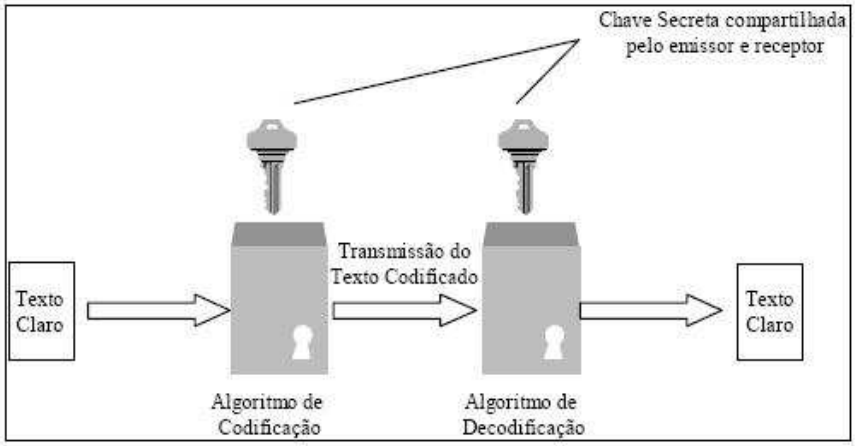
A figura 1 ilustra o princípio básico destes cifradores, que têm como objetivo tornar um objeto que está em sua forma bruta, da qual qualquer pessoa possa identificá-lo facilmente, em algo que não possa ser mais identificado, e que o único meio de transformar

este objeto em sua forma bruta novamente é utilizando este cifrador de forma inversa sobre o objeto.

A criptografia moderna está dividida em duas classes: criptografia convencional ou simétrica, e criptografia assimétrica ou de chave pública.

2.2.1 Criptografia simétrica

A criptografia simétrica pode ser chamada também de criptografia convencional. Esta técnica utiliza uma chave secreta nos procedimentos de cifrar e decifrar, portanto a chave deve ser distribuída de forma segura entre o responsável pela cifragem e o responsável que irá receber a mensagem cifrada. Os algoritmos dos cifradores simétricos são públicos, conseqüentemente são projetados para serem seguros, ou seja, juntando o algoritmo e o texto cifrado por este algoritmo, não se consegue chegar ao texto claro, por isto não há riscos em torná-los públicos. A segurança da criptografia convencional está na chave.



Fonte: Ramos(2002).

Figura 2 – Funcionamento da criptografia simétrica

A figura 2 exibe o processo de criptografia convencional. A mensagem original ou texto claro é convertido em um texto cifrado através de um processo de codificação. Este processo de codificação consiste de um algoritmo e uma chave compartilhada pelo emissor e receptor (RAMOS, 2002). A chave deve ser compartilhada através de um canal seguro, visto

que a chave é relativamente menor do que o texto a ser cifrado e que o segredo da criptografia convencional está unicamente na chave.

Algumas das vantagens de utilizar a criptografia convencional são: a velocidade nos processos de cifrar e decifrar, a segurança do algoritmo e a constante evolução destes algoritmos, fruto da popularização perante os aplicativos que necessitam de transmissão segura de dados. Em contrapartida, a criptografia simétrica requer compartilhamento seguro da chave secreta, utiliza um grande número de sub-chaves em seus processos de cifragem, tem uma administração complexa e não permite a implementação da não-retratação, que é uma das propriedades de segurança na transmissão de dados, da qual a pessoa que originou a mensagem não pode negar tê-lo feito (SCHNEIER, 1996).

A criptografia convencional moderna está dividida em:

a) codificadores de fluxo: processam continuamente a entrada de elementos, produzindo ao longo da saída um elemento de cada vez;
b) codificadores de bloco: processa um bloco de elementos a cada momento, produzindo um bloco de saída para cada bloco de entrada.

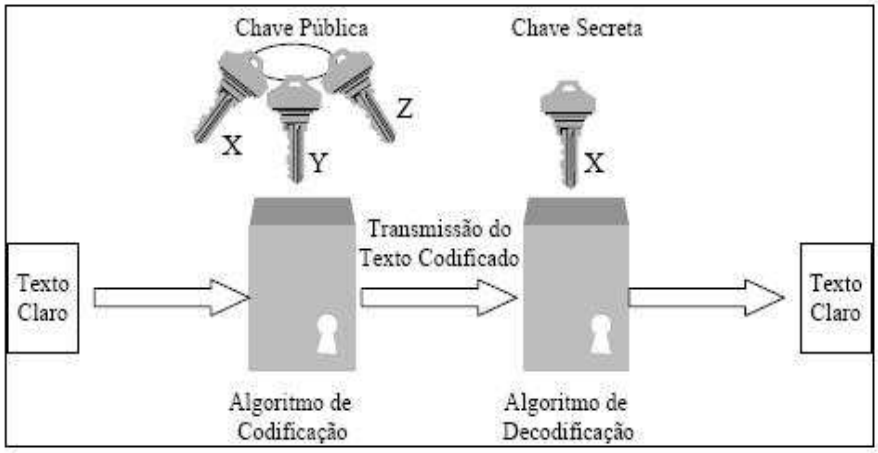
A criptografia simétrica é muito utilizada para criptografar senhas em sistemas operacionais, operações bancárias, transmissão de dados, entre outros. Os cifradores mais conhecidos de criptografia simétrica são: DES, 3DES, RC5, RC6, IDEA, AES e Blowfish.

2.2.2 Criptografia assimétrica

A criptografia assimétrica também é conhecida como criptografia de chave pública. No sistema de criptografia por chave pública, cada usuário obtém um par de chaves: uma é chamada de chave pública, a outra é chamada de chave privada. Através da chave pública uma mensagem pode ser codificada, mas a decodificação só poderá ser feita pelo detentor da

chave privada.

A figura 3 ilustra o processo de criptografia de chave pública utilizado para codificação. A mensagem original ou texto claro é cifrado através de um processo de codificação que utiliza a chave pública do receptor da mensagem. Do lado do receptor, a mensagem é decodificada utilizando um algoritmo de decodificação e a chave privada do receptor (RAMOS, 2002).

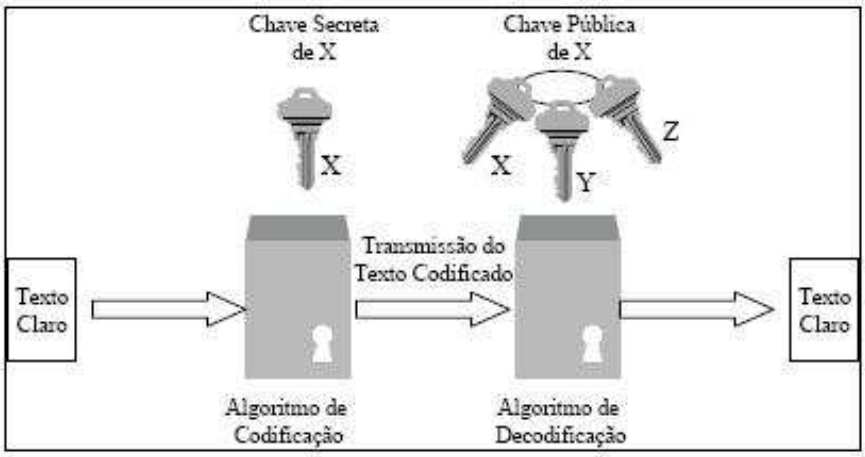


Fonte: Ramos(2002).

Figura 3 – Funcionamento da criptografia assimétrica

Segundo Ramos (2002), o algoritmo de criptografia de chave pública, além de oferecer confidencialidade, pode ser utilizado, também, para autenticação.

A figura 4 exibe o processo de criptografia de chave pública utilizado para autenticação. Uma mensagem é cifrada através do algoritmo de codificação e da chave secreta do emissor (X). Do lado do receptor (Y), a mensagem é decodificada através do algoritmo de decodificação e da chave pública do emissor. Devido à mensagem ter sido cifrada usando a chave privada de X, somente X poderia ter codificado a mensagem, logo, a mensagem codificada serve como assinatura digital. Além de autenticar o emissor, este tipo de codificação garante também a integridade da mensagem, pois somente com a chave privada de X a mensagem poderia ser alterada.



Fonte: Ramos(2002).

Figura 4 – Autenticação

Entre as vantagens de se usar criptografia de chave pública estão a possibilidade de implementar a confidencialidade, a autenticação ou ambos simultaneamente. Por outro lado a computação matemática usada para criptografar dados é intensiva, exigindo muito tempo de processamento, o que para algumas aplicações é inaceitável. Outra desvantagem é a necessidade de um centro de distribuição de chaves. A criptografia assimétrica é muito usada em certificação digital e seus principais cifradores são: RSA, DAS e ECC.

2.3 CIFRADOR DE BLOCO

Nos cifradores de bloco, o texto é quebrado em blocos. Os blocos e as chaves possuem tamanhos fixos, não podendo ser relativamente pequenos por motivos de segurança e nem muito grandes por problemas com desempenho. O cifrador utiliza um bloco do texto claro para produzir um bloco cifrado de tamanho igual, sendo que o texto claro e o crifrado não devem ter correlação, ou seja, através do procedimento de espalhamento de bits, metade deve permanecer igual e a outra parte diferente (PUTTINI, 2004, p. 28).

Segundo Puttini (2004), o tamanho típico dos blocos é de 64 bits, e as principais operações de um cifrador de blocos são:

a) substituição ou alfabeto permutado: utiliza o mapeamento de um bloco de k

bits em 2^k blocos de k bits, ou seja, efetua uma troca de bits com outros bits previamente definidos;

b) transposição ou permutação no bloco: executa a mudança da posição de cada bit no bloco, conhecido também como espalhamento de bits;

c) *round*: utiliza a combinação das operações de substituição com permutação, de tal forma que um bit de entrada possa afetar todos os bits de saída.

2.3.1 Modos de operação

Os cifradores de bloco possuem quatro modos de operação de cifragem, são eles:

a) *Electronic Code Book* (ECB);

b) *Cipher Block Chaining* (CBC);

c) *Ciphertext Feedback* CFB);

d) *Output Feedback* (OFB).

Cada modo de operação é descrito abaixo, com certa ênfase no cifrador IDEA, que é o objeto de estudo deste trabalho.

2.3.1.1 Electronic Code Book

É o modo mais simples de cifragem e recebe esta denominação em virtude de um bloco de texto claro, dada uma chave, sempre resultar em um único bloco de texto cifrado, independente de sua localização. Teoricamente, é possível criar um livro código de textos claros e seus correspondentes textos cifrados. Entretanto, se os blocos forem de 64 bits existirão 2^{64} entradas para o livro de código, isto para cada chave, o qual torna muito elevada a computação e o espaço de armazenamento.

Cada bloco de texto claro é cifrado independentemente, tornando-se útil para arquivos encriptados em que se deseja acesso aleatório (por exemplo, um banco de dados). Este modo também é ideal para pequenas quantidades de dados, tal como encriptação de chave (RAMOS, 2002).

Para mensagens maiores, o modo ECB não se mostra seguro, pois no caso de um texto bem estruturado é possível explorar as regularidades do texto e compor um livro código. Para minimizar este problema, deve-se mudar a chave freqüentemente ou até adicionar alguns pares de bits a cada bloco, no intuito de reforçar a segurança. Em contrapartida, os blocos de 64 bits ou mais devem conter características únicas (entropia), tornando um ataque através de um livro de códigos ineficaz, por exemplo, imagens com um grande número de repetições. No modo ECB, qualquer bit errado em um bloco cifrado afeta somente a decriptação desse bloco. Caso os blocos de textos cifrados forem reordenados, serão reordenados apenas os blocos de textos claros correspondentes, que não afetará toda a decodificação (MEDIACRYPT, 2005).

2.3.1.2 Cipher Block Chaining

O modo CBC sugere um mecanismo de realimentação para um cifrador de bloco, pois neste modo a entrada para o algoritmo de criptografia é o XOR do bloco de texto claro atual e o bloco cifrado anteriormente e a mesma chave é usada para cada bloco.

Para produzir o primeiro bloco de texto cifrado, faz-se um XOR do bloco de texto claro com um vetor de inicialização. O vetor de inicialização evita que blocos de texto claro de textos idênticos gerem blocos iguais de texto cifrado. Deste modo, é impossível um intruso tentar repetir um bloco, bem como se torna mais difícil a geração de um livro código. O vetor de inicialização deve ser conhecido por ambos: o emissor e o receptor. O modo CBC é apropriado para criptografar textos maiores que 64 bits (RAMOS, 2002).

Apenas um bit errado em um bloco no texto cifrado afeta a decodificação de todos os blocos subseqüentes. A reordenação dos blocos de texto cifrado faz com que a decodificação seja corrompida. Perante o ECB têm-se a vantagem de que o XOR embaralha a padronização do texto claro. Idealmente, o vetor da inicialização deve ser diferente para cada dois textos cifrados com a mesma chave. Embora o vetor da iniciação não necessite ser secreto, em algumas aplicações acha-se aconselhável tornar o vetor secreto, conforme definido em Mediacrypt (2005).

2.3.1.3 Ciphertext Feedback

Através do modo CFB é possível converter um cifrador de bloco em cifrador de fluxo. Um cifrador de fluxo elimina a necessidade de preenchimento de um texto para obter o tamanho de bloco ideal, bem como, pode operar em tempo real. Dessa maneira, se um fluxo de caracteres está sendo transmitido, cada caractere pode ser criptografado e transmitido imediatamente usando um cifrador de fluxo orientado por caractere. No cifrador de fluxo é desejável que o texto cifrado seja do mesmo comprimento que o texto claro. Assim, se caracteres de 8 bits estão sendo transmitidos, cada caractere deve ser criptografado com 8 bits (RAMOS, 2002).

O CFB utiliza como entrada na função de criptografia um registrador de deslocamento. A parte que realimenta o registrador de deslocamento, até que todo texto claro seja cifrado, é o resultado do XOR realizado entre os 8 bits mais significativos da saída da função de criptografia e a unidade do texto claro, ou seja, é a unidade cifrada.

Como no modo CBC, trocando o vetor de inicialização para o mesmo bloco do texto claro resulta em uma saída diferente. Ao reordenar a seqüência dos blocos de texto cifrado, a saída na decodificação é alterada, já que a decriptação de um bloco depende da decodificação

dos blocos seguintes. Uma aplicação típica do modo de operação CFB é a autenticação de usuários em sistemas operacionais e aplicativos diversos (MEDIACRYPT, 2005).

2.3.1.4 Output Feedback

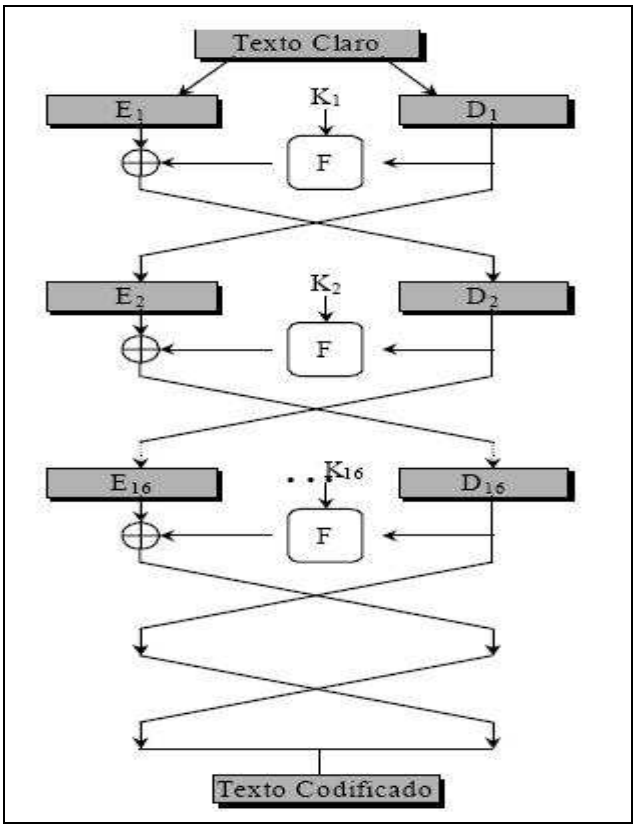
Conforme em Ramos (2002), o modo OFB também possui um vetor de inicialização e é semelhante ao modo CFB. A diferença entre eles consiste da parte que realimenta o registrador de deslocamento da entrada da função de criptografia. No modo OFB, a saída da função de criptografia é a parte que realimenta o registrador de deslocamento, enquanto que no modo CFB o registrador de deslocamento é realimentado pela unidade cifrada.

O modo OFB é às vezes chamado de realimentação interna, porque o mecanismo de realimentação é independente dos fluxos de texto claro e texto cifrado. Uma vantagem desse modo de operação é que erros de bits na transmissão não se propagam. Por exemplo, se ocorre um erro de bit na parte cifrada, somente o valor recuperado da parte do texto claro equivalente é afetada; as unidades de texto claro subseqüentes não são modificadas. Devido a esta vantagem, uma aplicação típica desse modo de operação é a transmissão orientada por fluxo feita em canal ruidoso, por exemplo, comunicação por satélite (RAMOS, 2002).

2.4 CIFRADOR DE FEISTEL

Conforme mencionado em Ramos (2002), Horst Feistel, em 1973, propôs um codificador de blocos que alterna funções de substituição e permutação. A estrutura do codificador de Feistel constitui-se a base dos modernos cifradores simétricos. Este cifrador foi o primeiro a utilizar os termos difusão e confusão. O termo difusão foi utilizado com o objetivo de tornar a relação estatística entre texto claro e texto cifrado a mais complexa

possível, a fim de evitar a descoberta da chave. O termo confusão foi empregado para evitar a descoberta da chave de codificação, portanto a confusão busca tornar o mais complexo possível a relação estatística entre o texto cifrado e a chave de codificação.



Fonte: Ramos(2002).

Figura 5 – Estrutura do cifrador de Feistel

A figura 5 demonstra que a função F, aplicada ao lado direito do dado, constitui-se o coração do cifrador Feistel, uma vez que ele fornece os elementos de confusão. A função F não precisa ser reversível. Independente de como seja constituída essa função, a estrutura do cifrador permite que o mesmo algoritmo da codificação seja utilizado para decodificação. Para o algoritmo de codificação ser utilizado para decodificação, é preciso que as sub-chaves geradas sejam usadas em ordem reversa, ou seja, a primeira volta da decodificação utiliza a última sub-chave de codificação, e assim por diante, até que o dado seja recuperado (RAMOS, 2002).

