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

Charles Christian Miers

Modelo Simplificado do Cifrador AES

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

Prof. Ricardo Felipe Custódio, Dr.
Orientador

Florianópolis, Julho de 2002

Modelo Simplificado do Cifrador AES

Charles Christian Miers

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

Prof. Ricardo Felipe Custódio, Dr.
Orientador

Prof. Fernando Ostuni Gauthier, Dr.
Coordenador do Curso

Banca Examinadora

Prof. Jeroen Antonius Maria van de Graaf, Dr.

Prof. Sérgio Peters, Dr.

Prof. Ricardo Felipe Custódio, Dr.

”Pensem o que quiserem de ti, faze aquilo que te parece justo.”
Pitágoras

Ofereço esse trabalho aos meus avós, Augusto Braatz (in memorian) e Edith Frahm Braatz (in memorian), cujo incentivo aos estudos e contínuo auto-aperfeiçoamento serviram de apoio para o desenvolvimento desse e muitos outros trabalhos que estão por vir.

Agradecimentos

Ao professor Wesley Masterson Melo de Abreu que, ao me apresentar ao prof. Custódio, abriu-me o caminho para iniciar esse trabalho.

A meu orientador, excelente professor e pessoa a quem muito admiro como grande profissional que é e com quem continuo aprendendo muito.

A minha namorada Jani Floriano pelo incentivo, ajuda, apoio e compreensão para a execução desse trabalho.

A Gilsiley Daru por todo o auxílio na parte de implementação na linguagem C, sem o qual esse trabalho não seria possível.

A ARLS ”Amigos para Sempre”, 73 do GOSC pela compreensão do meu afastamento temporário para esse auto-aperfeiçoamento.

Minha gratidão também aos mestres e amigos, incontáveis colaboradores, que me ensinaram, incentivaram e ajudaram direta e indiretamente no decorrer desta caminhada.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
BNR	Bell-Northern Research
CPGCC	Curso de Pós-Graduação em Ciência da Computação
DES	Data Encryption Standard
IBM	International Business Machines Corporation
IDEA	International Data Encryption Algorithm
FIPS	Federal Information Processing Standards
MAC	Message Authentication Code
MDC	Máximo Divisor Comum
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NSA	National Security Agency
RC5	Rivest Cipher 5
RC6	Rivest Cipher 6
SAES	Simplified Advanced Encryption Standard
SDES	Simplified Data Encryption Standard
SRC6	Simplified Ron’s Code 6
TI	Trabalho Individual

Lista de Símbolos

E	texto cifrado.(E =*CIFRADOR*$_K(P)$)
CIFRADOR	algoritmo de cifragem / decifragem utilizado.
GF	Galois Field, Corpo de Galois.
$_K$	chave de cifragem.(E =*CIFRADOR*$_K(P)$)
P	texto aberto (E =*CIFRADOR*$_K(P)$).
$\oplus$	*OU* exclusivo, *XOR*.
LS-X	Left Shift (Deslocamento à Esquerda), onde *X* representa o valor a ser deslocado.
Nb	Number of blocks, Número de blocos.
Nk	Number of keys, Número de chaves.
Nr	Number of rounds, Número de rodadas.
RS-X	Right Shift (Deslocamento à Direita), onde *X* representa o valor a ser deslocado.
$\in$	pertence, está contido.
$\overline{G}$	não (*NOT*).
$\approx$	aproximadamente.

$\lfloor$x$\rfloor$	maior inteiro menor ou igual a x. Arredondamento em direção a $-\infty$. Por exemplo $\lfloor 5,3 \rfloor = 5$.
$\vee$	para todo.
$\lvert x \rvert$	módulo de x.
XOR	*OU* exclusivo.
xxh	valor expresso em notação hexadecimal, onde xx é o valor e o 'h' indicador da notação.
$\bullet$	multiplicação dentro de um corpo finito.

Resumo

Nesta dissertação é descrito o desenvolvimento de um modelo simplificado para o ensino do cifrador AES destinado a estudantes de pós-graduação e fases finais da graduação. O meio adotado foi o estudo dos princípios matemáticos/algébricos seguido do desenvolvimento de uma versão simplificada do AES com a maior redução possível do número de parâmetros para a compreensão da estrutura básica. Foram desenvolvidas implementações em linguagem C do cifrador AES e SAES para facilitar a execução de testes de mesa. O objetivo é de que uma vez explicada a versão simplificada, facilitar a explicação tanto dos mecanismos que provêem forte resistência, quanto do algoritmo completo do AES.

Abstract

A Simplified Version of AES Encryption Algorithm

In this paper we describe the development of a simplified model of the AES cipher for the students of after-graduation and final phases of the undergraduate course. The adopted way was the study of algebraic and mathematical principles followed of the development of a simplified version of the AES with the maximum possible reduction of the number of parameters for the understanding of the basic structure. Implementations in C language of cipher AES and SAES had been developed to simplify the execution of classroom tests. The objective is once time the simplified algorithm has been explained, it is easier to explain not only the mechanisms that provide strong resistance, but also to explain the real one.

Capítulo 1

Introdução

Este documento consiste de uma dissertação de mestrado requisito para obtenção do título de mestrado em Ciência da Computação. Nas linhas de pesquisa abordadas pelo CPGCC[1] tem-se a linha de segurança e comércio eletrônico. Dentro dessas linhas uma das tecnologias mais essenciais, eficiente e amplamente empregada para conseguir-se segurança é o uso de criptografia.

A palavra ”criptografia”é composta dos termos gregos ”kryptos”(secreto, oculto, não inteligível) e ”grapho”(escrita, escrever). Consiste na ciência e na arte de comunicar-se secretamente. O objetivo principal é tornar a mensagem indecifrável para terceiros, que possam a vir a interceptar a mensagem/comunicação. A criptografia é tão antiga quanto a escrita, porém, somente nas últimas décadas tornou-se alvo de estudos científicos mais extensos e aprofundados.

Com o advento da era da informática os métodos tradicionais de criptografia manuais (não computadorizados) tornaram-se frágeis, devido ao poder computacional dos novos equipamentos. Surgiu, então, uma nova geração de métodos de cifragem (simétrico e assimétrico) e de algoritmos cifradores. Devido ao aparecimento de vários métodos de cifragem diferentes as empresas, instituições de pesquisa e governos necessitavam comprovar cientificamente a eficiência e eficácia do algoritmo criptográfico que utilizavam através da pesquisa em meios de quebrar[2] a segurança desses métodos de

[1] Curso de Pós-Graduação em Ciência da Computação

[2] técnicas de criptoanálise

cifragem. Contudo a comunidade cívil, indústria, comércio e atividades bancárias ao redor do planeta necessitavam de uma padronização para prosperar nesta nova realidade, visto que a forma de operação e funcionamento dos protocolos criptográficos eram diferentes em cada método de cifragem o que os tornava incompatíveis.

O uso de criptografia assimétrica para comunicações exige um poder computacional maior em ambos os lados da comunicação devido ao fato das chaves serem maiores que as usadas na criptografia simétrica[3], consumindo mais recursos computacionais e deixando desta forma as comunicações mais demoradas. Os métodos de criptografia assimétrica em geral são mais demorados que os simétricos devido justamente ao tamanho da chave utilizada. Isso fez que o emprego de algoritmos assimétricos, na maioria dos casos, ficasse restrito a troca de chaves de sessão simétricas e a assinatura digital.

O primeiro cifrador a tornar-se padrão no mundo foi uma variante do Lucifer da IBM[4], que tornou-se então conhecido como DES[5] em 1977, com o tamanho padrão de chave com 56 bits. O DES foi sendo re-certificado pelo NIST a cada cinco anos, na última renovação, em 1993, ficou estabelecido 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. Já em 1993 - Michael Wiener [WIE 93], pesquisador do BNR[6] canadense, mostrou com uma riqueza de detalhes como construir uma máquina dedicada, com arquitetura paralela, para quebrar chaves DES. O componente chave era um ”chip”capaz de testar 50 milhões de chaves por segundo e que custava, a preços da época, de apenas US$ 10 cada chip. Assim, com US$ 1 milhão era perfeitamente possível construir um equipamento de arquitetura paralela baseada nesses chips, que descobriria a chave DES empregada para cifrar uma mensagem em apenas três horas e trinta minutos (tempo médio). Comprovou-se então a necessidade de novos cifradores com chaves

[3]Na criptografia simétrica normalmente são empregadas chaves de 64/128/192/256 bits enquanto na assimétrica chaves de 1024/2048/4096 bits

[4]a diferença entre o DES e Lucifer está no tamanho de chave que no Lucifer é de 128 bits

[5]Data Encryption Standard

[6]Bell-Northern Research

maiores e mecanismos de cifrar nos algoritmos que dificultassem mais os métodos de criptoanálise atuais e futuros.