2.5 CIFRADOR DES

O DES foi desenvolvido pela IBM e adotado, em 1977, pelo *National Bureau of Standards* (NBS), atual *National Institute of Standards and Technology* (NIST), como o Padrão de Processamento de Informação Federal dos Estados Unidos (STALLINGS, 2003). Conforme Miers (2002), o NIST renovou a padronização do DES em 1993 pela última vez, estabelecendo que o DES continuaria como padrão somente até o final de 1998, ocasião em que deveria ser escolhido seu substituto.

Com o passar dos anos e o aumento exponencial do poder computacional, o DES tornou-se frágil. Em 1993 já era possível quebrar uma chave gerada pelo DES em 3 horas e meia, portanto comprovou-se a necessidade de novos cifradores com chaves maiores e mecanismos de cifrar nos algoritmos que dificultassem mais os métodos de criptoanálise atuais e futuros.

O DES utiliza uma chave de 56 bits para codificar blocos de texto claro de 64 bits. O processamento do texto claro é realizado em três etapas. Inicialmente, o bloco de 64 bits passa por uma função de permutação inicial (IP), gerando uma saída permutada. Em seguida passa por um processo de 16 voltas que consiste de funções de permutação e substituição, envolvendo texto claro e chave. Por último, a saída do processo de 16 voltas passa através de uma permutação (IP^{-1}) que é o inverso da função de permutação inicial.

Devido ao enfraquecimento do cifrador DES, foram surgindo cifradores substitutos com chaves maiores e algoritmos com maior velocidade como: 3DES, IDEA e AES.

2.6 CIFRADOR S-DES

Uma dificuldade encontrada no estudo do DES é o seu nível de complexidade e a

dificuldade de seu entendimento. O DES Simplificado, que é chamado de S-DES, é um cifrador desenvolvido para ser utilizado de forma mais didática do que em cifragem segura, tendo propriedades e estrutura similares ao DES, porém com parâmetros muito menores. Uma vez tendo entendido o S-DES, se torna mais fácil expandir os parâmetros para ver como o DES funciona.

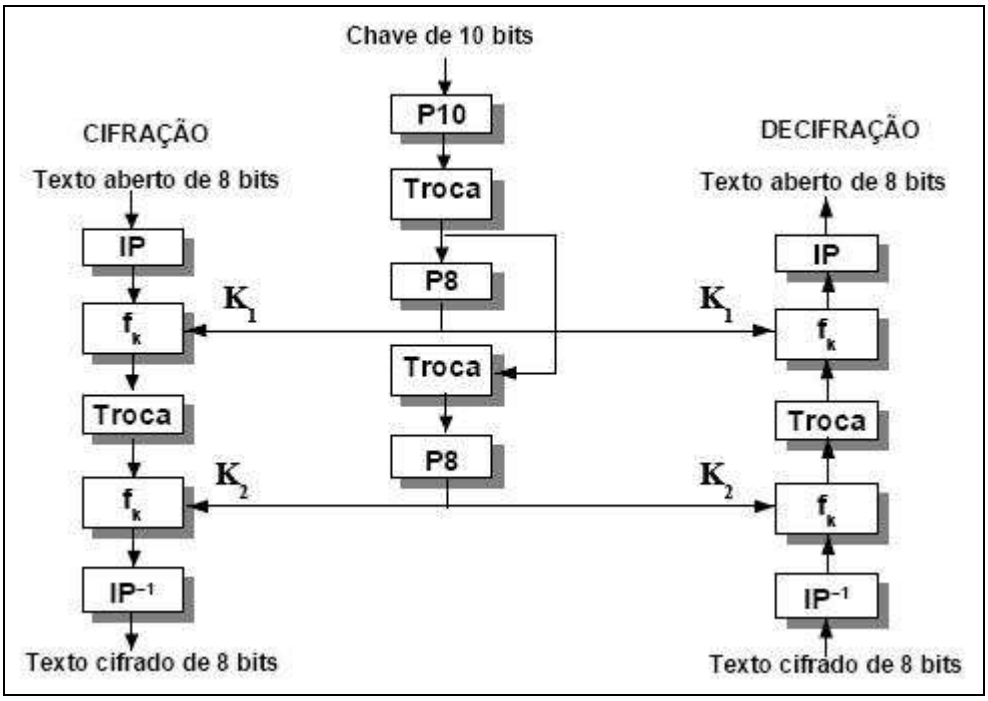
O algoritmo de cifragem S-DES recebe um texto claro de 8 *bits* e uma chave de 10 *bits* como entrada e produz um bloco de texto cifrado de 8 *bits* como saída. O algoritmo é simétrico e portanto a mesma chave é usada na decifragem.

2.6.1 Funcionamento do algoritmo S-DES

O algoritmo de cifragem do S-DES envolve cinco funções:

a) permutação inicial $(IP)_1$;
b) função complexa F que envolve operações de permutação e substituição e é dependente da chave de entrada;
c) operação de permutação simples que troca as duas metades do dado;
d) função F novamente;
e) função final de permutação, que é o inverso da permutação inicial (IP).

O esquema de funcionamento do S-DES pode ser visto na figura 6. Pode-se notar que o cifrador aplica o conceito de permutação em seu algoritmo, isto é, os bits envolvidos neste processo de permutação são trocados por bits previamente definidos. Para a geração de chaves o S-DES utiliza a permutação de 10 bits (P10) e posteriormente outra permutação de 8 bits (P8).



Fonte: Mendes(2002).

Figura 6 – Esquema de funcionamento do S-DES

Conforme apresentado na figura 6, o algoritmo de cifragem do S-DES recebe um texto aberto de 8 *bits* e uma chave de 10 *bits* como entrada e produz um bloco de texto cifrado de 8 *bits* como saída. O algoritmo é simétrico e portanto a mesma chave é usada na decifragem (MENDES, 2002).

3 CIFRADOR IDEA

Segundo Mediacrypt (2005), o algoritmo IDEA começou a ser desenvolvido em 1990, por Xuejia Lai e James Massey, da Suíça, chamado inicialmente de *Proposed Encryption Algorithm* (PES). No ano seguinte, após diversas análises que demonstraram alguns pontos fracos potenciais, seus autores fortaleceram o algoritmo original, denominando a nova versão de *Improved Proposed Encryption Algorithm* (IPES). Em 1992, o nome foi alterado para IDEA.

O IDEA é um cifrador de bloco iterativo, com um bloco de tamanho igual a 64 bits, um tamanho de chave de 128 bits e baseia-se na mistura de operações matemáticas de adição, multiplicação e XOR a partir de grupos algébricos diferentes, o que provê à comunidade de criptografia alguma dimensão para confiar na segurança do mesmo (CHAVES, 2001). Por outro lado, o sistema criptográfico DES trabalha com uma chave de 56 bits e possui menos teoria matemática e é suscetível a alguns métodos criptoanalíticos, tais como a criptoanálise linear que é baseada em ataques por mensagens conhecidas ou criptoanálise diferencial que se baseia em ataques por mensagens escolhidas.

O IDEA é um algoritmo criptográfico considerado novo para a comunidade de criptografia e atende aos principais critérios utilizados pelo NIST para avaliar as cifras de bloco na prática, que são: nível de segurança, tamanho da chave, desempenho do sistema, domínio público e complexidade da função de criptografia. Conforme as características descritas anteriormente, o cifrador IDEA tornou-se popular e é amplamente utilizado atualmente, sendo aplicado no mundo inteiro, desde a comunicação segura a radio, o PGP até o *internet banking* (MEDIACRYPT, 2005).

Apesar do IDEA possuir um algoritmo complexo, o desempenho do cifrador é algo que foi muito trabalhado pelos seus criadores. A tabela 1 apresenta os resultados obtidos em

implementações desenvolvidas em linguagem C-ANSI, executadas em um Pentium II – 400 Mhz no sistema operacional Linux – Conectiva.

Tabela 1 – Tempo de execução do IDEA e do DES

Quantidade de blocos de 64 bits	Algoritmo IDEA	Algoritmo DES
10.000	0.063s	2.938s
50.000	0.356s	14.447s
100.000	0.730s	28.732s

Fonte: Freitas e Oliveira (2006).

3.1 PROCESSOS MATEMÁTICOS DO IDEA

O processamento do texto claro envolve oito rodadas e uma função de transformação final. O algoritmo divide a entrada em quatro sub-blocos de 16 bits. A cada rodada, quatro sub-blocos de 16 bits são tomados como entrada, e produzem quatro sub-blocos de saída de 16 bits. A transformação final também produz quatro blocos de 16 bits, que são concatenados formando um texto cifrado de 64 bits. Além dos sub-blocos, cada rodada utiliza dezesseis sub-chaves, enquanto a transformação final utiliza quatro sub-chaves. Durante o processo de codificação 52 sub-chaves são utilizadas, e todas elas são geradas a partir da chave inicial de 128 bits. A cada rodada três operações são utilizadas:

a) XOR bit-a-bit;

b) adição de inteiros módulo 2^{16}, com entradas e saídas tratadas como inteiros de 16 bits sem sinal, isto é, variáveis com valores somente positivos (*unsigned*);

c) multiplicação de inteiros módulo $2^{16} + 1$, com entradas e saídas tratadas como inteiros de 16 bits sem sinal, exceto o bloco constituído por zeros que é tratado como representação 2^{16}.

A estrutura geral do IDEA é dividida em quatro partes:

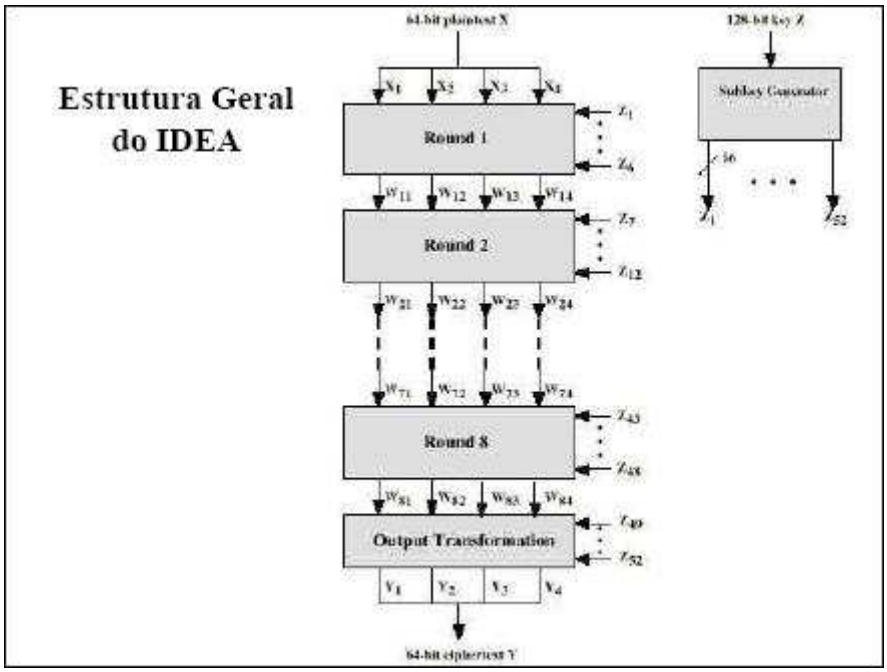
a) o texto claro de 64 bits é dividido em quatro partes;

b) a partir da chave de 128 bits são geradas 52 sub-chaves de 16 bits cada;

c) o IDEA é composto por oito rodadas, sendo que em cada rodada são aplicadas 6 sub-chaves, e a mesma sub-chave não é repetida;

d) ao final, os dados passam por uma transformação.

A figura 7 demonstra de forma simplificada a estrutura geral do IDEA, começando pelas rodadas iniciais com quatro entradas do texto claro (X_1, X_2, X_3, X_4) até a transformação final de saída que gera quatro saídas cifradas (Y_1, Y_2, Y_3, Y_4) de 16 bits cada, que compõem o texto cifrado completo de 64 bits. Para a geração do texto cifrado (Y1, Y2, Y3, Y4), são utilizadas as 52 sub-chaves (Z1..Z52) que são aplicadas nas operações de XOR, adição e multiplicação que compõe o IDEA.

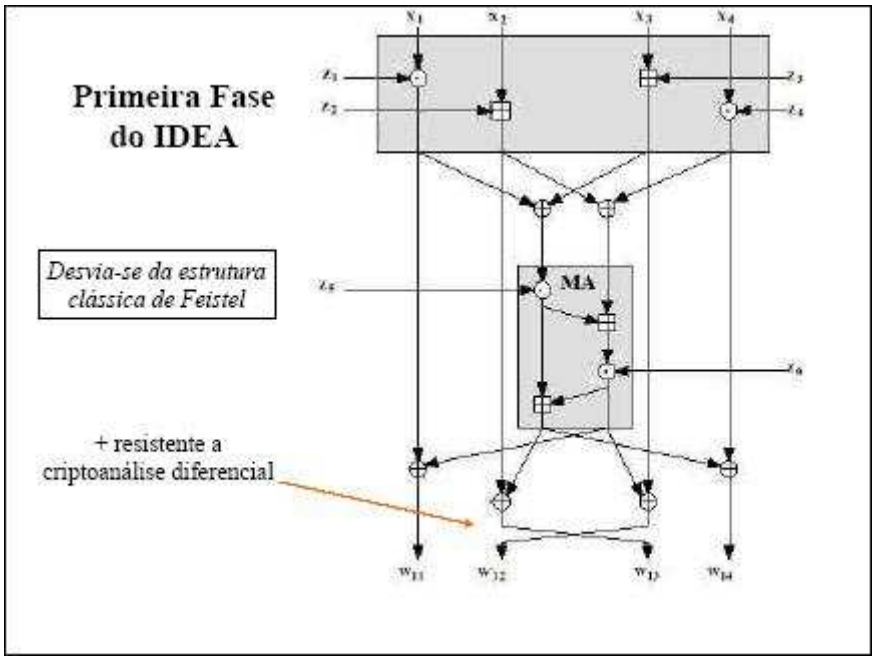


Fonte: Custódio (2003).

Figura 7 – Estrutura geral do IDEA

3.2 ETAPAS DO IDEA

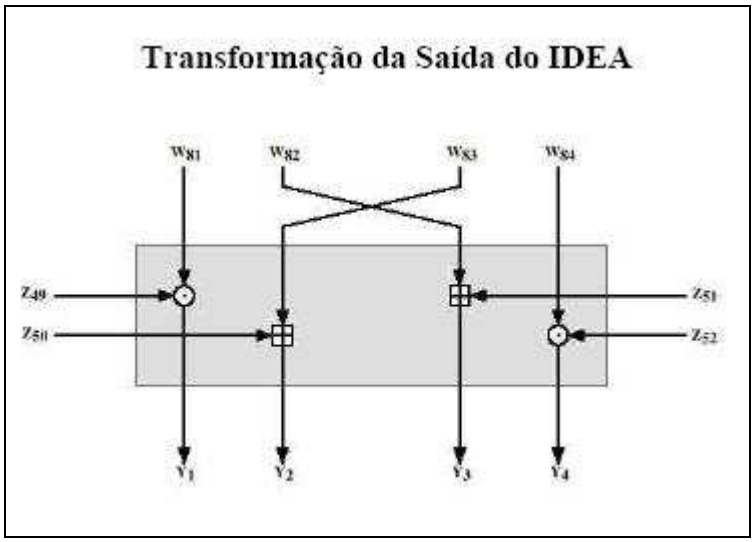
A figura 8 ilustra a primeira rodada do algoritmo IDEA. A primeira rodada do IDEA tem quatro entradas de 16 bits cada que sofrem XOR, adição, multiplicação. Durante o processo são utilizadas 6 sub-chaves sobre os blocos (X_1, X_2, X_3, X_4) gerando quatro blocos de 16 bits de saída. Pode-se destacar neste algoritmo que ele desvia-se da estrutura clássica de Feistel, e que a troca de posição de X_2 e X_3 torna mais resistente ao ataque diferencial (CUSTÓDIO, 2003, p. 59). As rodadas no IDEA são idênticas, ou seja, cada rodada aplica a mesma quantidade de operações matemáticas sobre os blocos (X1, X2, X3, X4) de entrada com 6 chaves (Z1, Z2, Z3, Z4, Z5, Z6) e cada rodada é dividida em 15 etapas até a geração dos blocos de saída (W11, W12, W13, W14).



Fonte: Custódio (2003).

Figura 8 – Primeira rodada do IDEA

A rodada do IDEA possui uma estrutura de multiplicação/adição (MA). Através de 2 operações de multiplicação e 2 operações de adição com a utilização de 2 sub-chaves, a estrutura de multiplicação/adição implementa o conceito de difusão, ou seja, cada bit de saída depende de todo bit de entrada e de todo bit da chave.



Fonte: Custódio (2003).

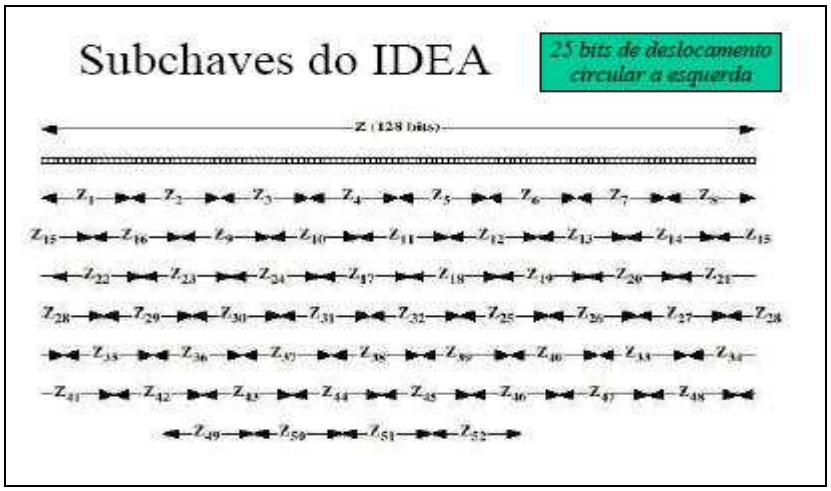
Figura 9 – Transformação da saída do IDEA

A figura 9 ilustra a transformação da saída do IDEA. Esta etapa se difere das demais pelo não uso do núcleo MA e por utilizar somente quatro sub-chaves.

3.3 GERAÇÃO DE CHAVES

A partir da chave inicial de 128 bits, o cifrador gera 52 sub-chaves de 16 bits cada.

Primeiramente a chave é dividida em oito sub-chaves de 16 bits cada, em seguida faz-se uma rotação circular à esquerda de 25 bits e divide-se novamente os 128 bits em oito sub-chaves de 16 bits, este processo se repete até gerar as 52 sub-chaves utilizadas no IDEA.

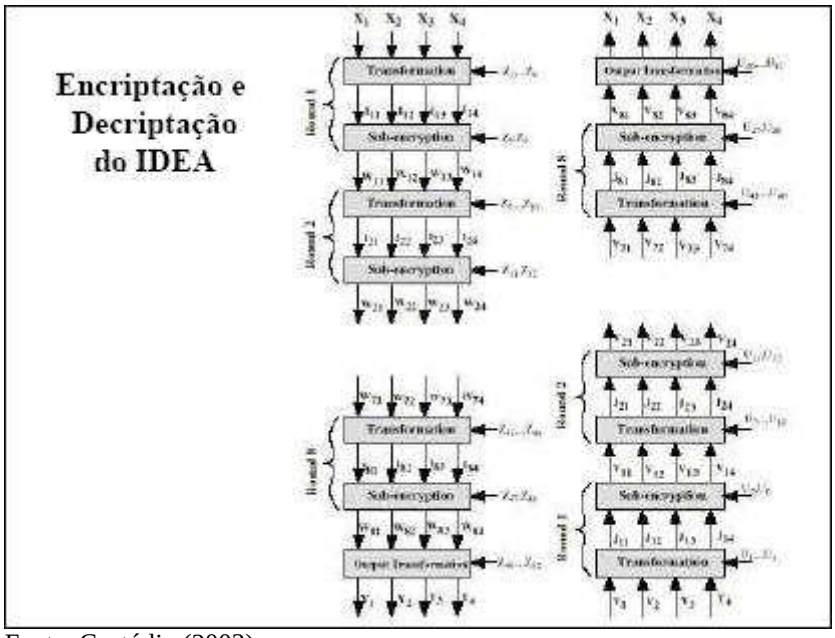


Fonte: Custódio (2003).

Figura 10 – Geração das sub-chaves no IDEA

A figura 10 demonstra a geração de chaves no IDEA, desde a divisão da chave inicial de 128 bits, até a última rotação circular de 25 posições que gerou as 52 sub-chaves.

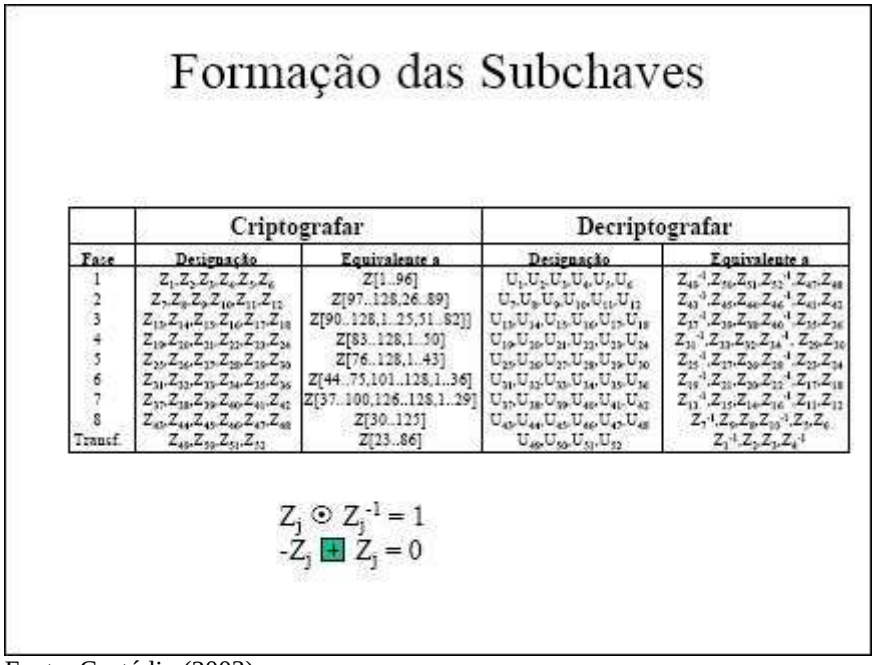
O processo de decriptação do IDEA é semelhante ao processo de encriptação. Como pode ser visto na figura 11, o algoritmo de decodificação possui a mesma estrutura que o processo de encriptação que é formado por 8 rodadas, sendo que cada rodada é dividida em 15 etapas. A diferença é que no processo de decriptação, as chaves utilizadas são geradas a partir das sub-chaves que são utilizadas na encriptação, que por sua vez, são geradas a partir da chave inicial.



Fonte: Custódio (2003).

Figura 11 – Funcionamento do IDEA

A figura 12 exibe o processo de formação das sub-chaves nas etapas de cifrar e decifrar. Nas fases de criptografar e decriptografar são usadas seis sub-chaves por rodada e apenas quatro na transformação final, a diferença está na geração destas sub-chaves, pois na geração das sub-chaves para o processo de decriptação usa-se a multiplicação inversa. As operações de multiplicação e de adição utilizadas no processo de formação das sub-chaves estão identificadas na legenda da figura 12 respectivamente.



Fonte: Custódio (2003).

Figura 12 – Formação das sub-chaves

3.4 MODOS DE OPERAÇÃO DO IDEA

O IDEA é um cifrador de blocos, portanto possui os modos ECB, CBC, CFB e OFB de operação. Para demonstrar os resultados dos diferentes modos de operação do IDEA são apresentadas seqüências de imagens que contêm textos pequenos e textos grandes.

A figura 13 não sofreu nenhuma transformação e está em sua forma original.

Fonte: Mediacrypt (2005).

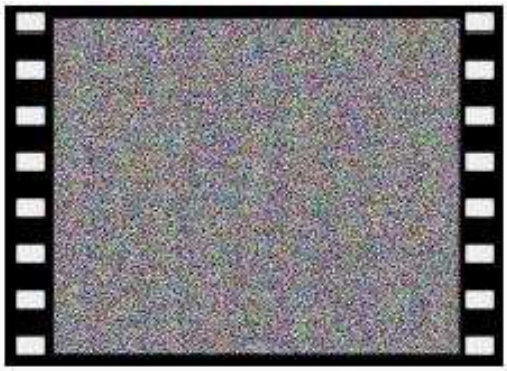
Figura 13 – Imagem original

A figura 14 foi codificada com o modo ECB do IDEA. Como pode ser visto, não esconde toda a informação da figura original. A imagem ainda pode ser reconhecida através

de seu esboço do gráfico e do texto contido na imagem.

Fonte: Mediacrypt (2005).
Figura 14 – Imagem encriptada com o modo ECB

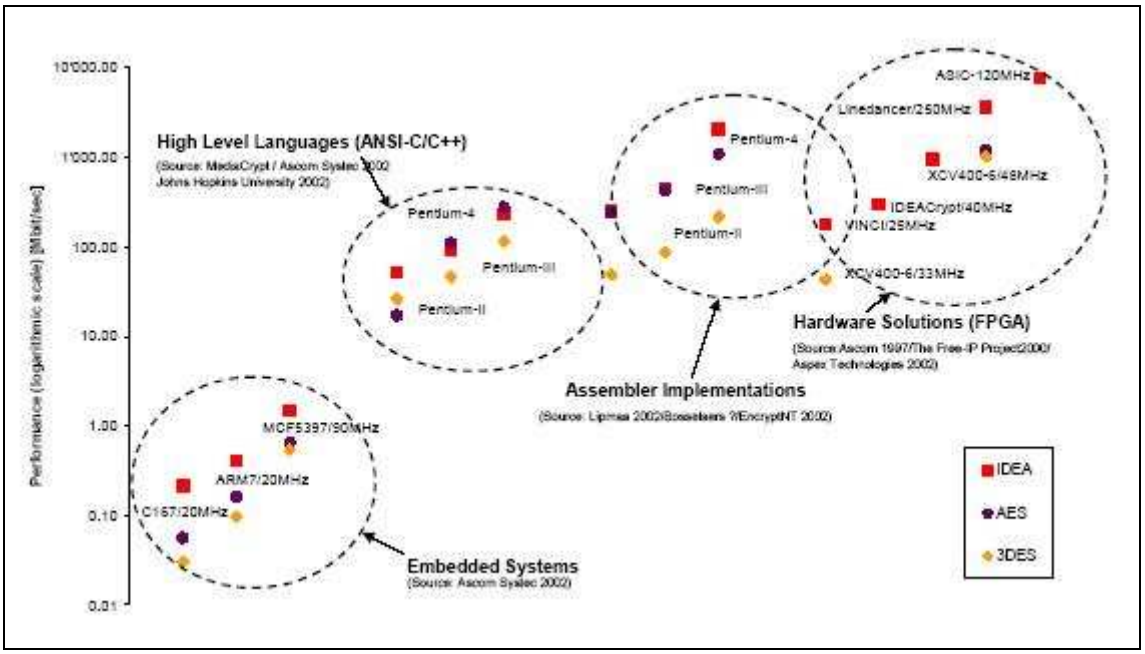
Usando o CBC, CFB ou OFB, toda a informação é escondida, tornando-se completamente invisível. Como pode ser visto na figura 15, estes são os melhores modos a serem usados quando a informação necessita estar completamente segura.



Fonte: Mediacrypt (2005).
Figura 15 – Imagem encriptada com o modo CBC

3.5 DESEMPENHO DO IDEA

O algoritmo de cifragem do IDEA é ideal para uso em sistemas embutidos com segurança de alto nível, pois é relativamente pequeno e extremamente rápido. Através da não utilização de operações com caixas S, que é uma das técnicas de espalhamento de bits utilizadas em alguns cifradores de blocos (DES, 3DES, AES), o IDEA necessita de menos memória em sua execução, melhorando o desempenho do cifrador (MEDIACRYPT, 2005).

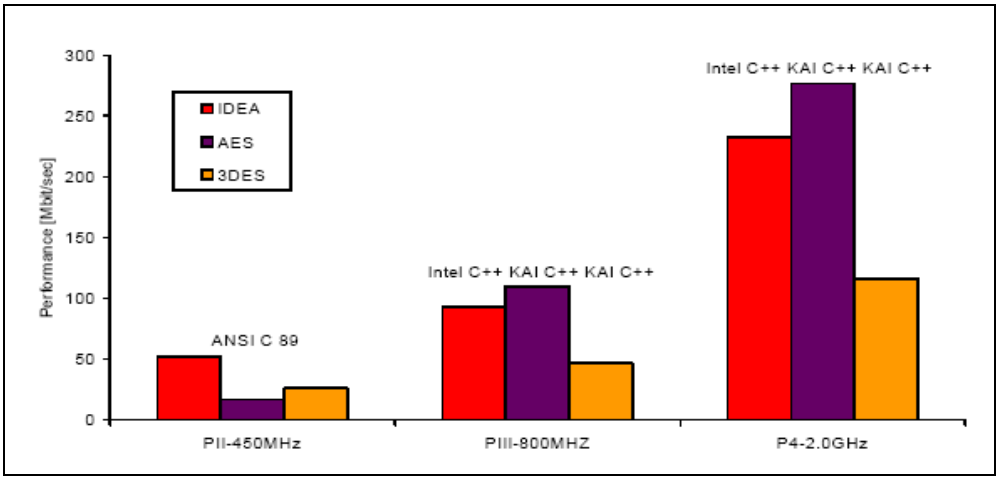


Fonte: Mediacrypt (2005).

Figura 16 – Desempenho geral do IDEA

A figura 16 apresenta uma comparação de desempenho entre os cifradores IDEA, 3DES e AES nos quesitos de sistemas embutidos, linguagens de alto nível, implementações em *assembler* e soluções de hardware. Como pode ser notado, o IDEA lidera em três dos quatro testes.

A figura 17 mostra uma comparação dos cifradores no quesito de linguagens de alto nível (C-ANSI / C++), do qual é o único teste de desempenho que apresenta certo equilíbrio entre os cifradores, lembrando que neste teste foram usados diferentes compiladores afetando o resultado final do teste.



Fonte: Mediacrypt (2005).

Figura 17 – Comparação de desempenho entre 3DES, AES e IDEA

Conforme apresentado na figura 17, têm-se uma comparação de desempenho dos três cifradores que surgiram para substituir o DES. Pode-se observar que os cifradores IDEA e AES tiveram uma performance semelhante nos três tipos de equipamentos utilizados para os testes, e o 3DES que utiliza a estrutura do DES obteve um desempenho inferior.

3.6 SEGURANÇA DO CIFRADOR IDEA

Desde a criação do cifrador IDEA, o mesmo tem sofrido diversos tipos de ataques, e obteve resistência perante todos os ataques sofridos. Nenhum outro algoritmo além do DES, que já foi quebrado, sofreu tantos testes como o IDEA (MEDIACRYPT, 2005).

Os ataques realizados no IDEA e os resultados destes ataques estão descritos abaixo:

a) ataque por força bruta: devido ao tamanho da chave de 2^{128}, o IDEA resistiu a este tipo de ataque;
b) criptoanálise diferencial: o *design* do IDEA dificulta este tipo de ataque, portanto resistiu com facilidades a este ataque;
c) chaves fracas: algumas chaves que contêm longas *strings* com zeros são relativamente mais fracas, porém a probabilidade de uma escolha aleatória de uma chave fraca é relativamente pequena, apenas uma entre 2^{76} possíveis;
d) criptoanálise linear: o cifrador IDEA permaneceu seguro perante a este tipo de ataque;
e) criptoanálise através de escolha de chaves: este ataque é similar ao ataque através de chaves fracas, e o IDEA manteve-se seguro ao ataque;
f) ataques com diferencias truncados: se o IDEA tivesse apenas três rodadas, este ataque poderia ter sucesso, porém o IDEA tem oito rodadas tornando-se extremamente seguro perante a este tipo de ataque;

g) criptoanálise diferencial-linear: este ataque também é similar ao ataque por chaves fracas e mais uma vez o IDEA resistiu a este tipo de ataque.

3.7 PRÓXIMA GERAÇÃO DO IDEA

A Mediacrypt, a qual possui a patente do cifrador IDEA, está desenvolvendo em parceria com uma universidade o IDEA NXT, que é uma família da próxima geração de cifradores simétricos que ajudará na segurança de mídias digitais, comunicações e armazenamentos.

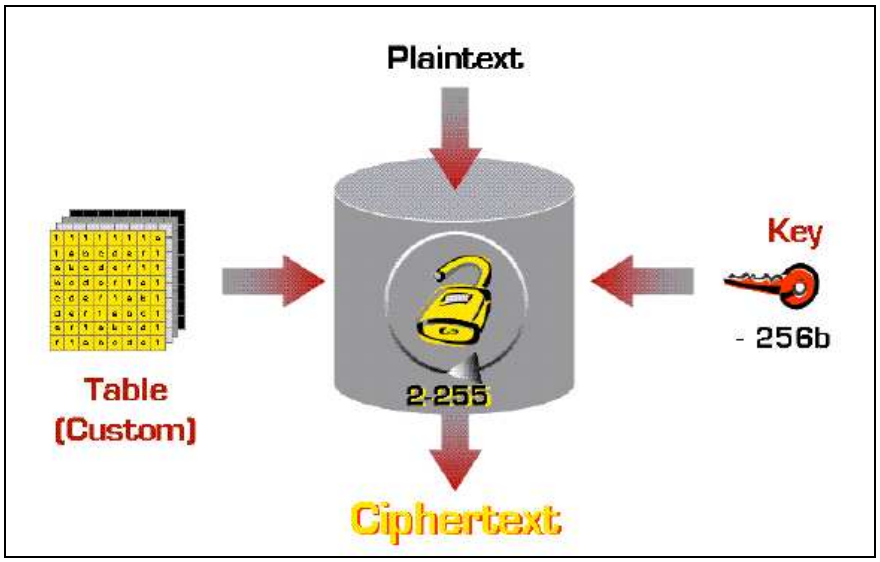
O IDEA NXT se caracteriza pela flexibilidade e pela escalabilidade de modos de operação, que podem otimizar dinamicamente seu funcionamento entre desempenho e segurança. O IDEA NXT oferece blocos de 64 e 128 bits com chaves de até 256 bits e podendo ser configurado para ter até 255 rodadas (MEDIACRYPT, 2005). A tabela 2 apresenta os possíveis valores de operação que o IDEA NXT pode assumir.

Tabela 2 - Possíveis tamanhos de blocos e de chaves do IDEA NXT

block size	key size	key schedule version
64	0 ≤ key ≤ 128	64-light
64	136 ≤ key ≤ 256	64-heavy
128	0 ≤ key ≤ 256	128

Fonte: Mediacrypt (2005).

A figura 18 apresenta uma visão geral do funcionamento do IDEA NXT, o qual aplica sobre o texto claro uma chave de 256 bits, permutação de matriz e 255 rodadas gerando o texto cifrado.



Fonte: Mediacrypt (2005).

Figura 18 – Funcionamento do IDEA NXT

Com um mecanismo inovador e diferenciado, o IDEA NXT difere-se do IDEA por possuir tamanhos de blocos e de chaves variáveis e pela utilização de caixas S em seu algoritmo (MEDIACRYPT, 2005).

3.8 TRABALHOS CORRELATOS

O ensino de cifradores complexos através de modelos simplificados teve como trabalho inicial de referência o modelo simplificado do DES, o S-DES, que foi desenvolvido pelo professor Edward Schaefer da Universidade de Santa Clara, Califórnia/EUA. Stallings (1998) difundiu mundialmente o trabalho com a inclusão no livro *Criptography And Network Security: Principles And Practice* e posteriormente o trabalho teve a publicação no *Journal of Cryptology* (MIERS, 2002, p. 22).

Há dois trabalhos correlacionados com o tema simplificação de cifradores.

Mendes (2001) desenvolveu um modelo simplificado para o cifrador RC6, gerando o SRC6. Este trabalho consistiu no estudo de outras simplificações do RC6, cifradores de bloco simétricos e os cifradores que antecederam o RC6 (RC2, RC4 e RC5) e teve como objetivo gerar uma simplificação coerente e didática do cifrador RC6.

Miers (2002) simplificou o cifrador AES, nomeado de SAES. Este trabalho teve como base o estudo dos princípios matemáticos e algébricos com a intenção de conseguir a maior redução possível do número de parâmetros para a compreensão da estrutura básica. O trabalho tinha como objetivo principal facilitar a compreensão do algoritmo AES aos estudantes de pós-graduação e fases finais de graduação.

Em relação a trabalhos correlatos referentes ao cifrador IDEA, não foi encontrado na literatura pesquisada nenhum trabalho que foi confeccionado através do estudo do IDEA, que obteve com o desenvolvimento final, uma simplificação do cifrador com fins didáticos ou qualquer simplificação com finalidades alternativas.