Em 1997 o NIST iniciou o processo de escolha do sucessor do DES, o Advanced Encryption Standard (AES). O NIST especificou que os candidatos deveriam operar com tamanho de chave e bloco variável com tamanho mínimo de 128 bits, desta forma ficaram excluídos o DES/3DES e algoritmos como o IDEA[7], o CAST[8], o Blowfish e outros que trabalham com tamanho de chave fixo e blocos de 64 bits.

O NIST colocou como requisitos fundamentais:

- Segurança Forte. O algoritmo projetado deve suportar os ataques futuros;
- Projeto Simples. De modo que possa ser com sucesso criptoanalisado;
- Desempenho. Razoavelmente bom em uma variedade de plataformas, variando de smartcards a servidores;
- Não serem patenteados. Os algoritmos devem ser de domínio público e devem estar disponíveis mundialmente.

No primeiro congresso, chamado também de rodada, o NIST recebeu vinte e uma submissões, das quais quinze atendiam as exigências. Estes algoritmos foram testados e avaliados pela comunidade científica e empresas ligadas ao ramo de segurança durante o intervalo de tempo até a segunda rodada.

Para o segundo congresso foram escolhidos apenas cinco algoritmos: MARS, RC6, Rijndael, Serpent e TwoFish os quais satisfaziam as principais condições, entre outras, de:

- cifrar e decifrar blocos de 128 bits;
- trabalhar com chaves de 128/192/256 bits;
- ser mais rápido que o 3DES.

[7] International Data Encryption Standard
[8] Carlisle Adams and Stafford Tavares

Como todos os requisitos básicos foram supridos pelos concorrentes, a decisão foi tomada com base em certas características, tais como: segurança (criptoanálise); eficiência em hardware e software; flexibilidade de implementação e modos de operação.

O algoritmo vencedor foi o Rijndael, sendo a decisão informada oficialmente em no dia 2 de outubro de 2000, desse momento em diante o cifrador Rijndael passou a ser chamado de AES[9]. A escolha do cifrador Rijndael contra os outros quatro finalistas do processo, os quais também foram classificados como altamente seguros pelo NIST, foi baseada na eficiência e baixa requisição de recursos. Os demais cifradores finalistas do concurso para AES poderão ter utilização em produtos específicos, da mesma forma que ocorreu com RC5[10] por exemplo, visto que o padrão é para o governo dos Estados Unidos da América e a comunidade privada poderá usar outros algoritmos para cifrar.

Para um estudo eficiente dos cifradores deve-se estudar os algoritmos para cifrar, decifrar e geração de chaves. Desta forma pode-se analisar os principais pontos de cada etapa e compreender melhor o conceito do funcionamento como um todo e operações em específico.

Baseado nesses fatos elegeu-se o AES como tema de pesquisa. O desenvolvimento de um modelo simplificado (SAES - Simplified Advanced Encryption Standard), do cifrador AES, de modo a simplificar o ensino em cursos de graduação e pós-graduação com disciplinas de criptografia e segurança, da mesma forma que ocorreu com o SDES para o cifrador DES[11]. O estudo do SDES, no Trabalho Individual [MIE 02] que precedeu essa dissertação, teve por finalidade mostrar o poder de um modelo simplificado no aprendizado e também o de fornecer noções básicas de como deve ser um modelo simplificado. Precedendo o cifrador AES foram incluídos tópicos de álgebra e funções matemáticas necessárias a compreensão do cifrador.

[9] De modo idêntico ao Lucifer que quando foi escolhido como padrão passou a chamar-se DES

[10] Rivest Cipher 5

[11] Conforme fora estudado no Trabalho Individual [MIE 02]

1.1 Objetivo Geral

Desenvolver um modelo simplificado do cifrador AES para fins didáticos, baseado em uma pesquisa bibliográfica.

1.2 Objetivos Específicos

De modo a atingir o Objetivo Geral estabelecido, determinados ítens devem ser estudados em específico os quais serão classificados como objetivos primários deste trabalho. Os objetivos primários que devem ser atingidos com este trabalho são:

- Estudar o cifrador AES;
- Descrever as ferramentas utilizadas no projeto do AES;
- Propor modificações no AES sem descaracterizar sua arquitetura básica;
- Propor um modelo simplificado do cifrador AES;
- Implementar em linguagem C versões didáticas do AES e SAES.

1.3 Trabalhos Correlacionados

O ensino de cifradores complexos através de modelos simplificados teve como trabalho inicial de referência o modelo simplificado do DES (SDES), desenvolvido pelo professor Edward Schaefer da Universidade de Santa Clara, Califórnia/EUA. O trabalho teve difusão mundial com a publicação no *Journal of Cryptologia* [SCH 96] e sua inclusão por Willian Stallings no livro *Criptography And Network Security: Principles And Practice* [STA 98].

Recentemente a UFSC[12], através do CPGCC[13], começou a dar ênfase no desenvolvimento de modelos simplificados dos principais cifradores de bloco simétricos

[12]Universidade Federal de Santa Catarina

[13]Curso de Pós-Graduação em Ciência Da computação

da atualidade através de trabalhos de dissertação de mestrado. Essas dissertações tem por finalidade o desenvolvimento de uma metodologia de ensino dos cifradores simétricos através do uso e correlação de modelos simplificados dos cifradores. Dentre os trabalhos que vem sendo realizados pode-se citar a dissertação de Marco André Lopes Mendes entitulada de: SRC6-Modelo Simplificado do Cifrador Simétrico RC6 [MEN 01].

Relativo ao AES até o momento não há referências na literatura especializada de criptografia de nenhum modelo simplificado com fins didáticos. Foram encontradas, apenas, algumas variantes reduzidas para fins de estudo de criptoanálise e que, por sua forma de concepção, não destinam-se ao ensino do cifrador AES.

1.4 Metodologia Empregada

O DES foi o padrão para criptografia simétrica para o governo americano nos Estados Unidos da América e isto o tornou mundialmente conhecido e motivo de estudos em âmbito mundial. Como o funcionamento do DES envolvia uma arquitetura complexa à didática, de ensino difícil, fora criado por Edward Schaefer um modelo simplificado, SDES [SCH 96], que facilitasse o aprendizado do DES. Da mesma forma que fora criado um modelo simplificado para o DES fora criado um modelo simplificado para o AES.

A metodologia empregada no desenvolvimento consistiu em um estudo bibliográfico do cifrador DES e metodologia empregada no desenvolvimento do SDES[14] para análise dos métodos empregados no seu desenvolvimento.

O AES, por empregar em sua arquitetura conceitos matemáticos/algébricos, não muito usuais em cursos de graduação/pós-graduação em Ciência da Computação. Uma revisão com base bibliográfica em matemática e álgebra foi realizada com a finalidade de elucidar conceitos. O estudo bibliográfico está baseado em artigos, documentos e técnicas de criptoanálise sobre o cifrador, desta forma estes estudos foram organizados do seguinte modo:

[14]Estudo realizado no Trabalho Individual [MIE 02]

1. Fundamentos de matemática e álgebra;
2. Operações utilizadas no AES;
3. Funcionamento do AES;
4. Simplificação das estruturas do AES;
5. Proposição de um modelo simplificado do AES.

Com base nesses estudos foi desenvolvido de um modelo simplificado do AES, que mantém as características de funcionamento original, mas com o tamanho dos parâmetros reduzidos de modo a possibilitar a sua implementação manual em sala de aula, de forma didática a compreender seu funcionamento.

Para auxiliar na tarefa de implementação e correção do SAES e AES foram desenvolvidas implementações dos cifradores em linguagem C voltadas ao auxílio da aprendizagem. A função dessas implementações não é a cifragem de arquivos e sim o fornecimento de informação detalhada do processo de geração de chaves, cifrar e decifrar.

1.5 Organização do Trabalho

No capítulo 2 é relatado o histórico e conjuntura em que o AES encontra-se inserido. Devido a essência matemática das operações e estruturas são esclarecidos conceitos e operações matemáticas e algébricas utilizadas. As operações necessárias para o processo de cifrar são explicadas isoladamente e analisadas quanto a sua estrutura, sendo aproveitadas novamente mais adiante para explicar o processo de cifrar/decifrar. Pelo fato do AES ter sua expansão de chaves gerada por um algoritmo com operações semelhantes ao processo de cifrar nenhum comentário estrutural adicional foi necessário. O processo de decifrar inclui explicações adicionais devido a sua estrutura utilizar as transformações inversas matemáticas do processo de cifrar e a possibilidade de uma estrutura alternativa.

O capítulo 3 possui a descrição das etapas e pontos principais no desenvolvimento do modelo simplificado do cifrador AES para fins didáticos. Com base nos

estudos realizados na concepção do SDES a partir do DES e com base na fundamentação matemática do cifrador, que facilitassem seu ensino em sala de aula, foram relacionados os pontos que devem ser parametrizados e simplificados no AES. Cada transformação do AES é simplificada sequencialmente de modo a demonstrar a derivação do AES para o SAES de forma racional. Foram incluídos exemplos gerados a partir da implementação didática e modelos de estruturas para facilitar o entendimento do cifrador em sua forma simplificada. Também é feita uma comparação com o objetivo de mostrar as principais diferenças entre AES e SAES.