4 DESENVOLVIMENTO DO TRABALHO

Este capítulo tem por objetivo detalhar as etapas para o desenvolvimento do protótipo, que tem como primeiro tópico a descrição dos requisitos do protótipo, tanto funcionais quanto não funcionais. Após o levantamento de requisitos é detalhada a especificação, a implementação, os aspectos da operacionalidade e os testes do protótipo, finalizando com uma descrição dos resultados obtidos.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Para o desenvolvimento de um trabalho que propõe um modelo simplificado de um determinado cifrador, fez-se o levantamento de vários requisitos para o protótipo final do projeto. Para este trabalho que propõe um modelo reduzido do cifrador IDEA, os requisitos levantados foram divididos em funcionais e não funcionais.

4.1.1 Requisitos Funcionais

Os requisitos funcionais são:

a) permitir a fácil compreensão do cifrador IDEA original;
b) permitir a implementação manual dos processos cifrar e decifrar;
c) permitir a implementação manual dos processos de geração de chaves e sub-chaves;
d) cifrar e decifrar textos e dados;
e) servir como referência para a proposição de outros modelos simplificados de cifradores.

4.1.2 Requisitos não funcionais

Os requisitos não funcionais são:

a) facilitar a apresentação do cifrador IDEA;

b) manter as características funcionais do IDEA com parâmetros reduzidos;

c) possuir estrutura simples e estável;

d) necessitar baixo requisito de *hardware* para o funcionamento do protótipo;

e) utilizar a linguagem de programação C com o padrão ANSI;

f) utilizar ferramentas e bibliotecas gratuitas e preferencialmente de código aberto;

g) ser de domínio público e estar exposto para estudos.

4.2 ESPECIFICAÇÃO

Os tópicos seguintes descrevem a especificação do protótipo. A especificação foi realizada através de fluxogramação no MS-VISIO (VISIO, 2003). Inicialmente é descrita a metodologia que será utilizada na redução do cifrador, e em seguida são descritas as etapas de redução do tamanho da chave, redução do tamanho do bloco, processo de geração das chaves, diminuição do número de rodadas, transformação final do cifrador e por fim como serão trabalhadas as operações de XOR, adição e multiplicação do protótipo e as características entre o cifrador padrão e o modelo simplificado. Também é apresentada a segurança do modelo simplificado.

4.2.1 Metodologia de redução

Ao se fazer uma redução de um cifrador, deve-se ter em mente que a simplificação

deve estar totalmente correlacionada com o modelo do cifrador envolvido. As operações matemáticas primitivas não podem ser modificadas, e os princípios do cifrador não podem ser alterados. Outro ponto que merece destaque é que a simples redução de bits nas chaves, blocos e *strings* de operações não são suficientes, tendo-se em vista a necessidade que esta redução seja proporcional ao tamanho contido no cifrador e que tenha correlação entre os modelos simplificados e o modelo padrão (HOFFMAN, 2005).

No caso do IDEA, reduzir simplesmente as operações contidas no cifrador fará com que a versão simplificada não tenha correlação e com isso as características de confusão e de difusão do IDEA não estarão presentes no modelo simplificado, características que tornam o IDEA resistente a diversos tipos de criptoanálise.

A metodologia empregada na simplificação do IDEA está baseada na maior redução possível nos processos do cifrador e na quantidade de bits envolvidos nestes processos sem descaracterizar o IDEA, mas de modo que seja possível tornar o protótipo didático para implementações manuais.

4.2.2 Redução da chave

Como a chave do IDEA possui 128 bits, torna-se inviável uma implementação manual do cifrador. Para o modelo simplificado, deu-se um tamanho de chave de 32 bits proporcionalmente reduzido e sem alterar seu modo de operação dentro do cifrador.

4.2.3 Redução do bloco

O IDEA trabalha com blocos de 64 bits, para o modelo reduzido foi adotado o bloco com 16 bits, respeitando a proporção de redução do tamanho da chave que é utilizado no

modelo simplificado. Como é feito no IDEA, o modelo simplificado também divide o bloco em quatro sub-blocos (X1, X2, X3, X4), porém os sub-blocos possuem apenas 4 bits e não mais 16 bits.

4.2.4 Geração das sub-chaves

No cifrador IDEA são geradas 52 sub-chaves a partir da chave inicial de 128 bits utilizando o deslocamento de 25 posições à esquerda, gerando oito sub-chaves por deslocamento. No modelo simplificado a chave inicial possui 32 bits e um deslocamento de 7 posições à esquerda para a geração de 28 sub-chaves.

A correlação entre os modelos pode ser observada pelo fato de que a chave inicial também é dividia em oito blocos, porém blocos reduzidos a 4 bits. Outra característica que permanece no modelo simplificado é que o deslocamento de posições detém as mesmas proporções do modelo padrão. Com o *shift* de 25 posições no IDEA, faz-se aproximadamente um deslocamento de 1.694 possíveis chaves e algo em torno de 19% de deslocamento do tamanho da chave inicial de 128 bits. No modelo simplificado o *shift* é de 7 posições e, portanto, tem-se um deslocamento aproximado de 1.875 possíveis chaves e 21.8% de deslocamento do tamanho da chave inicial de 32 bits. O deslocamento de 7 posições é o menor possível para a chave de 32 bits e ainda garante que as sub-chaves não sejam repetidas, tendo correlação com o IDEA original (HOFFMAN, 2005).

Na figura 19 está descrito o processo de geração das sub-chaves de encriptação no modelo simplificado, dividindo a chave inicial de 32 bits em oito blocos de 4 bits e posteriormente faz-se o deslocamento a esquerda de 7 posições gerando as próximas 8 sub-chaves. Ao todo são geradas 28 sub-chaves que serão utilizadas no processo de cifragem. A correlação entre os números de sub-chaves utilizados nos cifradores IDEA e no SIDEA

encontra-se ao número de rodadas de cada cifrador, portanto o número de sub-chaves utilizados é, respectivamente, 52 e 28.

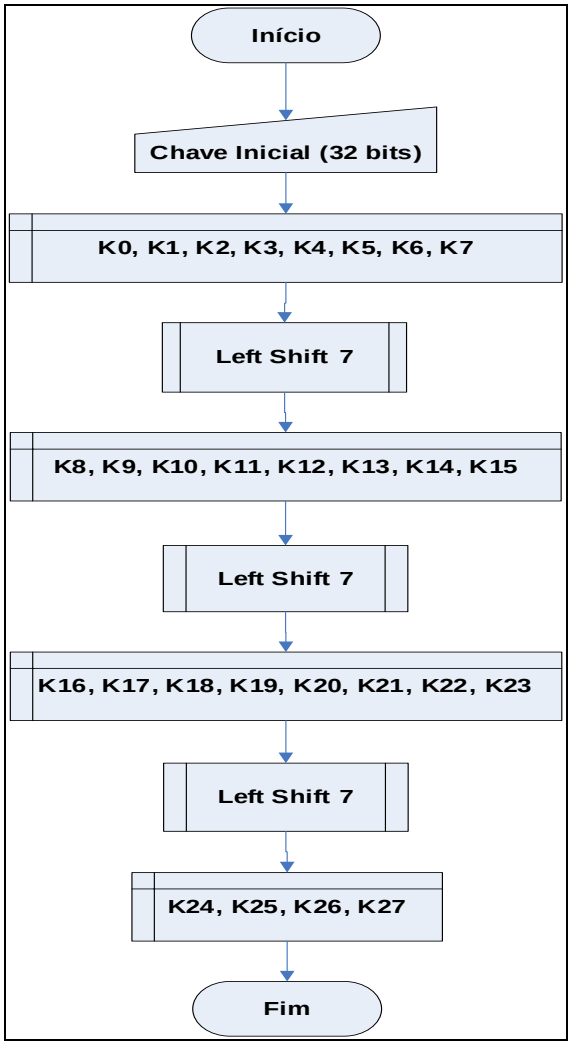


Figura 19 – Geração das chaves de encriptação

Para a geração das sub-chaves de decriptação, o modelo simplificado utiliza a mesma lógica encontrada no IDEA. O cifrador gera as chaves de decriptação a partir das chaves de encriptação, aplicando algumas operações como a inversão de valores e a função de multiplicação inversa (mul_inv). A figura 20 representa a geração das chaves de decriptação, pode-se perceber que o cifrador aplica as operações sobre as chaves de encriptação utilizando uma ordem não seqüencial para a geração das chaves de decriptação, mantendo a correlação com o IDEA.

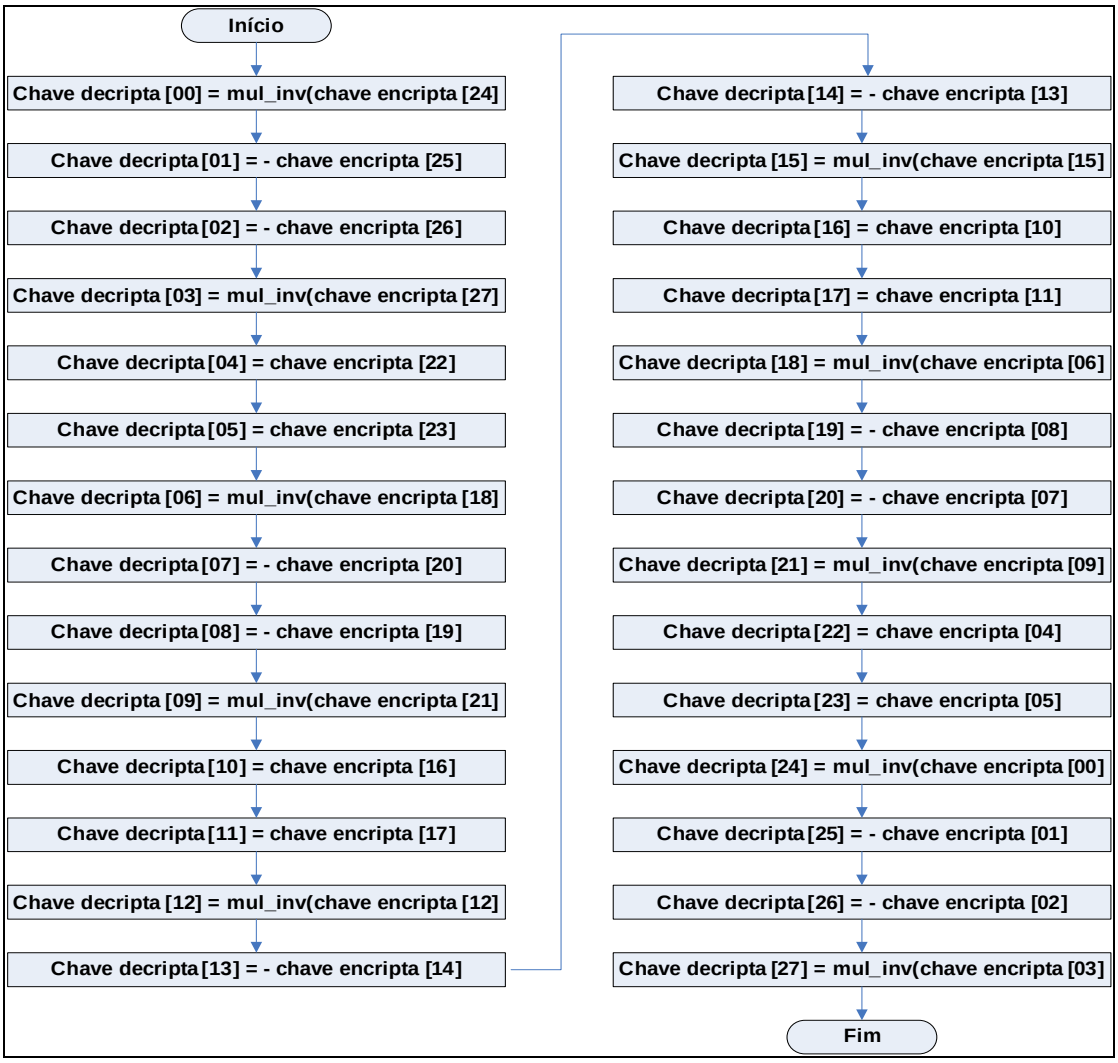


Figura 20 – Geração das chaves de decriptação

4.2.5 Número de rodadas

Para acompanhar a redução e funcionar adequadamente com a chave e os blocos com tamanhos reduzidos e principalmente para o modelo simplificado ser de fácil implementação manual, a quantidade de rodadas do cifrador também foi reduzida, enquanto o IDEA possui 8 rodadas, o modelo simplificado passa a ter apenas 4 rodadas. O número de rodadas foi proporcionalmente reduzido devido ao número de sub-chaves geradas no modelo simplificado, pois cada rodada utiliza 6 sub-chaves e a transformação final utiliza apenas 4 sub-chaves, totalizando as 28 chaves que foram geradas no SIDEA.

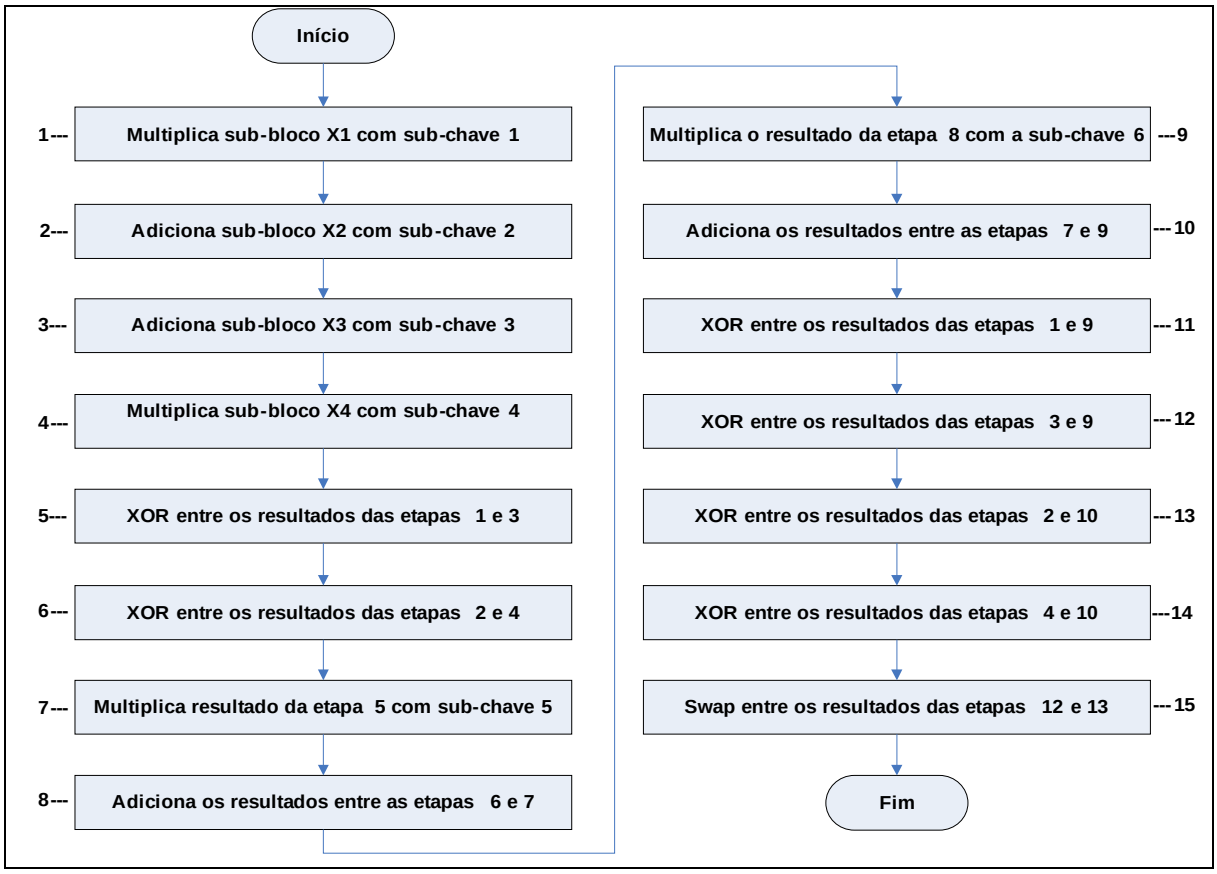


Figura 21 – Rodada do SIDEA

O modelo simplificado também possui 15 etapas de cifragem como no modelo padrão. A figura 21 descreve todos os processos envolvidos nas etapas de cada uma das quatro rodadas que o modelo simplificado possui. As rodadas são idênticas, a única diferença entre uma rodada e outra são as sub-chaves utilizadas, lembrando que uma sub-chave é utilizada uma única vez durante todo o processo de cifragem. Na primeira rodada são utilizadas as sub-chaves 1 à 6, já na segunda rodada são utilizadas as chaves 7 à 14 e assim sucessivamente até a rodada de número 4.

4.2.6 Transformação final

A etapa de transformação final do cifrador não sofreu alterações, visto que é a única etapa do cifrador que não possui a operação XOR, e como nesta etapa o cifrador já está trabalhando com blocos reduzidos a 4 bits, as duas operações de adição e as duas de

multiplicação permanecem inalteradas como no cifrador padrão mas com operandos de tamanhos proporcionalmente reduzidos.

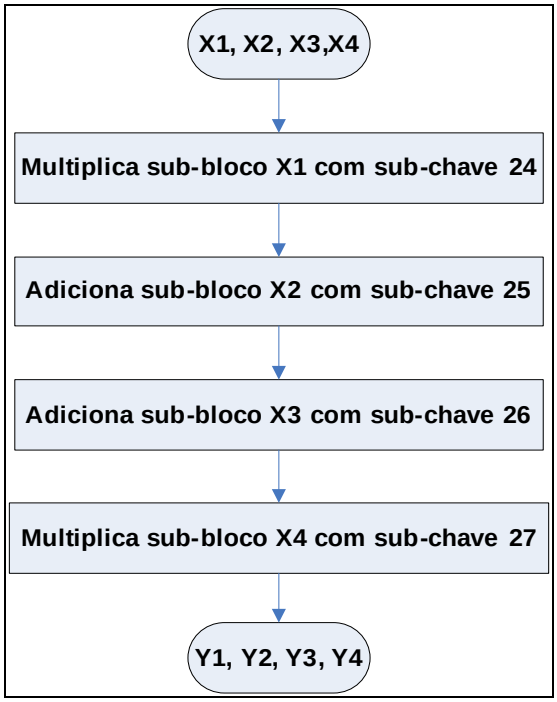


Figura 22 – Transformação final

Conforme descrito a figura 22, pode ser visto que a transformação final do cifrador não sofreu modificações, ou seja, transformação continua com quatro etapas, sendo duas multiplicações e duas adições. Faz-se a adição entre o sub-bloco 2 com a sub-chave 26 e a adição do sub-bloco 3 com a sub-chave 27 e a multiplicação do primeiro e do último sub-bloco com as sub-chaves 25 e 28 respectivamente.