No capítulo 4 são relatadas as considerações dos pontos relevantes dessa dissertação, assim como sugestões para trabalhos futuros. Nos Anexo A e Anexo B estão descritas as instruções para o uso das implementações didáticas dos cifradores AES e SAES, respectivamente. Também encontram-se listados os códigos fontes em linguagem C.

Capítulo 2

O Cifrador AES

2.1 Introdução

Neste capítulo serão abordados temas como o processo de seleção do Rijndael como AES, um breve histórico da origem e o estudo do cifrador. Serão estudados alguns conceitos matemáticos, componentes do cifrador, um estudo do cifrador propriamente dito (geração de chaves, processo para cifrar e decifrar) e breve revisão de artigos técnicos sobre criptoanálise do Rijndael.

2.2 Histórico

O concurso AES foi elaborado pelo NIST com o propósito de selecionar um algoritmo para cifradores de bloco simétrico para as três primeiras décadas do século 21. A chamada inicial para os algoritmos candidatos foi publicada em 12 de setembro de 1997. Quinze algoritmos foram selecionados para serem avaliados na primeira rodada em 14 de setembro de 1998. Durante o período de avaliação, o NIST solicitou comentários sobre os candidatos enfocando especificamente questões como: segurança, custo, características técnicas, conveniência para software e hardware, simplicidade, direitos de propriedade intelectual, análise comparativa, e sobretudo recomendações. Devido, em grande parte, aos comentários retornados deste trabalho os candidatos selecionados

para a segunda rodada foram: MARS [CB 98], RC6 [RR 98], Rijndael [JD 99a], Twofish [BS 98] e Serpent [RA 98].

Com o anúncio em 9 de agosto de 1999 dos cinco algoritmos classificados para a segunda rodada intensificaram-se os trabalhos de criptoanálise desses candidatos. Trabalhos de implementação em hardware[1] com comparativos entre os candidatos tiveram substancial importância na decisão e comentários da terceira rodada.

O estudo de versões reduzidas dos cifradores permitiu melhor avaliar aspectos como o grau de resistência a diversos tipos de ataques de criptoanálise, assim como, uma melhor compreensão dos mecanismos empregados e funcionamento dos cifradores candidatos. Pode-se citar alguns trabalhos, de criptoanálise, feitos sobre o cifrador Rijndael como: ”Improved Cryptanalysis of Rijndael”[NF 00], ”A Timing Attack Against Rijndael”[FK 99], ”A Collision Attack on 7 Rounds of Rijndael”[HG 00] e ”Cryptanalysis of Reduced Variants of Rijndael”[EB 00].

A terceira rodada ocorreu em abril de 2000 e teve por finalidade os últimos comentários e análises dos candidatos. Trabalhos complementares foram aceitos até 15 de maio de 2000, sendo que, o NIST fez uma ampla pesquisa entre os conferencistas participantes das rodadas do AES para saber de suas preferências e motivos pelas preferências nos candidatos. Os resultados dessas pesquisas indicavam a vontade de um único algoritmo como padrão (existia a hipótese de um ou mais algoritmos serem escolhidos como padrão) e dos algoritmos o preferido era o Rijndael.

Durante todo o processo seletivo as equipes competidoras mantiveram comunicação entre si através do site público do NIST e diretamente entre as equipes de desenvolvimento de cada algoritmo ou outros pesquisadores contribuintes do esforço de seleção. O clima de competição foi muito amigável sendo que Bruce Schneier, líder da equipe do Twofish, fez elogios públicos a equipe e ao algoritmo do Rijndael.

O resultado final do processo de seleção para o AES foi divulgado pelo

[1] pode-se citar os estudos de Fumihiko Sano, Masanobu Koike, Shinichi Kawamura e Masue Shiba em ”Performance Evaluation of AES Finalists on the High-End Smart Card”[KAW 00] e AJ. Elbirt, W. Yip, B. Chetwynd e C. Paar em ”An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists”[AE 00]

NIST em 2 de outubro de 2000 tendo o algoritmo Rijndael como vencedor. A partir deste momento o cifrador Rijndael passou a ser nomeado pelo NIST como AES. A escolha do cifrador Rijndael contra os outros quatro finalistas do processo, os quais também foram classificados como altamente seguros pelo NIST, foi baseada na eficiência e baixa requisição de recursos. O fato de existirem criptoanálises do Rijndael baseadas em variantes com menos rodadas, um pouco aproximadas do cifrador original, foi algo que contribuiu para que existissem controvérsias sobre a escolha do Rijndael como AES.

Os criadores do Rijndael são Vincent Rijmen e Joan Daemen da Universidade Católica de Leuven na Bélgica. O desenvolvimento do Rijndael foi precedido pelo cifrador de bloco Square [JD 99b], o qual possui semelhanças com o Rijndael. Quando o NIST anunciou o concurso AES em 1997, Vincent Rijmen e Joan Daemen já estavam trabalhando em uma evolução do cifrador Square. O que a equipe fez ao tomar conhecimento do concurso foi fazer as modificações necessárias para finalizar o projeto e adaptá-lo aos requisitos exigidos pelo NIST [DW 00].

O Rijndael não utiliza Estruturas de Feistel[2] devido ao fato deste tipo de operação ter um funcionamento serial. Nas Estruturas de Feistel o bloco de dados sofre operações em segmentos do bloco original seqüencialmente em cada rodada, isso quer dizer que em cada rodada somente uma parte do bloco é modificada. Sendo assim perceberam que o uso de Estruturas de Feistel possuiria uma influência direta na velocidade, visto que para cada bloco de dados ser alterado em uma estrutura de Feistel padrão duas rodadas são necessárias.

O Rijndael foi projetado de modo que fosse voltado mais ao paralelismo de processamento que suas etapas, e também o bloco de dados, não seria processado em segmentos a cada rodada mas realizaria o processamento do bloco inteiro a cada rodada utilizando-se para isto de uma operação de bloco chamada Rede S/P[3]. Através da figura 2.1 pode-se observar as diferenças da arquitetura de uma Rede S/P e uma estrutura de Feistel.

[2]um estudo mais aprofundado sobre Estruturas de Feistel foi realizado no Trabalho Individual [MIE 02]

[3]Uma rede S/P(Substituição/Permutação) é uma estrutura composta somente de permutações e rotacionamentos circulares

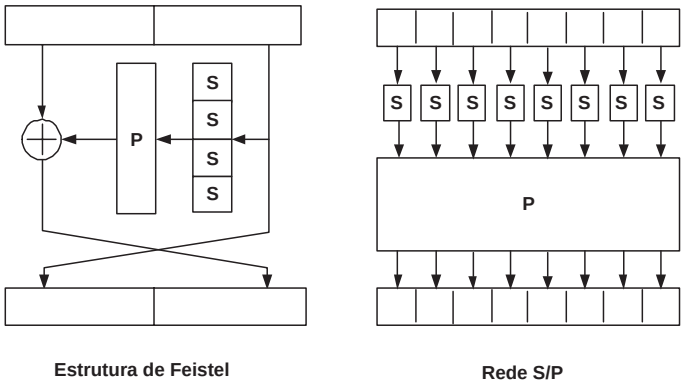

Figura 2.1: Diferenças da arquitetura de uma Rede S/P e um Estrutura de Feistel

As operações utilizadas no Rijndael envolvem cálculos matemáticos como Corpos de Galois e multiplicação modular. Tendo-se conhecimento destes assuntos pode-se perceber que o funcionamento é bastante simples e claro de entender. O uso destas operações faz o Rijndael muito flexível de modo que tanto a chave como bloco de texto aberto de entrada podem ser aumentados em incrementos de 32 bits, começando com o tamanho mínimo de 128 bits até 256 bits.

O fato do Rijndael de ser pequeno e rápido, mesmo com uma chave com o tamanho de 128 bits, permite o seu uso em aplicações aonde a segurança podia ser a comparada ao desejado pelo DES, mas aonde os recursos computacionais são muito limitados como celulares e smartcards. A previsão do início de uso em massa do AES é para o ano de 2002.

2.3 Fundamentos Matemáticos

O objetivo desta seção é fazer uma breve revisão sobre algumas operações utilizadas no cifrador. O AES utiliza funções matemáticas e álgebra, torna-se necessário revisar alguns conceitos para compreender o funcionamento do cifrador e as operações utilizadas. Maiores informações podem ser encontradas no livro *Algebra Moderna* [HHD 82] e *Cryptography with Coding Theory* [WAS 01].

2.3.1 Grupos

Pode-se definir grupos como sistemas que com uma operação, prestam-se a mais simples descrição formal. Apesar desta simplicidade de descrição, os conceitos fundamentais como homomorfismo, construção de quocientes e os análogos, que desempenham um papel importante em todas estruturas algébricas já entram na Teoria do Grupos de uma forma pura e reveladora [HER 70].

Em álgebra abstrata, existem certos sistemas básicos que são em geral conjuntos com os quais podemos operar algebricamente com seus elementos. Isto é, pode-se combinar dois elementos do conjunto, talvez de diversas maneiras, para poder obter um terceiro elemento do conjunto de modo que assume-se que estas operações algébricas estão sujeitas a certas regras que são enunciadas explicitamente no que denomina-se axiomas ou postulados que definem o sistema.

Definição 1: Um grupo *(G,•)* consiste em um conjunto *G* com uma operação binária • que satisfaça os cinco axiomas seguintes:

1. Para todo *a,b* $\in$ *G*, *a•b* $\in$ *G*. Conhecida como propriedade de fechamento.

2. A operação de grupo é associativa. Isto significa que, *a•(b•c)=(a•b)•c* para todo $a, b, c \in G$. Conhecida como propriedade associativa.

3. Existe um elemento 1 $\in$ *G*, chamado de elemento identidade, tal que *a•1=1•a=a* para todo *a* $\in$ *G*. Conhecida como propriedade identidade.

4. Para cada *a* $\in$ G exista um elemento $a^{-1} \in G$, chamado inversa de *a*, tal que *a•*a^{-1}*=*a^{-1}*•a=1*. Conhecida como propriedade inversa.

5. A operação *a•b=b•a* para todo *a,b* $\in$ *G*. Conhecida como propriedade comutativa.

Definição 2: Um grupo é dito como abeliano quando satisfaz a condição exposta no item número 5 da Definição 1, o que significa que é comutativo. A comutação indica que quando uma operação é realizada por dois elementos *a,b* $\in$ G, não importa a posição dos elementos na operação que o resultado será o mesmo.

Definição 3: Um grupo G é finito se $|G|$ é finito. O número de elementos em um grupo finito é chamado de ordem.

2.3.2 Homomorfismo

Por homomorfismo pode-se entender como uma aplicação de um sistema algébrico, em outro sistema algébrico semelhante, que conserva a estrutura.

Definição 1: Uma aplicação Φ de um grupo G em um grupo $\overline{G}$ é dita como sendo um homomorfismo se para todos $a, b \in G$, $\Phi(ab) = \Phi(a)\Phi(b)$.

Definição 2: Dois grupos G e G^* são ditos isomorfos se existe um homomorfismo bijetor (ou seja, um isomorfismo) de G em G^* [HER 70]. Neste caso escreve-se $G \approx G^*$.

Com base na definição acima pode-se verificar os três fatos seguintes:

1. $G \approx G$;

2. $G \approx G^*$ implica $G^* \approx G$;

3. $G \approx G^*$, $G^* \approx G^{**}$ implica $G \approx G^{**}$.

Quando dois grupos são isomorfos, então de certo modo, eles são iguais, diferindo-se apenas no fato de que seus elementos possuem uma notação diferente. O isomorfismo fornece uma chave da notação, e com ela, tendo conhecimento de uma dada computação em um grupo, pode-se efetuar a computação análoga no outro.

O isomorfismo pode ser definido de uma maneira simplificada como um dicionário que permite traduzir uma sentença de um idioma numa sentença, com o mesmo significado, em outro idioma. Mas simplesmente, dizer que uma dada sentença num idioma pode ser expressa em outro é pouco significativo, é necessário o dicionário para efetuar a tradução. Da mesma forma pode ter pouco significado saber que dois grupos são isomorfos, o objeto de interesse pode ser o próprio isomorfismo [ROT 94].

2.3.3 Algoritmo de Euclides

O Máximo Divisor Comum (*MDC*) de dois inteiros a e b pode ser computado via:

$$MDC(a,b) = {p_1}^{min(e1,f1)} {p_2}^{min(e2,f2)} \ldots {p_k}^{min(ek,fk)}$$

onde p_i são números primos distintos e e_i são inteiros positivos. Contudo, calcular o *MDC* através de primeiro obter-se a fatorização de potências de números primos não resulta em um método algorítmico eficiente, da mesma forma que fatorar inteiros pode tornar-se relativamente difícil. O Algoritmo de Euclides é um algoritmo eficiente para calcular o *MDC* de dois inteiros que não requer a fatorização dos inteiros.

O Algoritmo de Euclides é uma técnica utilizada para calcular o *MDC*, não apenas de pares de inteiros, mas também de pares de polinômios, ou realmente para qualquer par de elementos pertencentes a um Domínio Euclidiano.

Um Domínio Euclidiano pode ser definido como um domínio de inteiros considerado de conjunto D, desde que duas operações binárias, $+$ e $\bullet$, tal que [MCE 89]:

1. Os elementos de D formem um grupo abeliano sobre a operação $+$; o elemento identidade aditivo é denotado por 0;

2. A multiplicação é associativa e comutativa, e possui um elemento identidade, denotado por 1;

3. As leis de cancelamento estão asseguradas. Isto é, se $ab = ac$ e $a \neq 0$, então $b = c$;

4. As leis distributivas estão asseguradas. Isto é, se *a*, *b* e *c* pertencem a D, então $a(b + c) = ab + ac$.

Um Domínio Euclidiano é um domínio de inteiros com uma característica adicional de uma noção do tamanho entre seus elementos. Este tamanho de $a \neq 0$, denotado $g(a)$, é um inteiro não negativo tal que:

1. $g(a) \leq g(ab)$ se $b \neq 0$;

2. Para todo $a, b \neq 0$ deve existir q e r (quociente e resto) tal que $a = qb + r$, com $r = 0$ ou $g(r) < g(b)$.

O modo que o Algoritmo de Euclides trabalha baseia-se na repetição da identidade de *MDC(a,b) = MDC(b, a mod b)*. A especificação do algoritmo em pseudo-código [MEN 97] pode ser dada por:

ENTRADA: dois inteiros não negativos a e b com $a \geq b$.
SAÍDA: o Máximo Divisor Comum entre a e b.
1. Enquanto $b \neq 0$ faça:
1.1. Atribuir $r \leftarrow a$ mod b, $a \leftarrow b$, $b \leftarrow r$.
2. Retornar(a).

Exemplo. Passos efetuados para o cálculo de *MDC*(4864,3458)=38 através do uso do Algoritmo de Euclides [MEN 97].

$$4864 = 1 * 3458 + 1406$$
$$3458 = 2 * 1406 + 646$$
$$1406 = 2 * 646 + 114$$
$$646 = 5 * 114 + 76$$
$$114 = 1 * 76 + 38$$
$$76 = 2 * 38 + 0$$

No Algoritmo Extendido de Euclides é dado a, b e ela retorna não apenas d=*MDC*(a, b) mas os inteiros x, y tal que $d = ax + by$. É muito similar ao Algoritmo de Euclides, diferindo apenas que pode-se ter informação extra em cada passo. Esta informação adicional permite expressar o *MDC* como uma combinação linear dos dois números originais.

A especificação do algoritmo em pseudo-código [MEN 97] pode ser dada por:

ENTRADA: dois inteiros não negativos a e b com $a \geq b$.
SAÍDA: d =*MDC*(a, b)e os inteiros x,y que satisfaçam $ax + by = d$.

1. Se $b = 0$ então atribuir $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$, retorne(d, x, y).

2. Atribuir $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$.

3. Enquanto $b \neq 0$ faça:

3.1. $q \leftarrow \lfloor a/b \rfloor$, $r \leftarrow a - qb$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$.

3.2. $a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$, $y_1 \leftarrow y$.

4. Atribuir $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$ e Retornar(d, x, y).

Exemplo. Passos efetuados para as entradas $a = 4864$ e $b = 3458$ no cálculo de *MDC*(4864,3458)=38 e $(4864)(32) + (3458)(-45) = 38$ através do uso do Algoritmo Extendido de Euclides [MEN 97].

Tabela 2.1: Etapas do Algoritmo Extendido de Euclides com entradas $a = 4864$ e $b = 3458$.

q	r	x	y	a	b	x_2	x_1	y_2	y_1
-	-	-	-	4864	3458	1	0	0	1
1	1406	1	-1	3458	1406	0	1	1	-1
2	646	-2	3	1406	646	1	-2	-1	3
2	114	5	-7	646	114	-2	5	3	-7
5	76	-27	38	114	76	5	-27	-7	38
1	38	32	-45	76	38	-27	32	38	-45
2	0	-91	128	38	0	32	-91	-45	128

2.3.4 Corpos

De um modo genérico, pode-se dizer, que um corpo é um local aonde pode-se somar, subtrair, multiplicar e dividir. Mais formalmente, um corpo é um conjunto de F, conjuntamente com duas operações binárias , ”+”e $\bullet$, tal que:

1. F é um grupo abeliano sobre ”+”, com elemento identidade 0.

2. Os elementos diferentes de zero em F formem um grupo abeliano sobre ”$\bullet$”, com elemento identidade 1.

3. A lei distributiva $a \bullet (b + c) = a \bullet b + a \bullet c$ esteja assegurada.

Um corpo é chamado de finito ou infinito de acordo com qual a subjacência de conjunto que está ligado. Exemplos clássicos de corpos infinitos podem ser: os números Reais, os números Racionais, os números complexos e funções racionais sobre um corpo. Pelo fato dos corpos infinitos não serem utilizados no cifrador AES eles não serão estudados neste trabalho.