4.2.7 Operações matemáticas

As operações matemáticas contidas no IDEA operam com blocos de 16 bits, como o modelo simplificado tem o tamanho da chave e o tamanho do bloco reduzidos, as operações matemáticas operam com blocos de 4 bits apenas. No IDEA a operação de adição é realizada em mod 65536 (2^{16}), enquanto no modelo simplificado é realizada em mod 16 (2^4) e a multiplicação no IDEA é efetuada em mod 65537 ($2^{16}+1$), já no modelo simplificado a

multiplicação é efetuada em mod 17 (2^4+1).

Tanto no IDEA quanto no modelo simplificado, as operações de multiplicação são efetuadas sobre mod de números primos, 65537 e 17, respectivamente. A necessidade de se ter um número primo é a de encontrar-se um multiplicador inverso, que é utilizado no cálculo das sub-chaves de encriptação.

A tabela 3 exibe uma comparação entre os cifradores DES e o IDEA e seus respectivos modelos simplificados.

Tabela 3 – Comparação dos cifradores padrões com suas simplificações

CIFRADORES				
	S-DES	DES	SIDEA	IDEA
Bloco	8 bits	64 bits	16 bits	64 bits
Chave	10 bits	56 bits	32 bits	128 bits
Sub-Chave	8 bits	48 bits	4 bits	16 bits
Rodadas	2	16	4	8

4.2.8 Segurança no SIDEA

Conforme descrito anteriormente, o modelo simplificado tem como objetivo auxiliar o ensino do cifrador IDEA e com isso possibilitar sua implementação manual, cifrando textos e dados com operações de 4 bits. Devido ao tamanho de 32 bits da chave inicial, o SIDEA não provê segurança, ou seja, qualquer texto cifrado com o SIDEA pode facilmente ser quebrado por diferentes tipos de criptoanálise, inclusive criptoanálise por força bruta, que testaria todos os bits possíveis da chave em um tempo relativamente curto.

4.2.9 Características entre o IDEA e o SIDEA

Conforme descrito na metodologia utilizada para confeccionar o modelo simplificado, tem-se como prioridade a permanência das principais características do IDEA no cifrador reduzido, ou seja, ambos devem ter correlação.

Tabela 4 – Características presentes no IDEA e no SIDEA

Características	**IDEA**	**SIDEA**
XOR/Adição/ Multiplicação	SIM	SIM
15 Etapas por rodada	SIM	SIM
Sub-chaves não repetidas	SIM	SIM
Multipl. c/ número primo	SIM	SIM
Divisão s-chave/s-bloco	SIM	SIM
Transformação final	SIM	SIM

Na tabela 4 estão descritas todas as características contidas no IDEA e no SIDEA. As principais operações matemáticas não foram alteradas, apenas os operandos tiveram seus tamanhos reduzidos. As etapas de cada rodada permaneceram inalteradas, porém os sub-blocos e as sub-chaves utilizadas possuem 4 bits ao invés dos 16 bits do IDEA. No modelo simplificado também foram implementadas a função de multiplicação a base de número primo e a transformação final. Através do deslocamento circular de 7 posições conforme descrito no item 5.2.4, o SIDEA não gera sub-chaves repetidas. Outra correlação com o IDEA, é que a primeira operação sobre a chave inicial e o bloco de entrada é a divisão em sub-blocos de mesmo tamanho, no modelo simplificado o tamanho é 4 bits.

4.3 IMPLEMENTAÇÃO

Nesta seção são apresentados os aspectos sobre a implementação do protótipo e as ferramentas utilizadas para a construção.

4.3.1 Técnicas e ferramentas utilizadas

Para implementação do protótipo foi utilizado o ambiente de desenvolvimento Bloodshed Dev C++. A linguagem de programação utilizada é C com o padrão ANSI. Como o protótipo foi desenvolvido em plataforma Windows, foram utilizadas as bibliotecas stdio.h e stdlib.h que são nativas do C. O protótipo não possui diferentes modos de operação (ver seção 3.4), a implementação do SIDEA foi realizada com base nos modos de operação ECB e CBC.

Nas próximas sessões são apresentados conceitos que sejam relevantes para o entendimento do protótipo, seguidos de trechos de códigos que implementam as funções utilizadas no protótipo para o seu total entendimento. O código completo implementado em linguagem C com o padrão ANSI que compõe o SIDEA encontra-se em anexo. No Apêndice A, encontra-se o código que implementa a função cifra do SIDEA e no Apêndice B encontra-se o código que implementa a função decifra do SIDEA.

4.3.1.1 Velocidade do Algoritmo

Um dos requisitos para os algoritmos que compõem cifradores é a velocidade do mesmo. Um cifrador não é considerado aplicável se seu algoritmo é eficiente e não tenha um bom desempenho, ou seja, o cifrador é avaliado em uma escala de custo/benefício entre eficiência e performance. Com o problema identificado, a implementação do protótipo foi

realizada em médio e baixo nível, ou seja, a implementação possui funções e operadores binários que provêem um desempenho superior as funções e operadores lógicos. Os números em formato hexadecimal (0xff) também são utilizados na implementação como são normalmente encontrados em implementações de cifradores.

```
for (i=0;i<8;i++)
       key32 = ((chave_entrada[i] & 0x0f) << (4*i)) | key32;

for (i;i<SIDEA_SUBCHAVES;i++) {
       key32 = i % 8 ? key32 : (key32 << 7) | (key32  >> 25);
       chave_entrada[i] = (key32 << (7-(i%8))*4) >> 28;
   }
for (i=0;i<SIDEA_SUBCHAVES;i++) {
    i % 8 ? printf("\t") : printf("\n");
    printf("0x%x",chave_entrada[i]);
}
printf("\n");
}
```

Quadro 1 – Código com operadores binários

No quadro 1 tem-se um trecho do código implementado que utiliza operadores binários. Os operadores binários utilizados neste código (&, <<, |, %, ?, :, >>) implementam operações como: deslocamento de bits, condições lógicas (IF) e operações lógicas (OR, AND).

4.3.1.2 Algoritmo de Euclides

O algoritmo de Euclides é utilizado na implementação do cifrador IDEA, e consequentemente foi utilizado no modelo simplificado do cifrador. É um dos algoritmos mais antigos conhecidos, desde que apareceu na obra Elementos de Euclides por volta de 300 aC (WIKIPEDIA, 2000).

O algoritmo de Euclides refere-se ao máximo divisor comum de dois números inteiros diferentes de zero. O algoritmo se baseia no seguinte fato: se *c* é resto, quando *a* é dividido por *b* então o máximo divisor comum de *a* e *b* é igual ao máximo divisor comum de *b* e *c.*

No quadro 2 é apresentado o algoritmo de Euclides implementando a operação de multiplicação inversa utilizada no protótipo, a qual é uma das três operações aritméticas que o IDEA possui.

```
static uint4 mul_inv(uint4 x){
    int8 n1 = MUL_MOD;
    int8 n2 = (int8)x;
    int8 b1 = 0;
    int8 b2 = 1;
    int8 q, r, t;

    if (x <= 1)
        return x;
    while (1) {
        r = n1 % n2;
        q = n1 / n2;
        if (!r) {
            if (b2 < 0) b2 += MUL_MOD;
            return LSW4(b2);
        }
        else {
            n1 = n2;
            n2 = r;
            t  = b2;
            b2 = b1 - q * b2;
            b1 = t;
        }
    }
}
```

Quadro 2 – Algoritmo de Euclides

4.3.1.3 Geração das sub-chaves de encriptação

A função de geração de chaves divide a chave inicial em 8 blocos, gerando as chaves de 0 a 7. Depois a função aplica o LSHIFT 7, deslocando 7 posições à esquerda dos 32 bits da chave e gera as próximas chaves, divide-se em 8 blocos novamente até gerar as 28 chaves que o cifrador aplica na cifragem.

No quadro 3 está demonstrado como foi implementada a função que gera as 28 sub-chaves (*chave_entrada*) utilizadas no cifrador durante o processo de cifragem.

```
void gera_chaves_cifra(){

for (i=0;i<8;i++)
       // Divide a chave inicial de 32 bits em 8 blocos de 4 bits
       key32 = ((chave_entrada[i] & 0x0f) << (4*i)) | key32;

for (i;i<SIDEA_SUBCHAVES;i++) {
       // Deslocamento a esquerda de 7 posicoes
       key32 = i % 8 ? key32 : (key32 << 7) | (key32  >> 25);
       chave_entrada[i] = (key32 << (7-(i%8))*4) >> 28;
   }
printf("\nChaves de encriptacao\n");
for (i=0;i<SIDEA_SUBCHAVES;i++) {
    i % 8 ? printf("\t") : printf("\n");
    printf("0x%x",chave_entrada[i]);
}
printf("\n");
}
```

Quadro 3 – Geração das chaves de encriptação

4.3.1.4 Geração das sub-chaves de decriptação

A função *gera_chaves_decifra* gera as chaves que serão utilizadas no processo de decifragem a partir das chaves geradas anteriormente na função *gera_chaves_cifra.* Para isso faz-se a multiplicação inversa e a inversão de valores sobre cada chave de cifragem, gerando uma chave de decifragem correspondente, porém a ordem de geração não segue uma seqüência, ou seja, a primeira chave de cifragem não tem correlação com a primeira chave de decifragem. Este procedimento é realizado com todas as chaves de cifragem gerando assim as 28 chaves de decifragem.

No quadro 4 está descrito o código que foi utilizado para implementar a função que gera as 28 sub-chaves de decriptação (*decrypt_subkeys*) utilizadas na decriptação do cifrador. Pode-se notar que as sub-chaves de decriptação (*decrypt_subkeys*) são geradas a partir das sub-chaves de encriptação (*encrypt_subkeys*).

```
void gera_chaves_decifra(unsigned char *encrypt_subkeys,unsigned char *decrypt_subkeys){

  decrypt_subkeys[0 ] = mul_inv(encrypt_subkeys[24]);
  decrypt_subkeys[1 ] =   LSW4(-encrypt_subkeys[25]);
  decrypt_subkeys[2 ] =   LSW4(-encrypt_subkeys[26]);
  decrypt_subkeys[3 ] = mul_inv(encrypt_subkeys[27]);
  decrypt_subkeys[4 ] =    LSW4(encrypt_subkeys[22]);
  decrypt_subkeys[5 ] =    LSW4(encrypt_subkeys[23]);

  decrypt_subkeys[6 ] = mul_inv(encrypt_subkeys[18]);
  decrypt_subkeys[7 ] =   LSW4(-encrypt_subkeys[20]);
  decrypt_subkeys[8 ] =   LSW4(-encrypt_subkeys[19]);
  decrypt_subkeys[9 ] = mul_inv(encrypt_subkeys[21]);
  decrypt_subkeys[10] =    LSW4(encrypt_subkeys[16]);
  decrypt_subkeys[11] =    LSW4(encrypt_subkeys[17]);

  decrypt_subkeys[12] = mul_inv(encrypt_subkeys[12]);
  decrypt_subkeys[13] =   LSW4(-encrypt_subkeys[14]);
  decrypt_subkeys[14] =   LSW4(-encrypt_subkeys[13]);
  decrypt_subkeys[15] = mul_inv(encrypt_subkeys[15]);
  decrypt_subkeys[16] =    LSW4(encrypt_subkeys[10]);
  decrypt_subkeys[17] =    LSW4(encrypt_subkeys[11]);

  decrypt_subkeys[18] = mul_inv(encrypt_subkeys[6]);
  decrypt_subkeys[19] =   LSW4(-encrypt_subkeys[8]);
  decrypt_subkeys[20] =   LSW4(-encrypt_subkeys[7]);
  decrypt_subkeys[21] = mul_inv(encrypt_subkeys[9]);
  decrypt_subkeys[22] =    LSW4(encrypt_subkeys[4]);
  decrypt_subkeys[23] =    LSW4(encrypt_subkeys[5]);

  decrypt_subkeys[24] = mul_inv(encrypt_subkeys[0]);
  decrypt_subkeys[25] =   LSW4(-encrypt_subkeys[1]);
  decrypt_subkeys[26] =   LSW4(-encrypt_subkeys[2]);
  decrypt_subkeys[27] = mul_inv(encrypt_subkeys[3]);
```

Quadro 4 – Geração das chaves de decifragem

4.3.1.5 Função cifrar/decifrar do cifrador

A função *sidea_cifra* do protótipo utiliza as três operações lógicas encontradas no IDEA, ou seja, esta função implementa as operações matemáticas de multiplicação, adição e XOR, sendo que estas operações no SIDEA utilizam operandos de 4 bits. A função *sidea_cifra* implementa a operação de encriptação e de decriptação no protótipo, sendo a única diferença os parâmetros que são passados para a função. Para cifrar passam-se os parâmetros chave cifra (*chave*) e bloco plano de entrada (*bloco_in*), já para decifrar passam-se os parâmetros chave decifra (*chave*) e bloco cifrado de entrada (*bloco_in*).

```
void sidea_cifra(unsigned char *bloco_in, unsigned char *bloco_out, unsigned char *chave){

    uint4 *pin = (uint4*)bloco_in;
    uint4 *pout = (uint4*)bloco_out;
    uint4 *pk = (uint4*)chave;
    uint4 word1, word2, word3, word4;
    uint4 t1, t2;
    int i;

    word1 = *pin++;
    word2 = *pin++;
    word3 = *pin++;
    word4 = *pin;

    for (i=SIDEA_RODADAS;i>0;i--) {
        word1 = mul(word1,*pk++);
        word2 = LSW4(word2 + *pk++);
        word3 = LSW4(word3 + *pk++);
        word4 = mul(word4,*pk++);

        t2 = LSW4(word1 ^ word3);
        t2 = mul(t2,*pk++);
        t1 = LSW4(t2 + (word2 ^ word4));
        t1 = mul(t1,*pk++);
        t2 = LSW4(t1 + t2);

        word1 ^= t1;
        word4 ^= t2;

        t2 ^= word2;
        word2 = LSW4(word3 ^ t1);
        word3 = t2;
    }

    word1 = mul(word1,*pk++);
    *pout++ = word1;
    *pout++ = LSW4(word3 + *pk++);
    *pout++ = LSW4(word2 + *pk++);
    word4 = mul(word4,*pk);
    *pout = word4;
}
```

Quadro 5 – Função que implementa a cifragem e a decifragem

Como pode ser visto no quadro 5, a função cifrar/decifrar do protótipo aplica as sub-chaves geradas (*chave)* anteriormente sobre os blocos de entrada (*bloco_in*), e com isso obtêm-se os blocos de saída (*bloco_out*).

4.3.1.6 Multiplicação em módulo $2^4 + 1$

A multiplicação é uma das principais operações do cifrador juntamente com a operação de adição e com o XOR. A multiplicação é realizada em mod 17, ou seja, em

módulo 2^4 + 1 e utiliza a variável definida como LSW4 que garante que o código trabalhe com operandos de apenas 4 bits (*nibbles*). O quadro 6 exibe o código implementado nesta função, pode-se notar que a utilização da função LSW4, que utiliza apenas os 4 bits menos significantes de cada byte. Já a função MSW4 utiliza apenas os 4 bits mais à esquerda de cada byte (8 bits), ou seja, os 4 bits mais significantes.

```
static uint4 mul(uint4 x, uint4 y){

   uint4 t4;
   uint8 t8;

   x =  LSW4(x - 1);
   t4 = LSW4(y - 1);
   t8 = (uint8) x * t4 + x + t4 + 1;
   x = LSW4( t8 );
   t4 = MSW4( t8 );
   x = LSW4((x - t4) + (x <=t4));
   return x;
}
```

Quadro 6 – Multiplicação de módulo 2^4 + 1

4.3.2 Operacionalidade da implementação

Conforme descrito no item 5.3.1, o protótipo foi dividido em 2 aplicações. O SIDEA_CIFRA aplica a função de cifrar do protótipo, já o SIDEA_DECIFRA a função de decifrar.

O protótipo interage diretamente com o usuário, ou seja, para o protótipo cifrar ou decifrar, o usuário necessita informar a chave inicial e o bloco de entrada de operação. A chave e o bloco são informados com valores entre 0 e 255 separados por espaços, cada valor é de 1 byte (8 bits), para a chave são informados 4 valores (32 bits) e para o bloco apenas 2 (16 bits). Para tornar mais prático o entendimento do cifrador, foi definida a chave inicial com os valores {11, 22, 33, 44} e o bloco de entrada com os valores {10, 20} para um estudo de caso. O protótipo exibe algumas caixas que explicam os processos que o cifrador executará e

também os resultados gerados pelo cifrador. Ao final, a aplicação gera um arquivo texto com informações para o entendimento do protótipo e com os valores trabalhados durante as etapas. A figura 23 apresenta o funcionamento do SIDEA_CIFRA.

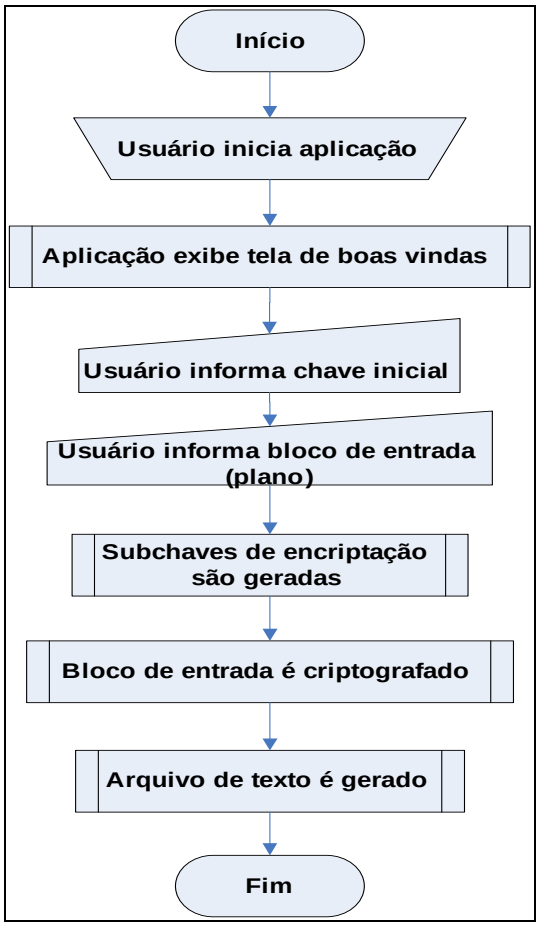


Figura 23 – Funcionamento SIDEA_CIFRA

A tela inicial do SIDEA_CIFRA é uma tela de boas vindas. Nesta tela descreve-se a origem do cifrador, suas características, tal como plataforma e ambiente em que o protótipo foi gerado. A tela também exibe instruções antes do usuário prosseguir para a próxima tela. Todas as telas sofrem intervenção da função *system ("pause")*, que através do *prompt* do DOS paralisa a tela até que o usuário deseje que a aplicação prossiga. A figura 24 mostra a tela inicial em execução.

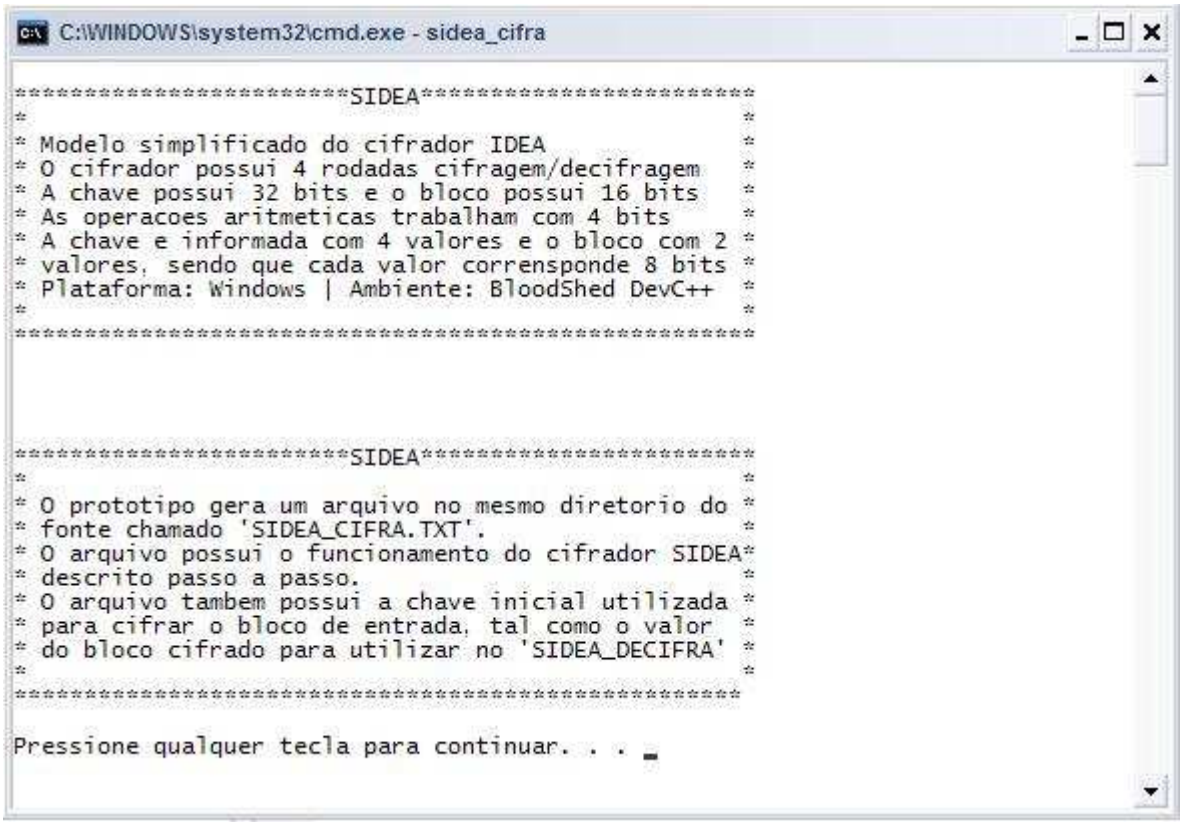


Figura 24 – Tela de boas vindas do SIDEA_CIFRA

Depois que o usuário intervir na tela inicial, a aplicação mostra uma nova tela. Nesta tela o usuário informa a chave inicial e o bloco de entrada em valores no formato decimal. Após a entrada dos valores pelo usuário, a aplicação executa a função *gera_chaves_cifra* e exibe em tela os valores das chaves de encriptação em notação hexadecimal. Após calcular as chaves, a aplicação executa a função *sidea_cifra* e exibe em tela o valor do bloco cifrado que será a entrada para o SIDEA_DECIFRA. Ao final a aplicação gera o arquivo SIDEA_CIFRA.TXT no mesmo diretório em que se encontra a aplicação SIDEA_CIFRA com informações didáticas sobre as operações efetuadas com os valores de entrada. A figura 25 mostra a tela citada em execução.

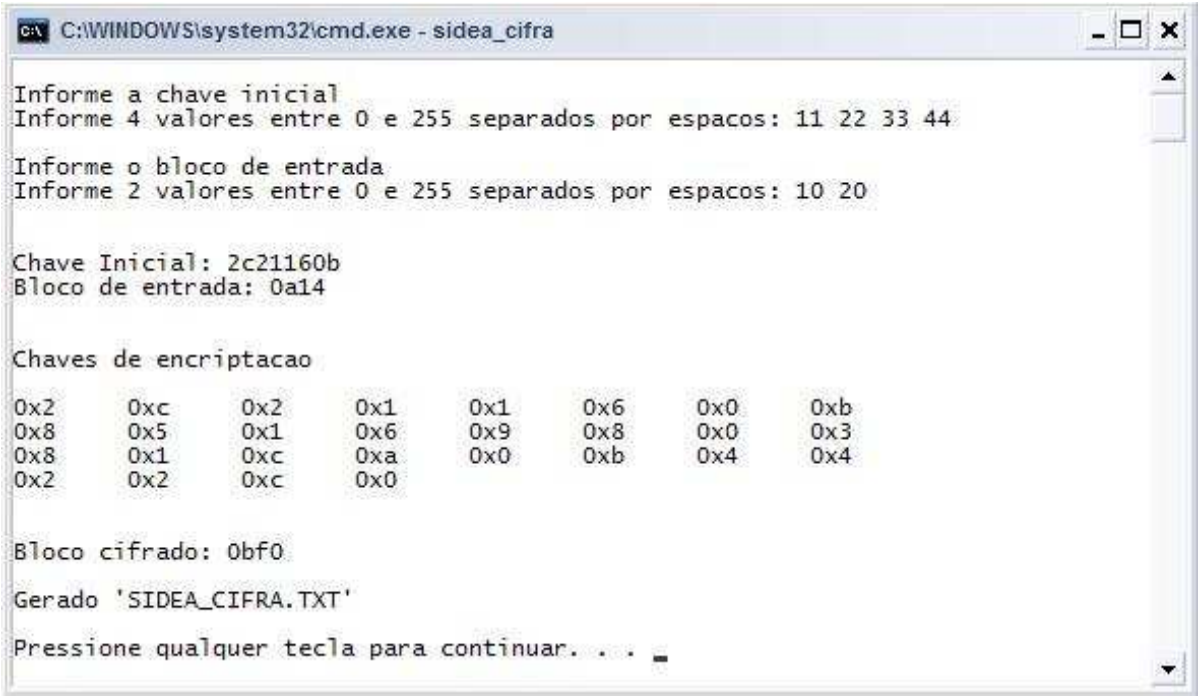


Figura 25 – Tela do SIDEA_CIFRA

O arquivo gerado pelo SIDEA_CIFRA, contém caixas auto-explicativas sobre os processos de geração de chaves e de cifragem, tal como os valores das chaves utilizadas no processo de encriptação e o valor do bloco cifrado.

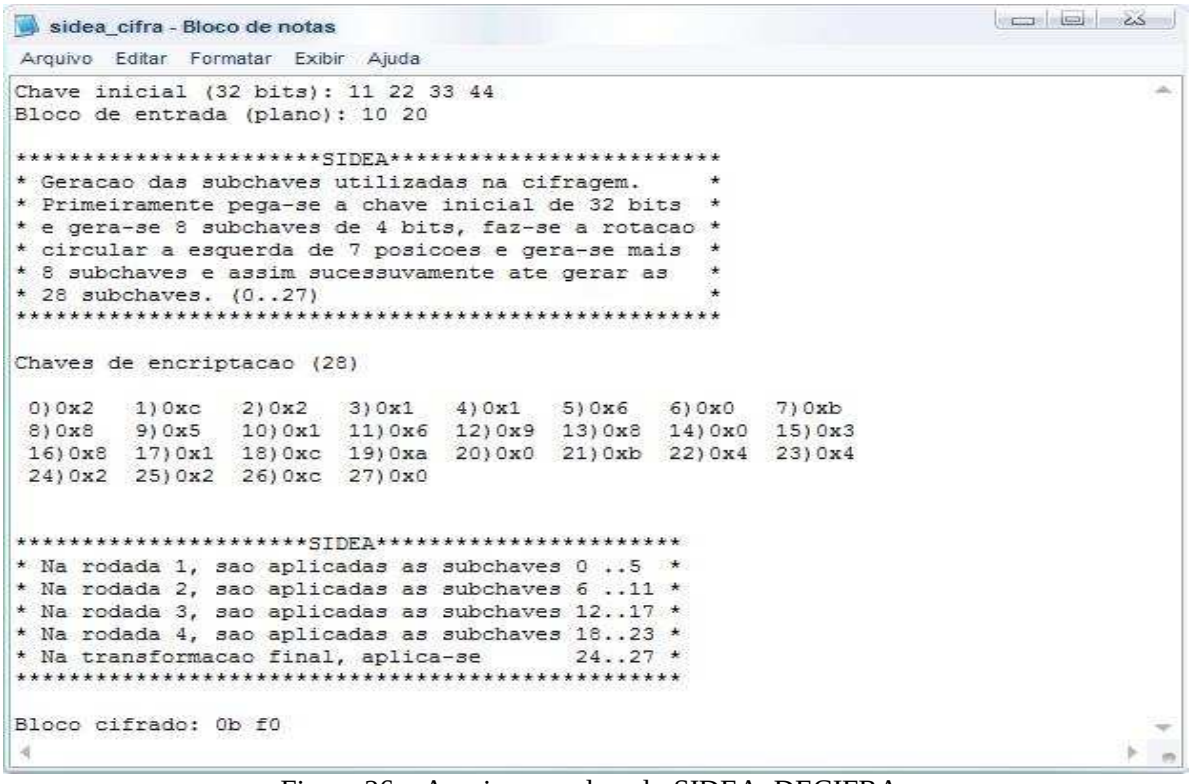


Figura 26 – Arquivo gerado pelo SIDEA_DECIFRA

A figura 26 exibe o arquivo gerado no estudo de caso. Nota-se que o bloco cifrado

possui o valor 0b 6f, que será o bloco de entrada para o SIDEA_DECIFRA.

A figura 27 apresenta as etapas do SIDEA_DECIFRA. Pode-se notar que em relação ao SIDEA_CIFRA, tem-se mais uma etapa de geração de chaves de decriptação, que são passadas como parâmetro para decifrar o bloco cifrado que foi informado pelo usuário.

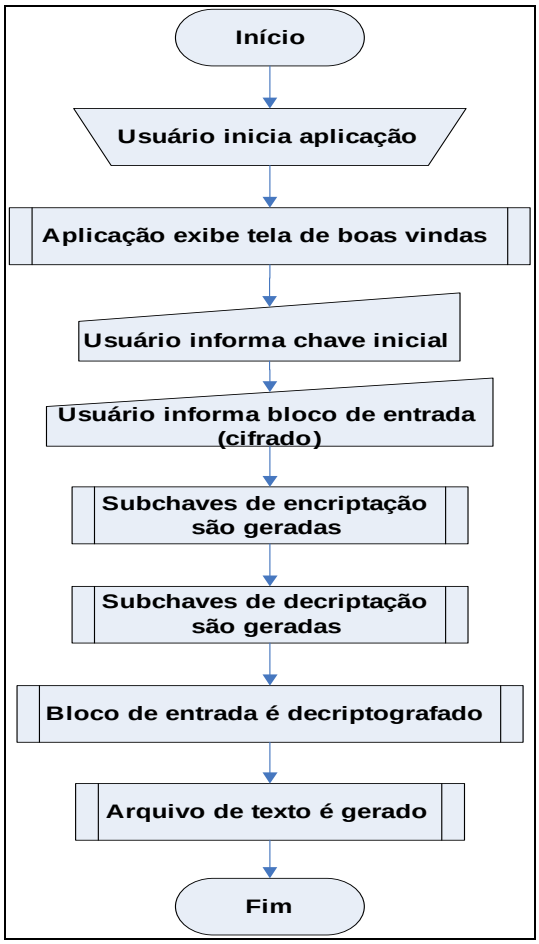


Figura 27 – Funcionamento SIDEA_DECIFRA

Ao iniciar o SIDEA_DECIFRA, a aplicação também exibe uma tela de boas vindas, porém com informações referentes ao processo de decifragem. A figura 28 exibe a tela inicial do SIDEA_DECIFRA.

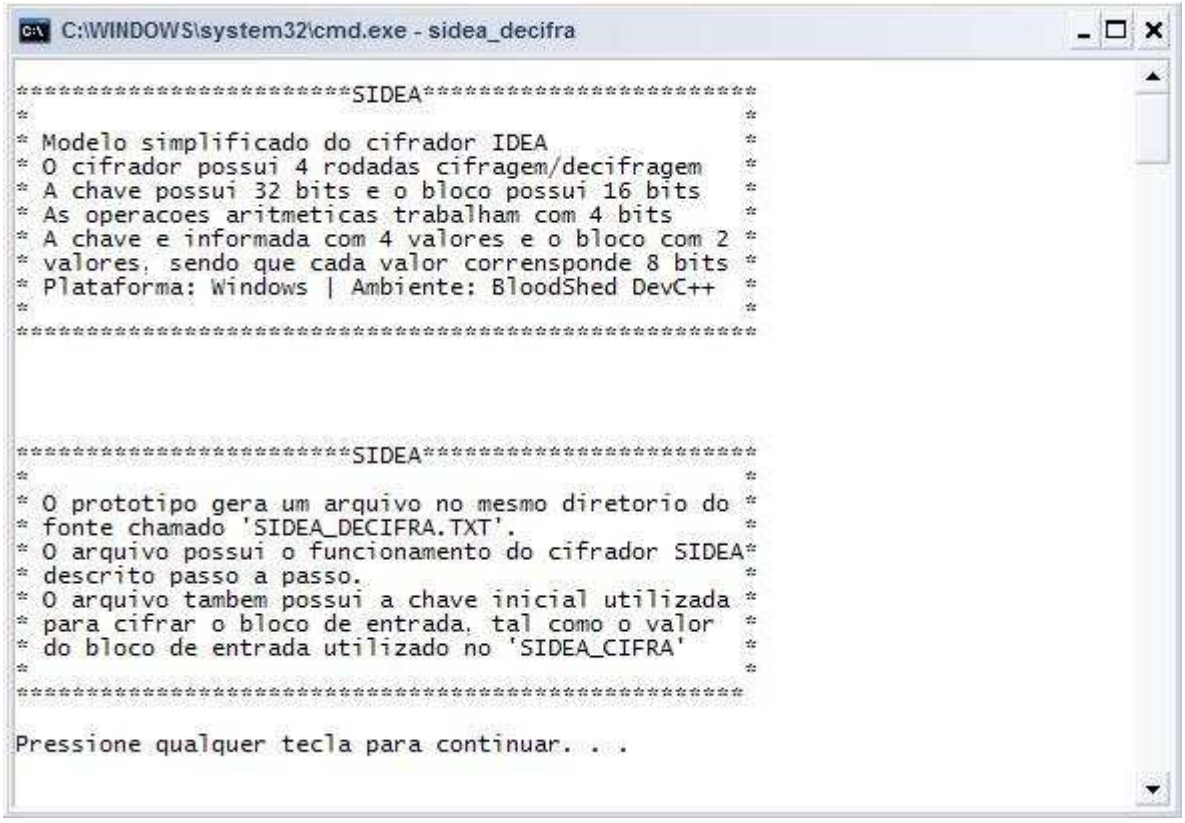


Figura 28 – Tela de boas vindas do SIDEA_DECIFRA

Na próxima tela, o usuário informa a chave inicial (11, 22, 33, 44) que foi utilizada no processo de cifragem no SIDEA_CIFRA e o bloco cifrado (0b f0) gerado pelo mesmo em notação hexadecimal. Após o usuário ter informado os valores de entrada a aplicação chama as funções *gera_chaves_cifra* que gera as chaves de encriptação e posteriormente a função *gera_chaves_decifra* que gera as chaves de decriptação a partir das chaves de encriptação. A aplicação exibe em tela o valor das chaves envolvidas no processo de decifragem e chama o processo *sidea_cifra* para decifrar o bloco cifrado que o usuário informou como entrada, mas utilizando as chaves de decriptação. Ao final a aplicação gera o arquivo texto SIDEA_DECIFRA.TXT que contém informações didáticas dos processos de decifragem. A figura 29 exibe a tela do SIDEA_DECIFRA.

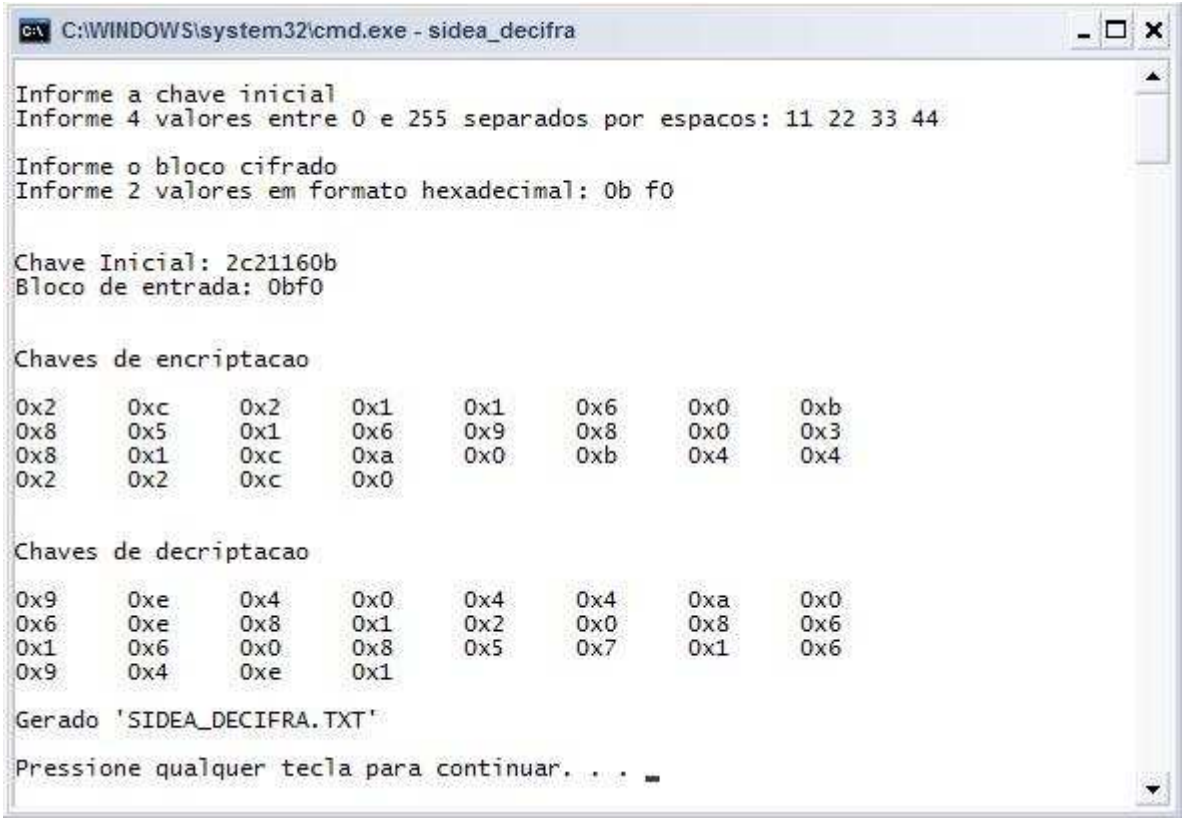


Figura 29 – Tela do SIDEA_DECIFRA

O arquivo gerado pelo SIDEA_DECIFRA, contém caixas auto-explicativas sobre os processos de geração de chaves e de decifragem, tal como os valores das chaves utilizadas no processo de decriptação e o valor do bloco plano.