Por definição, pode-se dizer que, um corpo finito é um corpo F que possui uma quantidade finita de elementos. A ordem de F é o número de elementos de F. Os corpos finitos também são conhecidos como Corpos de Galois[4].

Alguns fatos que provam a existência e singularidade dos corpos finitos:

1. Se F é um corpo finito, então F contém p^m elementos para algum primo p e inteiro $m \geq 1$.

2. Para todo número de potência prima de ordem p^m, existe um único (isomorfismo) corpo finito de ordem p^m. Este corpo é denotado por F_{p^m} ou $GF(p^m)$.

Note-se que caso p seja primo então F_p é um corpo, e portanto corpo de ordem p é isomórfico a F_p.

2.4 Operações Utilizadas no AES

Várias operações do AES são definidas a nível de byte, com um byte representando um elemento no corpo finito $GF(2^8)$. Outras operações são definidas em termos com palavras de 4 bytes.

Como AES trabalha sobre corpos finitos $GF(2^8)$ pode-se definir que uma operação binária $\bullet$ em um conjunto GF como um mapeamento de matrizes SxS para S. Isto é, uma regra pela qual associa-se cada par ordenado de elementos de S para um elemento de S.

Na proposta original do AES [JD 99a] os blocos de dados (bytes) estão expressos em notação hexadecimal. Na abordagem feita nesta seção será dado mais ênfase

[4] GF - Galois Field

na notação binária, de modo que a própria derivação dos bits em polinômios neste modo é vista de forma mais clara. Porém na análise das operações do AES irá manter-se a notação hexadecimal, devido a facilidade para expressar um elemento (dois carácteres *XXh*).

2.4.1 Corpo GF(2^8)

Os elementos de um corpo finito podem ser representados de várias maneiras diferentes. Para cada potência prima existe um corpo finito simples, de modo que todas as representações de $GF(2^8)$ são isomórficas. Apesar desta equivalência, a representação possui considerável impacto na complexidade de implementação. A equipe de desenvolvimento do AES optou por uma representação clássica de polinômios [JD 99a].

Neste tipo de notação um byte b consiste dos bits $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ considerados como um polinômio com coeficiente em $\{0,1\}$. Isso significa que sua interpretação como polinômio deve ser interpretada como:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

Exemplo. Demonstrar a derivação do byte 01010111 para seu polinômio correspondente.

Partindo-se da representação padrão:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

Substitui-se os coeficientes b_x pelo bits equivalentes de 01010111:

$$0x^7 + 1x^6 + 0x^5 + 1x^4 + 0x^3 + 1x^2 + 1x^1 + 1$$

Efetua-se as multiplicações necessárias e tem-se o polinômio correspondente:

$$x^6 + x^4 + x^2 + x^1 + 1$$

2.4.1.1 Adição

Na representação polinomial, a soma de dois elementos é o polinômio com os coeficientes fornecidos pela operação de soma no módulo 2 dos coeficientes dos dois termos [JD 99a].

Exemplo. Demonstrar a soma dos bytes 01010111 e 10000011.

Convertendo os bytes para notação de polinômios:
01010111 = $x^6 + x^4 + x^2 + x^1 + 1$
10000011 = $x^7 + x + 1$
logo: $(x^6 + x^4 + x^2 + x^1 + 1)$+$(x^7 + x + 1)$ = $(x^7 + x^6 + x^4 + x^2)$

Através da notação hexadecimal 57h + 83h = D4h não consegue-se verificar nenhum procedimento simplificado para a operação. Porém analisando a operação em notação binária:

$$\begin{array}{rr} & 01010111 \\ \oplus & 10000011 \\ \hline & 11010100 \end{array}$$

Percebe-se que a adição corresponde a operação binária simples *XOR* ($\oplus$) a nível de bits.

Todas as condições necessárias para ter um grupo abeliano são satisfeitas: interno, associativo, elemento neutro (00h), elemento inverso (todo elemento é seu próprio inverso aditivo) e comutativo. Como todo elemento é seu próprio inverso aditivo, subtração e adição são os mesmos [JD 99a].

2.4.1.2 Multiplicação

Na representação polinomial, a multiplicação em $GF(2^8)$ corresponde com a multiplicação modular de polinômios com um polinômio binário irredutível de grau 8. Um polinômio é irredutível caso não tenha nenhum divisor diferente de 1 e ele próprio. Para o AES, este polinômio é chamado de *m(x)* e é dado por:

$m(x)=x^8 + x^4 + x^3 + x + 1$

ou 11Bh na notação hexadecimal e 100011011 em notação binária.

Exemplo. Multiplicação de 57h por 83h.

$$(x^6 + x^4 + x^2 + x^1 + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 +$$
$$x^7 + x^5 + x^3 + x^2 + 1 +$$
$$x^6 + x^4 + x^2 + x^1 + 1 +$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

Agora:

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \text{ módulo } x^8 + x^4 + x^3 + x + 1$$
$$= x^7 + x^6 + 1$$

Este resultado em notação hexadecimal tem-se que 57h • 83h = C1 como produto da multiplicação.

Devido a operação módulo, com o polinômio irredutível *m(x)*, o resultado sempre será um polinômio binário de grau inferior a 8. Diferente da operação de adição, não existe uma operação simples a nível de bits.

A multiplicação definida acima é associativa e possui um elemento neutro (01h). Para qualquer polinômio binário $b(x)$ de grau inferior a 8 esta operação de multiplicação pode ser feita através do Algoritmo Extendido de Euclides para calcular $a(x)$, $c(x)$ como $b(x)a(x) + m(x)c(x) = 1$.

Conseqüentemente, $a(x) \bullet b(x)$ mod $m(x) = 1$ ou $b^{-1} = a(x)$ mod $m(x)$. Isto assegura também que $a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x)$, em outras palavras a propriedade distributiva.

Isto faz com que se tenha um conjunto de 256 valores de bytes, utilizando *XOR* ($\oplus$) como adição e a multiplicação sendo definida como sendo uma estrutura sobre um corpo finito $GF(2^8)$.

As operações de multiplicação e similares (como a inversa) também podem ser realizadas através da utilização de uma tabela de logaritmos. Sendo que cada byte $b \neq 0$ pode ser escrito na forma $b = g^e \ (mod\ x^8 + x^4 + x^3 + x + 1)$ com $g = FFh$

para um único expoente $e \in 0...254$, para $e > 254$ tem-se $g^e \equiv g^{e\ mod 255}$. A tabela 2.2 mostra os valores equivalentes dos valores em hexadecimal para g.

Através do uso de um vetor que contenha a tabela 2.2 consegue-se aumentar o desempenho nas operações de multiplicação devido ao fato de não ser mais necessário calcular valores e sim realizar pesquisas nesse vetor. Porém o uso de vetores/tabelas de busca exige mais recursos de memória sendo assim o uso de uma tabela de logaritmos é mais indicada para ser utilizada em computadores, visto que, em implementações de hardware os componentes são desenvolvidos para realizar com alto desempenho estas operações matemáticas.

Tabela 2.2: Tabela de logaritmos para multiplicação

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	0	40	73	80	146	113	174	120	247	186	197	153	34	214	219
1	160	37	32	2	226	93	237	107	193	89	74	65	254	15	4	64
2	200	137	77	208	72	245	42	88	11	69	133	162	22	138	147	90
3	233	116	129	110	114	75	105	78	39	180	55	192	44	21	104	166
4	240	239	177	27	117	139	248	94	112	86	30	253	82	10	128	8
5	51	68	109	189	173	183	202	14	62	151	178	131	187	238	130	148
6	18	125	156	210	169	26	150	228	154	161	115	12	145	19	118	63
7	79	163	220	3	95	132	232	211	84	176	61	142	144	235	206	231
8	25	49	24	230	217	252	67	53	157	207	179	249	33	215	134	106
9	152	140	126	236	70	76	38	35	122	29	50	98	168	97	48	205
A	91	111	108	198	149	28	229	143	213	46	223	225	242	199	54	99
B	102	136	191	195	218	123	171	92	227	121	23	234	170	85	188	241
C	58	244	165	57	196	100	250	36	209	167	66	175	190	9	13	212
D	194	81	201	45	155	96	52	83	185	6	59	159	158	71	103	243
E	119	221	203	31	5	41	43	56	135	127	172	224	17	204	251	60
F	124	47	216	141	101	7	182	222	184	87	20	164	246	181	16	1

Exemplo. Multiplicação de 57h por 83h utilizando a tabela de logaritmos.

A partir dos valores originais em hexadecimal procura-se o valor do coeficiente de g na tabela 2.2, sendo que a ordem de busca é linha e depois coluna. Sendo o primeiro valor 57h, o valor do coeficiente de g estará na linha 5, coluna 7 e corresponde a 14. Para o valor 83h o valor do coeficiente g estará na linha 8, coluna 3 e corresponde a 230. Através da tabela obtém-se:

$$57\text{h} = g^{14}$$

$$83\text{h} = g^{230}$$

Logo 57h • 83h $= g^{14} \bullet g^{230} = g^{244} =$ C1.