Figura 30 – Arquivo gerado pelo SIDEA_DECIFRA

A figura 30 exibe o arquivo gerado no estudo de caso, pode-se notar que o valor do bloco plano (10, 20) é o mesmo que o usuário informou como bloco de entrada no SIDEA_CIFRA.

4.4 RESULTADOS E DISCUSSÃO

O objetivo de disponibilizar uma versão didática em linguagem C do SIDEA foi alcançado. A versão didática do SIDEA se encontra em anexo nos apêndices A e B.

Analisando os outros objetivos traçados, observa-se que os mesmos foram atingidos, pois o SIDEA:

a) utiliza as mesmas operações primitivas do IDEA, isto é, adição, multiplicação, XOR, deslocamento de bits;
b) pode ser facilmente realizada uma implementação manual dos processos de geração de chaves, cifragem e de decifragem;
c) pode ser utilizado em ambiente acadêmico, como por exemplo cursos de criptografia ou mesmo numa disciplina de graduação ou pós-graduação. O algoritmo é simples de entender e utilizar, embora tenha todas as características principais do IDEA;
d) facilita a apresentação didática do IDEA. É possível apresentar inicialmente o modelo simplificado e, após o entendimento deste, apresentar o cifrador IDEA.
e) facilita a análise do cifrador. Rotinas de busca exaustiva de chaves, ou outras técnicas de criptoanálise tornam-se mais simples num cifrador com parâmetros reduzidos;
f) serve como referência para a proposição de outros modelos simplificados de cifradores simétricos. Partindo das estratégias utilizadas, pode-se desenvolver

modelos simplificados para outros cifradores simétricos, alcançando objetivos semelhantes aos obtidos neste trabalho.

Conforme descrito no item 4, nenhum modelo simplificado do cifrador IDEA havia sido confeccionado antes da realização deste trabalho. O modelo desenvolvido neste trabalho teve a mesma finalidade dos trabalhos correlatos mencionados anteriormente, isto é, a realização de um modelo simplificado para apresentação didática do cifrador completo e não um modelo reduzido com a finalidade de explorar falhas ou melhorias do cifrador em estudo. Outra característica encontrada no SIDEA e nos modelos simplificados (SAES, SRC6) desenvolvidos nos trabalhos correlatos, é a correlação com seus cifradores completos e a possibilidade de fazer implementações manuais dos modelos reduzidos.

5 CONCLUSÕES

Uma preocupação do projeto foi desenvolver um modelo simplificado que tivesse correlação com o cifrador em estudo. Tendo em vista este ponto, os resultados deste protótipo são expressivos, pois atende a todos os objetivos previamente formulados. Apesar de o modelo simplificado parecer grande devido a chave de 32 bits e o bloco de entrada de 16 bits, as operações internas são realizadas com nibbles (4 bits), ou seja, o protótipo se torna didático para a apresentação do IDEA e a implementação manual é viável.

O protótipo foi desenvolvido na plataforma windows, porém com apenas alguns simples ajustes como a troca do formato de documentação no código (// - /* */) e a importação da biblioteca *unix.h* ao invés da *stdlib.h*, é possível compilá-lo e executá-lo para a plataforma Linux e FreeBSD utilizando o compilador gcc. Apesar de o protótipo ter sido desenvolvido para a plataforma windows, foi utilizada a ferramenta Bloodshed Dev C++ para a confecção do mesmo que é gratuita e não foi necessário a utilização de qualquer ferramenta paga para o desenvolvimento do mesmo.

Uma dificuldade encontrada para o desenvolvimento do SIDEA foi a obtenção do código do IDEA. Foram necessários alguns contatos via e-mail delatando as intenções da utilização do código com a empresa detentora da patente do cifrador (Agency Mediacrypt) para a obtenção do código. Outra dificuldade encontrada foi o entendimento do código fornecido pela Mediacrypt, pois o mesmo se encontrava em baixo nível, tornando a compreensão do código mais difícil.

A principal contribuição deste trabalho foi justamente o desenvolvimento do primeiro modelo simplificado do cifrador IDEA, que é um cifrador popular e muito utilizado devido a sua eficiência e também pelo seu surgimento na época da procura de um substituto do famoso DES. Este modelo simplificado pode ser adicionado ao conteúdo de disciplinas que estudam a

criptografia, pois normalmente é fornecido o estudo do S-DES e do DES que surgiu em meados da década de 70 e posteriormente comenta-se sobre cifradores mais recentes como o 3DES, IDEA e AES, porém o funcionamento não é detalhado devido a complexidade destes cifradores, sendo que após a realização deste trabalho, o SIDEA ameniza a complexidade de ensino do cifrador IDEA. O SIDEA, por ser um modelo simplificado inédito do cifrador IDEA, pode ser incluído ao acervo bibliográfico de criptólogos juntamente com modelos simplificados de outros cifradores (DES, RC6, AES) que já são alvo de estudos dos mesmos.

5.1 EXTENSÕES

Conforme já mencionado, até a realização deste trabalho não havia nenhum outro modelo simplificado do cifrador IDEA. A finalidade do modelo simplificado produzido neste trabalho é a apresentação didática do cifrador IDEA. Para extensões poderiam se fazer outros modelos simplificados do cifrador em estudo visando outras finalidades como: exploração de falhas, exploração de pontos de melhoria de desempenho e exploração de pontos de melhoria visando a segurança do IDEA.

Alguns trabalhos futuros podem ser realizados utilizando o modelo simplificado desenvolvido neste trabalho. Abaixo segue possíveis pontos a serem trabalhados:

a) adicionar uma rotina que faça a chamada do SIDEA para arquivos com tamanhos superiores a 2 bytes (16 bits), isto é, adicionar uma rotina que passa por parâmetros um arquivo de tamanho qualquer como bloco de entrada, assim o protótipo divide o tamanho deste arquivo em blocos de 16 bits para a cifragem;
b) transformar o protótipo em uma biblioteca em linguagem C, para a implementação de uma protótipo em *sockets* com comunicação segura cliente/servidor, aplicando-se, por exemplo, em uma conexão remota semelhante ao Telnet.

REFERÊNCIAS BIBLIOGRÁFICAS

BERNSTEIN, Terry. et al. **Segurança na internet**. São Paulo: Campus, 1997.

CHAVES, Ricardo. **Processamento de sinal e criptografia baseados em sistemas de numeração por resíduos.** 2001. 81 f. Monografia (Licenciatura em Engenharia Electrotécnica e de Computadores) – Departamento de Sistemas Eletrônicos e de Computadores, Instituto Superior Técnico, Lisboa.

COUNTERPANE. **Counterpane internet security:** Twofish: a new block cipher**.** Estados Unidos, 1998. Disponível em: <http//:www.counterpane.com/twofish.html>. Acesso em: 30 jun. 2005.

CUSTÓDIO, Ricardo Felipe. **Segurança em computação.** Florianópolis, 2003. Disponível em: < http://www.das.ufsc.br/~werner/nivelamento/apostila_html/apostila.pdf&e=10401>. Acesso em: 20 ago. 2005.

FREITAS, José Luiz; OLIVEIRA, Renan Rodrigues. **Implementação, comparação e análise dos algoritmos IDEA e DES.** Goiânia, 2006. Disponível em: <http://wsmartins.net/ermacs/poster_39.pdf>. Acesso em: 20 ago. 2005.

HOFFMAN, Nick. **Cryptology: t**he international data encryption cipher. Estados Unidos, 2005. Disponível em: < http://www.nku.edu/~mcsc/mat494/uploads/IDEA.pdf>. Acesso em: 30 jun. 2005.

MEDIACRYPT, Agency. **International data encryption algorithm home page**. Suíca, 2005. Disponível em: <http://www.mediacrypt.com>. Acesso em: 20 ago. 2005.

MENDES, Marco André Lopes. **Modelo simplificado do cifrador RC6**. 2001. 147 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis.

MIERS, Charles Christian. **Modelo simplificado do cifrador AES**. 2002. 129 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis.

NIST. **National Institute of Standards and Technology home page.** Estados Unidos**,** 2001**.** Disponível em: <http://www.nist.gov>. Acesso em: 20 ago. 2005.

PUTTINI, Ricardo Staciarini. **Criptografia**. Brasília, 2004. Disponível em <http://www.redes.unb.br/security/criptografia.pdf>. Acesso em: 20 ago. 2005.

RAMOS, Karla Darlene Nepomucemo. **Proposta de um algoritmo de criptografia baseado no algoritmo de Viterbi e codificação convolucional**. 2002. 85 f. Dissertação (Mestrado em Sistemas e Computação) – Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal.

SCHNEIER, Bruce. **Applied Cryptography:** protocols, algorithms and source code in C. 2sd ed. New York: John Wiley & Sons, 1996.

STALLINGS, William. **Criptography and network security:** principles and practice. 3th ed. New Jersey: Pearson Education, 2003.

VISIO. **Microsoft OfficeVisio Professional 2003.** Version 11.3216.5606. Estados Unidos, 2003. Disponível em: <http://office.microsoft.com/visio>. Acesso em: 30 jun. 2005.

WIKIPEDIA. **Wikimedia fundation home page.** Estados Unidos, 2000. Disponível em: <http://pt.wikipedia.org/wiki/Algoritmo_de_Euclides>. Acesso em: 15 maio. 2006.

APÊNDICE A – Implementação do SIDEA_CIFRA em linguagem C

Este anexo consiste na implementação do modelo simplificado do IDEA, o SIDEA, em linguagem de programação C com o padrão ANSI. Este programa recebe um texto aberto na forma de 2 valores inteiros entre 0 e 255, formando um bloco de entrada de 16 bits. A chave é informada em 4 valores inteiros entre 0 e 255, formando a chave inicial de 32 bits. Após estes valores serem lidos, são executadas as rotinas de geração de sub-chaves de encriptação e a cifragem. A cada processamento os valores são impressos em tela, de forma a facilitar o acompanhamento do algoritmo. Ao final é gerado o arquivo SIDEA_CIFRA.TXT, que contêm as chaves utilizadas no processo de cifragem, tal como valor do bloco final cifrado.

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

      #define SIDEA_RODADAS 4                          // Numero de rodadas
      #define SIDEA_SUBCHAVES (6 * SIDEA_RODADAS + 4) // Numero de subchaves (28)
      #define LSW4(y) ((y) & 0xf)                     // Bit menos significativo
      #define MSW4(y) ((y >> 4) & 0xf)                // bit mais significativo

      unsigned char chave_entrada[SIDEA_SUBCHAVES];   // Chave de entrada
      unsigned char encrypt_subkeys[SIDEA_SUBCHAVES]; // Chaves de encriptacao
      unsigned int key32=0;
      unsigned int i,y;
      unsigned char chave_inicial[4]; //Chave inicial 4 posicoes CHAR(4x8->32 bits)
      unsigned char bloco_informa[2]; //Bloco entrada 2 posicoes CHAR(2x8->16 bits)
      unsigned char bloco_entrada[4], bloco_saida[4];

      typedef unsigned char   uint8;                  // 8-bit word
      typedef unsigned char   uint4;                   // 4-bit word
      typedef char            int8;                    // 8-bit int

      void gera_chaves_cifra();
      void sidea_cifra(unsigned char *bloco_in, unsigned char *bloco_out, unsigned
char *enc);

      int main() {

          system( "cls" );
       printf("\n************************SIDEA************************\n");
       printf("*                                                     *\n");
       printf("* Modelo simplificado do cifrador IDEA                *\n");
       printf("* O cifrador possui 4 rodadas cifragem/decifragem     *\n");
       printf("* A chave possui 32 bits e o bloco possui 16 bits     *\n");
       printf("* As operacoes aritmeticas trabalham com 4 bits       *\n");
       printf("* A chave e informada com 4 valores e o bloco com 2 *\n");
      printf("* valores, sendo que cada valor corrensponde 8 bits *\n");
       printf("* Plataforma: Windows | Ambiente: BloodShed DevC++  *\n");
       printf("*                                                     *\n");
       printf("*******************************************************\n\n");
       printf("\n\n");
       printf("\n************************SIDEA************************\n");
       printf("*                                                     *\n");
```

```
       printf("* O prototipo gera um arquivo no mesmo diretorio do *\n");
       printf("* fonte chamado 'SIDEA_CIFRA.TXT'.                  *\n");
       printf("* O arquivo possui o funcionamento do cifrador SIDEA*\n");
       printf("* descrito passo a passo.                           *\n");
       printf("* O arquivo tambem possui a chave inicial utilizada *\n");
       printf("* para cifrar o bloco de entrada, tal como o valor  *\n");
       printf("* do bloco cifrado para utilizar no 'SIDEA_DECIFRA' *\n");
       printf("*                                                   *\n");
       printf("*****************************************************\n\n");
       system( "pause" );
       system( "cls" );
       printf("\nInforme a chave inicial\n");
       printf("Informe 4 valores entre 0 e 255 separados por espacos: ");
       scanf("%u %u %u %u", &chave_inicial[0], &chave_inicial[1], &chave_inicial[2],
&chave_inicial[3]);

       // Transformacao de bytes (8-bits) em nibbles (4-bits)
       chave_entrada[1] = (unsigned char) (chave_inicial[3] <<4) >> 4;
       chave_entrada[0] = (unsigned char) (chave_inicial[3] >>4);
       chave_entrada[3] = (unsigned char) (chave_inicial[2] <<4) >> 4;
       chave_entrada[2] = (unsigned char) (chave_inicial[2] >>4);
       chave_entrada[5] = (unsigned char) (chave_inicial[1] <<4) >> 4;
       chave_entrada[4] = (unsigned char) (chave_inicial[1] >>4);
       chave_entrada[7] = (unsigned char) (chave_inicial[0] <<4) >> 4;
       chave_entrada[6] = (unsigned char) (chave_inicial[0] >>4);

       printf("\nInforme o bloco de entrada\n");
       printf("Informe 2 valores entre 0 e 255 separados por espacos: ");
       scanf("%u %u",&bloco_informa[0],&bloco_informa[1]);

       // Transformacao de bytes (8-bits) em nibbles (4-bits)
       bloco_entrada[0] = (unsigned char) (bloco_informa[0] >>4);
       bloco_entrada[1] = (unsigned char) (bloco_informa[0] <<4) >> 4;
       bloco_entrada[2] = (unsigned char) (bloco_informa[1] >>4);
       bloco_entrada[3] = (unsigned char) (bloco_informa[1] <<4) >> 4;

       printf("\n\nChave Inicial: ");
       for (i=0;i<8;i++){
              printf("%x",chave_entrada[i]);}
              printf("\n");

       printf("Bloco de entrada: ");
       for (i=0;i<4;i++){
              printf("%x",bloco_entrada[i]);}
              printf("\n\n");

       //chamada de funcoes
       gera_chaves_cifra();
       sidea_cifra(bloco_entrada, bloco_saida, chave_entrada);
       }

       void gera_chaves_cifra(){
       // Divide a chave inicial de 32 bits em 8 blocos de 4 bits
       for (i=0;i<8;i++)
              key32 = ((chave_entrada[i] & 0x0f) << (4*i)) | key32;

       for (i;i<SIDEA_SUBCHAVES;i++) {
              // Deslocamento a esquerda de 7 posicoes
              key32 = i % 8 ? key32 : (key32 << 7) | (key32  >> 25);
              chave_entrada[i] = (key32 << (7-(i%8))*4) >> 28;
          }
       printf("\nChaves de encriptacao\n");
       for (i=0;i<SIDEA_SUBCHAVES;i++) {
           i % 8 ? printf("\t") : printf("\n");
           printf("0x%x",chave_entrada[i]);
       }
       printf("\n");
       }
```

```
        // Multiplicacao de modulo 2^4 + 1, utilizado em SIDEA_CIFRA
        static uint4 mul(uint4 x, uint4 y){

          uint4 t4;
          uint8 t8;

          x =  LSW4(x - 1);
          t4 = LSW4(y - 1);
          t8 = (uint8) x * t4 + x + t4 + 1;
          x = LSW4( t8 );
          t4 = MSW4( t8 );
          x = LSW4((x - t4) + (x <=t4));
          return x;
        } // Fim da funcao de multiplicacao

        // Algoritmo cifragem
        // Implementa as operacoes de adicao, multiplicacao e XOR
        void sidea_cifra(unsigned char *bloco_in, unsigned char *bloco_out, unsigned
char *chave){

        uint4 *pin = (uint4*)bloco_in;
        uint4 *pout = (uint4*)bloco_out;
        uint4 *pk = (uint4*)chave;
        uint4 word1, word2, word3, word4;
        uint4 t1, t2;
        int i;

        word1 = *pin++;
        word2 = *pin++;
        word3 = *pin++;
        word4 = *pin;

        for (i=SIDEA_RODADAS;i>0;i--) {
              word1 = mul(word1,*pk++);
              word2 = LSW4(word2 + *pk++);
              word3 = LSW4(word3 + *pk++);
              word4 = mul(word4,*pk++);

              t2 = LSW4(word1 ^ word3);
              t2 = mul(t2,*pk++);
              t1 = LSW4(t2 + (word2 ^ word4));
              t1 = mul(t1,*pk++);
              t2 = LSW4(t1 + t2);

              word1 ^= t1;
              word4 ^= t2;

              t2 ^= word2;
              word2 = LSW4(word3 ^ t1);
              word3 = t2;
        }

        word1 = mul(word1,*pk++);
        *pout++ = word1;
        *pout++ = LSW4(word3 + *pk++);
        *pout++ = LSW4(word2 + *pk++);
        word4 = mul(word4,*pk);
        *pout = word4;

        printf("\n\nBloco cifrado: ");
        for (i=0;i<4;i++){
              printf("%x",bloco_out[i]);}
              printf("\n\n");