Sendo os multiplicadores de mesma base (g) o resultado da operação é a mesma base e o coeficiente final consiste da soma dos coeficientes. Contudo o valor que obtém-se no final da operação é o coeficiente de g resultante da multiplicação, para transformar este coeficiente em um valor hexadecimal deve-se fazer uma busca nos elementos da tabela 2.2. A linha e coluna correspondentes ao valor do coeficiente são o valor em hexadecimal do resultado da operação.

2.4.1.3 Multiplicação por x

Realizando a multiplicação de $b(x)$ por um polinômio x tem-se:

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

$x \bullet b(x)$ é obtido reduzindo-se o resultado acima com uma operação de módulo com $m(x)$. Caso $b_7 = 0$ a redução será a operação identidade, se $b_7 = 1$ o polinômio irredutível $m(x)$ deve ser subtraído. Isto significa que a multiplicação por x (02h) pode ser implementada a nível de byte por uma rotação à esquerda e a uma operação condicional de *XOR* ($\oplus$) com 1Bh.

2.4.2 Polinômios com Coeficientes em GF(2^8)

Polinômios podem ser definidos com coeficientes em um corpo finito $GF(2^8)$. Neste caso um vetor de 4 bytes corresponde a um polinômio com grau abaixo de 4.

A adição de polinômios pode ser realizada pela simples soma dos coeficientes correspondentes. Como a adição em um corpo finito $GF(2^8)$ é uma operação a nível de bit (*XOR*), a adição de dois vetores também é uma operação a nível de bit (*XOR*).

No caso de multiplicações a execução torna-se mais complicada. Uma noção deste grau de complexidade pode ser percebida no exemplo em que assume-se dois polinômios sobre o corpo $GF(2^8)$:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \text{ e } b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

O produto de $c(x) = a(x) \bullet b(x)$ é dado por:

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \quad \text{com}$$

$$c_0 = a_0 \bullet b_0 \qquad c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \qquad c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \qquad c_6 = a_3 \bullet b_3$$

$$c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

Pode-se perceber visivelmente que $c(x)$ não pode ser representado por um vetor de 4 bytes. Através da redução de $c(x)$ modular com um polinômio de grau 4, o resultado obtido é um polinômio de grau menor que 4. No AES isto é feito com o polinômio $M(x) = x^4 + 1$. Sendo $x^i \ mod \ (x^4 + 1) = x^{i \ mod \ 4}$ o produto modular de $a(x)$ e $b(x)$, denotado por $d(x) = a(x) \otimes b(x)$ é dado por:

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

$$d_0 = a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$d_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$d_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3$$

$$d_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

A operação consiste da multiplicação por um polinômio determinado $a(x)$ pode ser escrita como uma matriz de multiplicação onde a matriz é uma matriz circulante.

Utilizando a escrita por matriz tem-se:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Deve ser mencionado que $x^4 + 1$ não é um polinômio irredutível sobre o corpo finito $GF(2^8)$, conseqüentemente a multiplicação por um polinômio pré-determinado não é necessariamente inversível. No caso do cifrador AES foi escolhido um polinômio pré-determinado que é passível de ser invertido.

2.4.2.1 Multiplicação por x

Efetuando-se a multiplicação de $b(x)$ por um polinômio x tem-se:

$$b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

$x \otimes b(x)$ é obtido pela redução do resultado acima pelo módulo $x^4 + 1$. O resultado desta operação é:

$$b_2x^3 + b_1x^2 + b_0x + b_3$$

A multiplicação por x é equivalente a multiplicação por uma matriz com todos $a_i = 00h$, com exceção de $a_1 = 01h$. Tomando $c(x) = x \otimes b(x)$, tem-se:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00h & 00h & 00h & 01h \\ 01h & 00h & 00h & 00h \\ 00h & 01h & 00h & 00h \\ 00h & 00h & 01h & 00h \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Por esta razão, a multiplicação por x, ou potências de x, corresponde a uma rotação cíclica dos bits dentro do vetor.

2.5 Estrutura do Cifrador

O AES é um cifrador de bloco com tamanho de bloco e chave variáveis entre 128, 192 e 256 bits. Isto significa que pode-se ter tamanho de blocos com tamanhos de chaves diferentes. Em função do tamanho de bloco e chaves é que será determinado a quantidade de rodadas necessárias para cifrar/decifrar. O processo para cifrar e decifrar pode ser visualizado de uma maneira genérica na figura 2.2.

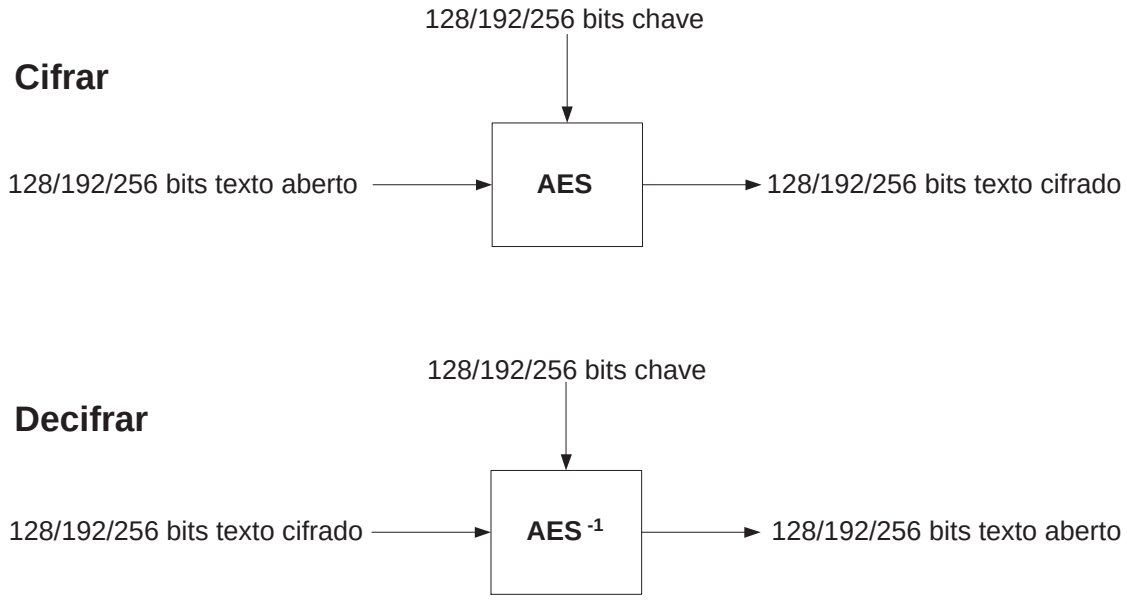

Figura 2.2: Visão genérica do AES

O AES opera desta forma com um determinado número de blocos de 32 bits que são ordenados em colunas de 4 bytes, as quais são chamadas de *Nb*[5]. Os valores de *Nb* possíveis são de 4, 6 e 8 equivalentes a blocos de 128, 192 e 256 bits. Assim sempre que for referido *Nb*, significa que tem-se *Nb* x 32 bits de tamanho de bloco de dados. A chave é agrupada da mesma forma em colunas que o bloco de dados mas com a sigla *Nk*[6]. Na tabela 2.3 tem-se um exemplo para *Nb*=4 e na tabela 2.4 com *Nk*=4.

Tabela 2.3: Exemplo de bloco de dados com *Nb*=4

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Tabela 2.4: Exemplo de chave com *Nk*=4

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

[5]Number of blocks, Número de Blocos

[6]Number of Key, Número de Chaves

Com base nos valores que *Nb* e *Nk* podem assumir é que será determinado a quantidade de rodadas a serem executadas, identificada pela sigla *Nr*[7]. Através da tabela 2.5 pode-se verificar as combinações possíveis e número de rodadas.

Tabela 2.5: *Nr* em função do tamanho de bloco e chave no AES.

Nr	*Nb*=4	*Nb*=6	*Nb*=8
Nk=4	10	12	14
Nk=6	12	12	14
Nk=8	14	14	14

Para entender a definição de uma rodada no AES é necessário compreender o conceito de estado no cifrador. O estado pode ser definido como uma matriz que contém os dados durante as transformações no processamento, essa matriz contém 4 linhas e *Nb* colunas. A representação de um estado pode ser observada conforme a matriz disposta na tabela 2.3. Percebe-se assim que o AES é um cifrador orientado ao byte em que cada rodada é composta de três transformações distintas e inversíveis, chamadas de camadas.