        //gera o arquivo que contem a chave inicial e o bloco cifrado
        FILE *sidea_cifra;
        sidea_cifra = fopen( "sidea_cifra.txt", "w" );
        fprintf(sidea_cifra, "Chave inicial (32 bits):");
```

```
for (i=0;i<4;i++){
      fprintf(sidea_cifra, " %i" ,chave_inicial[i]);
}
fprintf(sidea_cifra, "\nBloco de entrada (plano):");
for (i=0;i<2;i++){
    fprintf(sidea_cifra, " %i", bloco_informa[i]);
    }
fprintf(sidea_cifra, "\n\n**********************SIDEA*****************\n");
fprintf(sidea_cifra, "* Geracao subchaves utilizadas na cifragem.      *\n");
fprintf(sidea_cifra, "* Primeiramente pega-se a chave inicial 32 bits  *\n");
fprintf(sidea_cifra, "* e gera 8 subchaves de 4 bits, faz-se a rotacao *\n");
fprintf(sidea_cifra, "* circular a esquerda 7 posicoes e gera-se mais  *\n");
fprintf(sidea_cifra, "* 8 subchaves e im sucessivamente ate gerar as   *\n");
fprintf(sidea_cifra, "* 28 subchaves. (0..27)                          *\n");
fprintf(sidea_cifra, "************************************************\n\n");
fprintf(sidea_cifra, "Chaves de encriptacao (28)\n");

    for (i=0;i<SIDEA_SUBCHAVES;i++) {
    i % 8 ? fprintf(sidea_cifra, "\t") : fprintf(sidea_cifra, "\n");
    fprintf(sidea_cifra, " %i)""0x%x",i, chave_entrada[i]);
    }
fprintf(sidea_cifra, "\n\n\n**********************SIDEA****************\n");
fprintf(sidea_cifra, "* Na rodada 1, sao aplicadas as subchaves 0 ..5  *\n");
fprintf(sidea_cifra, "* Na rodada 2, sao aplicadas as subchaves 6 ..11 *\n");
fprintf(sidea_cifra, "* Na rodada 3, sao aplicadas as subchaves 12..17 *\n");
fprintf(sidea_cifra, "* Na rodada 4, sao aplicadas as subchaves 18..23 *\n");
fprintf(sidea_cifra, "* Na transformacao final, aplica-se       24..27 *\n");
fprintf(sidea_cifra, "************************************************\n\n");
fprintf(sidea_cifra, "Bloco cifrado: ");

for (i=0;i<2;i++){
      fprintf(sidea_cifra, "%x" ,bloco_out[i]);
}fprintf(sidea_cifra, " ");
for (i;i<4;i++){
      fprintf(sidea_cifra, "%x" ,bloco_out[i]);
}
fclose( sidea_cifra );
printf("Gerado 'SIDEA_CIFRA.TXT'\n\n");
system("pause");
} //Fim do algoritmo cifragem
```

Quadro 7 – Implementação do SIDEA_CIFRA em linguagem C

APÊNDICE B – Implementação do SIDEA_DECIFRA em linguagem C

Este anexo consiste na implementação do modelo simplificado do IDEA, o SIDEA, em linguagem de programação C com o padrão ANSI. Este programa recebe um texto cifrado pela aplicação SIDEA_CIFRA na forma de 2 valores em formato hexadecimal, formando um bloco de entrada de 16 bits. A chave inicial informada, é a mesma utilizada no SIDEA_CIFRA também em 4 valores inteiros entre 0 e 255, formando a chave inicial de 32 bits. Após estes valores serem lidos, são executadas as rotinas de geração de sub-chaves de encriptação, geração de sub-chaves de decriptação e a decifragem. A cada processamento os valores são impressos em tela, de forma a facilitar o acompanhamento do algoritmo. Ao final é gerado o arquivo SIDEA_DECIFRA.TXT, que contêm as chaves utilizadas no processo de decifragem, tal como valor do bloco final que é o mesmo informado como bloco de entrada no SIDEA_CIFRA.

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

      #define SIDEA_RODADAS 4                         // Numero de rodadas
      #define SIDEA_SUBCHAVES (6 * SIDEA_RODADAS + 4) // Numero de subchaves (28)
      #define LSW4(y) ((y) & 0xf)                     // Bit menos significativo
      #define MSW4(y) ((y >> 4) & 0xf)                // bit mais significativo
      #define MUL_MOD (uint8) (((uint8)1 << 4) | 1)   // 2^4 + 1

      unsigned char chave_entrada[SIDEA_SUBCHAVES], chave_decifra[SIDEA_SUBCHAVES];
      unsigned char encrypt_subkeys[SIDEA_SUBCHAVES];
      unsigned char decrypt_subkeys[SIDEA_SUBCHAVES];
      unsigned int key32=0;
      unsigned int i,y;
      unsigned char chave_inicial[4]; //Chave inicial 4 posicoes CHAR(4x8->32 bits)
      // Bloco de entrada c/ 2 posicoes CHAR (2x8 -> 16 bits)
      unsigned char bloco_informa[2], bloco_imprime[2];
      unsigned char bloco_entrada[4], bloco_saida[4];

      typedef unsigned char   uint8;                  // 8-bit word
      typedef unsigned char   uint4;                  // 4-bit word
      typedef char            int8;                   // 8-bit int

      void gera_chaves_cifra();
      void gera_chaves_decifra(unsigned char *enc,unsigned char *dec);
      void sidea_cifra(unsigned char *bloco_in, unsigned char *bloco_out, unsigned
char *dec);

      int  main() {

      system( "cls" );
      printf("\n**********************SIDEA**********************\n");
      printf("*                                                *\n");
```

```
       printf("* Modelo simplificado do cifrador IDEA              *\n");
       printf("* O cifrador possui 4 rodadas cifragem/decifragem   *\n");
       printf("* A chave possui 32 bits e o bloco possui 16 bits   *\n");
       printf("* As operacoes aritmeticas trabalham com 4 bits     *\n");
       printf("* A chave e informada com 4 valores e o bloco com 2 *\n");
      printf("* valores, sendo que cada valor corrensponde 8 bits *\n");
       printf("* Plataforma: Windows | Ambiente: BloodShed DevC++  *\n");
       printf("*                                                   *\n");
       printf("*****************************************************\n\n");
       printf("\n\n");
       printf("\n************************SIDEA************************\n");
       printf("*                                                   *\n");
       printf("* O prototipo gera um arquivo no mesmo diretorio do *\n");
       printf("* fonte chamado 'SIDEA_DECIFRA.TXT'.                *\n");
       printf("* O arquivo possui o funcionamento do cifrador SIDEA*\n");
       printf("* descrito passo a passo.                           *\n");
       printf("* O arquivo tambem possui a chave inicial utilizada *\n");
       printf("* para cifrar o bloco de entrada, tal como o valor  *\n");
       printf("* do bloco de entrada utilizado no 'SIDEA_CIFRA'    *\n");
       printf("*                                                   *\n");
       printf("*****************************************************\n\n");
       system( "pause" );
      system( "cls" );

      printf("\nInforme a chave inicial\n");
       printf("Informe 4 valores entre 0 e 255 separados por espacos: ");
       scanf("%u %u %u %u", &chave_inicial[0], &chave_inicial[1], &chave_inicial[2],
&chave_inicial[3]);

       // Transformacao de bytes (8-bits) em nibbles (4-bits)
       chave_entrada[1] = (unsigned char) (chave_inicial[3] <<4) >> 4;
       chave_entrada[0] = (unsigned char) (chave_inicial[3] >>4);
       chave_entrada[3] = (unsigned char) (chave_inicial[2] <<4) >> 4;
       chave_entrada[2] = (unsigned char) (chave_inicial[2] >>4);
       chave_entrada[5] = (unsigned char) (chave_inicial[1] <<4) >> 4;
       chave_entrada[4] = (unsigned char) (chave_inicial[1] >>4);
       chave_entrada[7] = (unsigned char) (chave_inicial[0] <<4) >> 4;
       chave_entrada[6] = (unsigned char) (chave_inicial[0] >>4);

       printf("\nInforme o bloco cifrado\n");
       printf("Informe 2 valores em formato hexadecimal: ");
       scanf("%x %x",&bloco_informa[0],&bloco_informa[1]);

       // Transformacao de bytes (8-bits) em nibbles (4-bits)
       bloco_entrada[0] = (unsigned char) (bloco_informa[0] >>4);
       bloco_entrada[1] = (unsigned char) (bloco_informa[0] <<4) >> 4;
       bloco_entrada[2] = (unsigned char) (bloco_informa[1] >>4);
       bloco_entrada[3] = (unsigned char) (bloco_informa[1] <<4) >> 4;

       printf("\n\nChave Inicial: ");
       for (i=0;i<8;i++){
             printf("%x",chave_entrada[i]);}
             printf("\n");

       printf("Bloco de entrada: ");
       for (i=0;i<4;i++){
             printf("%x",bloco_entrada[i]);}
             printf("\n\n");

       //chamada de funcoes
       gera_chaves_cifra();
       gera_chaves_decifra(chave_entrada, chave_decifra);
       sidea_cifra(bloco_entrada, bloco_saida, chave_decifra);
      }

      void gera_chaves_cifra()
      {   //transformacao dos bytes em nibbles (4 bits)
```

```
      for (i=0;i<8;i++)
            // Divide a chave inicial de 32 bits em 8 blocos de 4 bits
            key32 = ((chave_entrada[i] & 0x0f) << (4*i)) | key32;

      for (i;i<SIDEA_SUBCHAVES;i++) {
            // Deslocamento a esquerda de 7 posicoes
            key32 = i % 8 ? key32 : (key32 << 7) | (key32  >> 25);
            chave_entrada[i] = (key32 << (7-(i%8))*4) >> 28;
         }
      printf("\nChaves de encriptacao\n");
      for (i=0;i<SIDEA_SUBCHAVES;i++) {
          i % 8 ? printf("\t") : printf("\n");
          printf("0x%x",chave_entrada[i]);
      }
      printf("\n");
      }

       // Multiplicacao de modulo 2^4 + 1, utilizado no SIDEA_CIFRA
       static uint4 mul(uint4 x, uint4 y){

         uint4 t4;
         uint8 t8;

         x =  LSW4(x - 1);
         t4 = LSW4(y - 1);
         t8 = (uint8) x * t4 + x + t4 + 1;
         x = LSW4( t8 );
         t4 = MSW4( t8 );
         x = LSW4((x - t4) + (x <=t4));
         return x;
      } // Fim da funcao de multiplicacao

      // Multiplicacao inversa utilizando o principio do algoritmo de Euclides
      // Utilizado no GERA_CHAVES_DECIFRA
      static uint4 mul_inv(uint4 x){
      int8 n1 = MUL_MOD;
      int8 n2 = (int8)x;
      int8 b1 = 0;
      int8 b2 = 1;
      int8 q, r, t;

      // 0 e 1 sao self-inverse, neste caso retorna x
      if (x <= 1)
            return x;
      while (1) {
            r = n1 % n2;
            q = n1 / n2;
            if (!r) {
                  if (b2 < 0) b2 += MUL_MOD;
                  return LSW4(b2);
            }
            else {
                  n1 = n2;
                  n2 = r;
                  t  = b2;
                  b2 = b1 - q * b2;
                  b1 = t;
            }
      }
      } // Fim da funcao de multiplicacao inversa

      // Gera as subchaves decriptacao atraves das subchaves de criptacao
      void gera_chaves_decifra(unsigned char *encrypt_subkeys,unsigned char
*decrypt_subkeys){

        decrypt_subkeys[0 ] = mul_inv(encrypt_subkeys[24]);
        decrypt_subkeys[1 ] =   LSW4(-encrypt_subkeys[25]);
        decrypt_subkeys[2 ] =   LSW4(-encrypt_subkeys[26]);
```

```
        decrypt_subkeys[3 ] = mul_inv(encrypt_subkeys[27]);
        decrypt_subkeys[4 ] =    LSW4(encrypt_subkeys[22]);
        decrypt_subkeys[5 ] =    LSW4(encrypt_subkeys[23]);

        decrypt_subkeys[6 ] = mul_inv(encrypt_subkeys[18]);
        decrypt_subkeys[7 ] =   LSW4(-encrypt_subkeys[20]);
        decrypt_subkeys[8 ] =   LSW4(-encrypt_subkeys[19]);
        decrypt_subkeys[9 ] = mul_inv(encrypt_subkeys[21]);
        decrypt_subkeys[10] =    LSW4(encrypt_subkeys[16]);
        decrypt_subkeys[11] =    LSW4(encrypt_subkeys[17]);

        decrypt_subkeys[12] = mul_inv(encrypt_subkeys[12]);
        decrypt_subkeys[13] =   LSW4(-encrypt_subkeys[14]);
        decrypt_subkeys[14] =   LSW4(-encrypt_subkeys[13]);
        decrypt_subkeys[15] = mul_inv(encrypt_subkeys[15]);
        decrypt_subkeys[16] =    LSW4(encrypt_subkeys[10]);
        decrypt_subkeys[17] =    LSW4(encrypt_subkeys[11]);

        decrypt_subkeys[18] = mul_inv(encrypt_subkeys[6]);
        decrypt_subkeys[19] =   LSW4(-encrypt_subkeys[8]);
        decrypt_subkeys[20] =   LSW4(-encrypt_subkeys[7]);
        decrypt_subkeys[21] = mul_inv(encrypt_subkeys[9]);
        decrypt_subkeys[22] =    LSW4(encrypt_subkeys[4]);
        decrypt_subkeys[23] =    LSW4(encrypt_subkeys[5]);

        decrypt_subkeys[24] = mul_inv(encrypt_subkeys[0]);
        decrypt_subkeys[25] =   LSW4(-encrypt_subkeys[1]);
        decrypt_subkeys[26] =   LSW4(-encrypt_subkeys[2]);
        decrypt_subkeys[27] = mul_inv(encrypt_subkeys[3]);

        printf("\n\nChaves de decriptacao\n");
        for (i=0;i<SIDEA_SUBCHAVES;i++) {
          i % 8 ? printf("\t") : printf("\n");
          printf("0x%x",decrypt_subkeys[i]);
      }
      } // Fim da funcao de geracao das subchaves de decriptacao

      // Algoritmo decifragem
      // Implementa as operacoes de adicao, multiplicacao e XOR
      void sidea_cifra(unsigned char *bloco_in, unsigned char *bloco_out, unsigned
char *chave){

       uint4 *pin = (uint4*)bloco_in;
       uint4 *pout = (uint4*)bloco_out;
       uint4 *pk = (uint4*)chave;
       uint4 word1, word2, word3, word4;
       uint4 t1, t2;
       int i;

       word1 = *pin++;
       word2 = *pin++;
       word3 = *pin++;
       word4 = *pin;

       for (i=SIDEA_RODADAS;i>0;i--) {
             word1 = mul(word1,*pk++);
             word2 = LSW4(word2 + *pk++);
             word3 = LSW4(word3 + *pk++);
             word4 = mul(word4,*pk++);

             t2 = LSW4(word1 ^ word3);
             t2 = mul(t2,*pk++);
             t1 = LSW4(t2 + (word2 ^ word4));
             t1 = mul(t1,*pk++);
             t2 = LSW4(t1 + t2);

             word1 ^= t1;
             word4 ^= t2;
```

```
        t2 ^= word2;
        word2 = LSW4(word3 ^ t1);
        word3 = t2;
    }

    word1 = mul(word1,*pk++);
    *pout++ = word1;
    *pout++ = LSW4(word3 + *pk++);
    *pout++ = LSW4(word2 + *pk++);
    word4 = mul(word4,*pk);
    *pout = word4;

    //transforma bloco de hexadecimal para decimal
    bloco_imprime[0] = bloco_imprime[1] = 0;
    bloco_imprime[0] = ((bloco_out[0] <<4) | (bloco_out[1]));
    bloco_imprime[1] = ((bloco_out[2] <<4) | (bloco_out[3]));

    //gera o arquivo que contem a chave inicial e o bloco de entrada
    FILE *sidea_decifra;
    sidea_decifra = fopen( "sidea_decifra.txt", "w" );
    fprintf(sidea_decifra, "Chave inicial (32 bits):");
    for (i=0;i<4;i++){
        fprintf(sidea_decifra, " %i" ,chave_inicial[i]);
    }
    fprintf(sidea_decifra, "\nBloco de entrada (cifrado):");
    for (i=0;i<2;i++){
        fprintf(sidea_decifra, " %x", bloco_informa[i]);
        }
    fprintf(sidea_decifra, "\n\n\n**********************SIDEA***************\n");
    fprintf(sidea_decifra, "* Geracao subchaves utilizadas na decifragem   *\n");
    fprintf(sidea_decifra, "* As subchaves sao geradas a partir subchaves  *\n");
    fprintf(sidea_decifra, "* utilizadas cifragem. Para a geracao destas   *\n");
    fprintf(sidea_decifra, "* subchaves, utiliza a multiplicacao inversa.  *\n");
    fprintf(sidea_decifra, "*********************************************\n\n");
    fprintf(sidea_decifra, "Chaves de decriptacao (28)\n");

    for (i=0;i<SIDEA_SUBCHAVES;i++) {
        i % 8 ? fprintf(sidea_decifra, "\t") : fprintf(sidea_decifra, "\n");
        fprintf(sidea_decifra, " %i)""0x%x",i, chave_decifra[i]);
        }
    fprintf(sidea_decifra, "\n\n\n**********************SIDEA***************\n");
    fprintf(sidea_decifra, "* Na rodada 1, sao aplicadas subchaves 0 ..5   *\n");
    fprintf(sidea_decifra, "* Na rodada 2, sao aplicadas subchaves 6 ..11  *\n");
    fprintf(sidea_decifra, "* Na rodada 3, sao aplicadas subchaves 12..17  *\n");
    fprintf(sidea_decifra, "* Na rodada 4, sao aplicadas subchaves 18..23  *\n");
    fprintf(sidea_decifra, "* Na transformacao final, aplica-se    24..27  *\n");
    fprintf(sidea_decifra, "*********************************************\n\n");
    fprintf(sidea_decifra, "Bloco descriptografado: ");

    for (i=0;i<2;i++){
        fprintf(sidea_decifra, " %i" ,bloco_imprime[i]);
    }
    fclose( sidea_decifra );
    printf("\n\nGerado 'SIDEA_DECIFRA.TXT'\n\n");
    system ("pause");
    } //Fim do algoritmo decifragem
```

Quadro 8 – Implementação do SIDEA_DECIFRA em linguagem C