A escolha destas camadas foi feita com base na estratégia *Wide Trail*[8] [DAE 95], na qual cada camada possui sua própria função. Antes da primeira rodada ser realizada é aplicada uma camada de adição de chaves, a qual foi motivada pelo raciocínio de que qualquer operação realizada antes da adição de chaves é facilmente reversível e deste modo redundante. A descrição das três camadas escolhidas para serem utilizadas no AES:

- Camada Línear[9]: garante a característica de grande difusão através das múltiplas rodadas;
- Camada Não Línear[10]: aplicação paralela de Caixas-S para obter a melhor eficiência de propriedades não lineares, isto é uma característica de difusão;

[7] Number of Rounds, Número de Rodadas

[8] uma metodologia de projeto que provê resistência contra criptoanálise linear e diferencial, defendida por Joan Daemen em sua tese de doutorado em 1.995

[9] Linear Mixing Layer

[10] Non-Linear Layer

- Camada de Adição de Chaves[11]: adição de chave da rodada através de *XOR* sobre o estado intermediário.

Porém o processo de cifrar e decifrar no AES não são funções idênticas, como ocorre na maioria dos cifradores. Isto ocorre devido ao fato de ser um algoritmo essencialmente matemático[12]. Sendo assim o tempo para execução do processo de cifrar é diferente do tempo de execução do processo de decifrar.

2.6 Transformações por Rodada

As operações as quais cada bloco (estado) é sujeitado durante o processo para cifrar em cada rodada são:

- Byte Sub: os bytes de cada bloco são substituídos por seus equivalentes em uma tabela de substituição (Caixa-S), a qual não é linear. Emprego de técnicas de confusão segundo Shannon [SHA 49];

- Shift Row (Deslocamento de Linha): Nesta etapa os bytes são rotacionados em grupos de quatro bytes, de modo que cada grupo diferente irá interferir em cada outro. Emprego de técnicas de difusão segundo Shannon [SHA 49];

- Mix Column: Cada grupo de quatro bytes é sujeitado a uma multiplicação modular. Isto proporciona a cada byte do grupo a influenciar em todos os outros bytes. Emprego de técnicas de difusão segundo Shannon [SHA 49];

- Add Round Key(Adição da Chave de Rodada): nesta fase o bloco de dados é alterado através da subchave da rodada, a qual possui o mesmo tamanho do bloco, que realiza uma operação *XOR* com o bloco inteiro.

[11] Key Addition Layer

[12] uso de multiplicações por matrizes que exigem o uso de matrizes inversas para a obtenção dos valores originais para o processo de decifrar pode ser citado como exemplo

2.6.1 Transformação ByteSub

A transformação ByteSub modifica cada byte do estado através do uso de uma Caixa-S. Devido à orientação ser de substituição ao byte significa que podem ser realizadas várias substituições em paralelo.

A Caixa-S a ser utilizada pode ser criada em tempo de execução visto que ela é construída através de uma operação matemática. A Caixa-S é construída através da composição de duas transformações:

1. Realizando a inversa multiplicativa em $GF(2^8)$, com a representação definida por um polinômio com coeficientes em {0,1}, sendo o elemento $00h$ é mapeado em si mesmo.

2. Aplicando-se uma operação similar de transformação definida por:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

A aplicação desta operação para cada byte do estado consiste na operação denotada por ByteSub(Estado). Para realizar o processo de decifrar é necessário fazer a operação através da matriz inversa, que é obtida através da inversa multiplicativa da mesma sobre o corpo $GF(2^8)$.

Através da figura 2.3 pode ser visualizada a forma de atuação da operação ByteSub sobre o estado. Na figura 2.3 tem-se do lado esquerdo um estado com *Nb* = 4, isto é um bloco de dados de 128 bits, o qual cada elemento do estado[13] é mapeado através

[13]equivalente a um byte

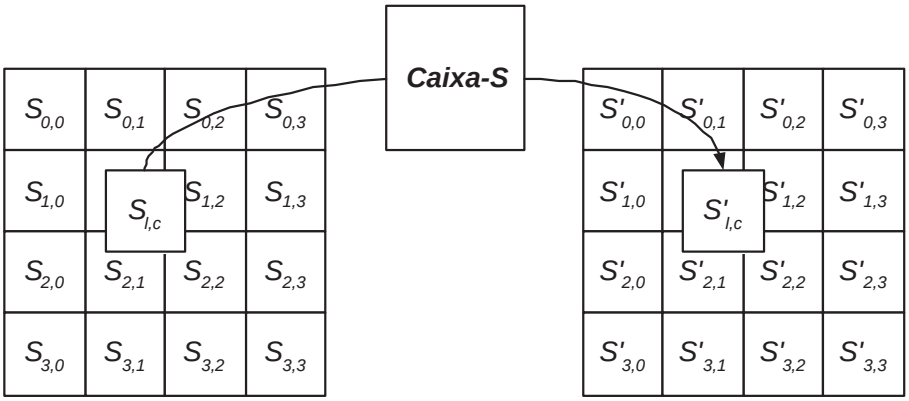

Figura 2.3: Modo de operação da transformação ByteSub em um estado do AES.

da Caixa-S em um novo elemento no estado novo ocupando exatamente a mesma posição dentro do estado.

A transformação também pode ser implementada de uma forma eficiente através de uma Caixa-S de dimensão 16x16 armazenada em memória. Isto diminuiria o tempo inicial para realizar a operação porém exigiria mais recursos de memória do sistema. A Tabela 2.6 demonstra esta Caixa-S em notação hexadecimal, aonde a transformação da Caixa-S inversa é feita através do emprego da mesma Caixa-S no sentido oposto.

Tabela 2.6: Caixa-S utilizada pelo AES baseada em notação hexadecimal

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	C6	37	87	47	DF	46	6	AC	F3	E0	86	42	1F	8D	4A	97
1	5C	D8	6C	27	5F	65	84	FF	2A	BD	DA	A	39	BA	D7	FC
2	8B	2F	C9	92	93	3	8F	3C	B3	AA	AE	EF	E7	7D	E3	A1
3	B0	8C	C2	CC	71	99	A0	59	80	D1	F8	DE	4E	82	DB	A7
4	60	C8	32	51	41	16	55	FA	D5	43	9D	CB	62	CE	2	B8
5	C5	ED	F0	2E	F2	3F	EB	45	56	4C	1B	63	54	34	75	C
6	FD	E	5A	4F	C4	24	C3	A8	A4	6F	D0	7	F5	33	9	7A
7	E5	CA	F4	8	D9	29	73	AF	3B	9B	5D	E2	F1	F	CF	DD
8	2C	30	C1	3E	5	89	B4	81	BC	8A	17	23	B6	25	61	C7
9	F6	E8	4	3D	D2	52	F9	78	94	1E	7B	B1	1D	15	40	4D
A	FE	D3	53	50	64	90	B2	35	DC	CD	3A	D6	E9	A9	BE	67
B	8E	7C	83	26	28	AD	14	6A	36	95	BF	5E	A6	57	1A	70
C	5B	77	A2	12	31	9A	BB	9C	7E	2D	B7	1	44	2B	48	58
D	F7	13	AB	96	74	C0	9F	10	E6	A3	85	6B	98	EC	21	19
E	EE	7F	79	E1	66	6D	18	B9	49	11	88	6E	1C	A5	72	D
F	38	EA	68	20	B	9E	D4	76	E4	69	22	0	FB	B5	4B	91

2.6.2 Transformação Deslocamento de Linha (ShiftRow)

Nesta transformação as linhas da matriz que representam o estado são rotacionadas de acordo com uma tabela de rotacionamento. A primeira linha do estado não sofre nenhum rotacionamento, sendo que as demais linhas sofrem um rotacionamento a nível de mudança de posição dos elementos na linha através de um rotacionamento cíclico à esquerda. A quantidade de elementos a ser efetuado o rotacionamento é dada na tabela 2.7 sendo que na nomenclatura *Cx* o *x* representa o número da linha em uma notação de 0 a 3.

Tabela 2.7: Deslocamento em função do tamanho de bloco no AES.

Nb	*C1*	*C2*	*C3*
4	1	2	3
6	1	2	3
8	1	3	4

O emprego de rotacionamento cíclico tem por finalidade aumentar a difusão sobre os elementos do estado. Note-se que de modo diferente que ocorre com os outros cifradores da atualidade aonde o rotacionamento é feito a nível de bits no AES o rotacionamento é realizado a nível de bytes. A motivação para esta operação ser feita a nível de byte está relacionada ao fato de que o estado já sofrera uma alteração a nível de bits no processo de adição de chaves no início do processo de cifrar.

Através da figura 2.4 pode-se visualizar mais claramente a ação da transformação ShiftRow(Estado). Na figura 2.4 foram colocadas letras do alfabeto como valores equivalentes aos bytes do estado para melhor visualizar a transformação.

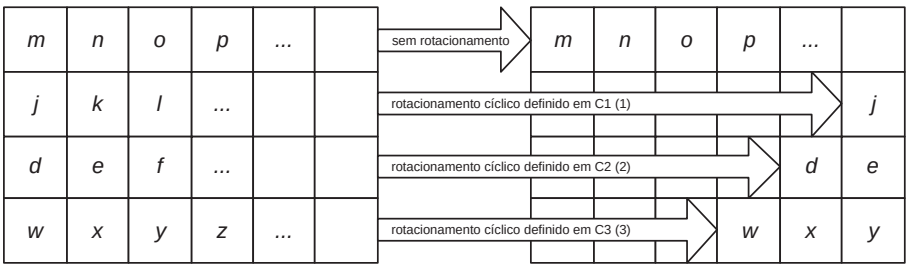

Figura 2.4: Modo de operação da transformação ShiftRow em um estado do AES.

Observando a tabela 2.8 pode-se analisar o efeito da transformação sobre o estado original a esquerda para o estado final na direita após a operação ser concluída.

Tabela 2.8: Exemplo de transformação ShifRow sobre estado de *Nb*=4.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$\Longrightarrow$	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$		$a_{1,3}$	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$		$a_{2,2}$	$a_{2,3}$	$a_{2,0}$	$a_{2,1}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$		$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,0}$

A operação inversa é feita rotacionando-se o mesmo número de posições na tabela 2.7 no sentido contrário, isto é um rotacionamento cíclico à direita.

2.6.3 Transformação MixColumn

Na transformação MixColumn as colunas do estado são considerados como polinômios sobre $GF(2^8)$ e feita uma multiplicação módulo $x^4 + 1$ com um polinômio fixo $c(x)$, dado por:

$$c(x) = 03hx^3 + 01hx^2 + 01hx + 02h$$

Este polinômio é reversível, o que torna possível a operação de decifrar. A transformação pode ser escrita através de uma matriz de multiplicação como $b(x) = c(x) \otimes a(x)$.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02h & 03h & 01h & 01h \\ 01h & 02h & 03h & 01h \\ 01h & 01h & 02h & 03h \\ 03h & 01h & 01h & 02h \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

A figura 2.5 mostra a operação realizada durante a transformação MixColumn(estado).

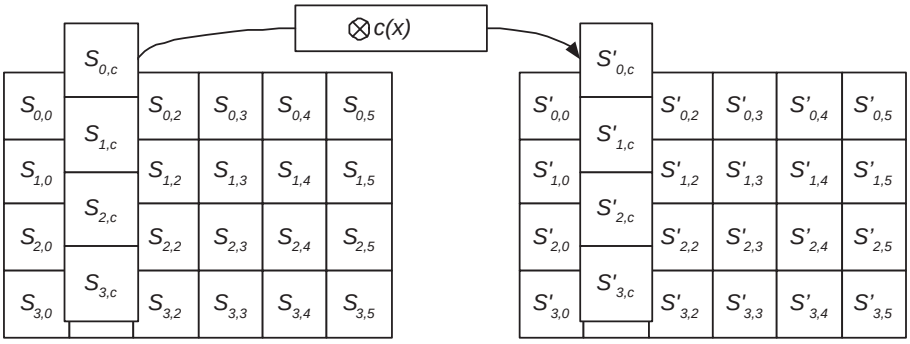

Figura 2.5: Modo de operação da transformação MixColumn em um estado do AES.

A transformação inversa de MixColumn é dada através da multiplicação pelo polinômio $d(x)$, definido por:

$$d(x) = 0Bhx^3 + 0Dhx^2 + 09hx + 0Eh$$

A transformação pode ser escrita através de uma matriz de multiplicação como:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0Eh & 0Bh & 0Dh & 09h \\ 09h & 0Eh & 0Bh & 0Dh \\ 0Dh & 09h & 0Eh & 0Bh \\ 0Bh & 0Dh & 09h & 0Eh \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

2.6.4 Adição de Chave da Rodada

A transformação realizada na adição de chaves é uma operação *XOR* byte a byte do estado com a subchave da rodada. O tamanho da chave de rodada é igual ao tamanho do bloco *Nb*.

Na tabela 2.9 tem-se uma ilustração da matriz que representa o estado sobre a qual a operação *XOR* com a matriz da chave de rodada, que produz o resultado que será o próximo estado do cifrador.

Tabela 2.9: Transformação Adição de Chave de Rodada sobre estado de *Nb*=4.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$\oplus$	$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$=$	$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$		$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$		$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$		$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$		$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$		$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$		$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

2.7 Geração de SubChaves do AES

O algoritmo de geração de chaves cria a partir da chave principal do AES uma quantidade de subchaves igual a quantidade de rodadas mais um, sendo que cada subchave possui o mesmo tamanho da chave principal. A quantidade de subchaves gerada é igual a quantidade de rodadas mais um, devido ao fato de antes de ser realizada a primeira rodada é feita uma adição de chaves.

O resultado produzido pelo processo de geração de chaves consiste em um vetor unidimensional com palavras (*w*) de 4 bytes (32bits), sendo que a cada rodada é utilizada como subchave a quantidade de quatro palavras seqüencialmente (no caso de *Nk*=4 e *Nb*=4), caminhando progressivamente sobre os elementos do vetor. O processo de geração de subchaves também é conhecido como expansão de chave. Através da figura 2.6 pode-se visualizar o vetor que contém as chaves, na figura cada byte é nomeado de w_x onde x é a posição do byte dentro do vetor.

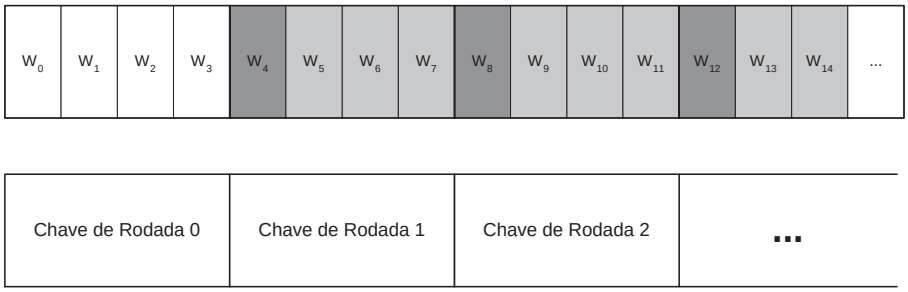

Figura 2.6: Exemplo do vetor de chaves para *Nk*=4.

Deve-se considerar que antes de realizar a expansão as *Nk* palavras são preenchidas com a chave original do cifrador. O processo de expansão de uma chave no

cifrador AES é definido pelo algoritmo mostrado nesta seção.

Para compreender corretamente o seu funcionamento e significado torna-se necessário compreender algumas funções utilizadas durante o processo de expansão de chaves:

1. A constante $cons(i)$ contém um valor obtido por $(x^{i-1}, 00h, 00h, 00h)$, com x^{i-1} sendo uma potência de x (x é definido por $02h$) no corpo $GF(2^8)$;

2. A Operação Rotacao() faz um rotacionamento cíclico à esquerda de uma posição nos quatro bytes de s;

3. A operação SubPalavra() aplica a Caixa-S do AES em cada um dos 4 bytes da palavra.

O algoritmo para expansão de chave para *Nk*=4 e *Nk*=6 representado em pseudo-código:

```
ExpansaoChave(byte Chave[4 * Nk], palavra w[Nb * (Nr + 1)], Nk)
inicio
  i=0
  enquanto (i < Nk)
    w[i] = palavra[Chave[4*i],Chave[4*i+1],Chave[4*i+2],Chave[4*i+3]]
    i = i +1
  fim enquanto
  i = Nk
  enquanto (i < Nb * (Nr + 1))
    temp = w[i - 1]
    se (i mod Nk = 0)
      entao
        temp = SubPalavra(Rotacao(temp)) xor cons[i / Nk]
      senao
        se (Nk = 8 e i mod Nk = 4)
          entao
            temp = SubPalavra(temp)
        fim se
    fim se
    w[i] = w[i - Nk] xor temp
    i = i + 1
  fim enquanto
fim
```

É importante saber que o processo de geração de chaves com tamanho de bloco 256 bits (*Nk*=8) é um pouco diferente do processo de chaves com 128 bits (*Nk*=4) e 192 bits (*Nk*=6) mostrado. A diferença no algoritmo consiste de quando *Nk*=8 e $i-4$ for múltiplo de *Nk* então a operação SubPalavra() é aplicada a $w[[i-1]]$ antes do *XOR*.

Exemplo. A tabela 2.10 demonstra o processo de geração de chaves de rodadas (*Nk*=4) para : 0a 1b 2c 3d 4e 5a 6b 7c 8d 9e aa ab ac ad ae af . Nesse exemplo encontra-se descrito o processo detalhado conforme o algoritmo descrito anteriormente.

Tabela 2.10: Geração das chaves de rodadas do AES ($Nk = 4$) para : 0a 1b 2c 3d 4e 5a 6b 7c 8d 9e aa ab ac ad ae af

i	temp	Após RotPalavra	Após Caixa-S	Rcon$[i/Nk]$	Após *XOR* $\oplus$ Rcon	$w[i-Nk]$	$w[i]$
0							0a1b2c3d
1							4e5a6b7c
2							8d9eaaab
3							acadaeaf
4	acadaeaf	adaeafac	95e47991	01000000	94e47991	0a1b2c3d	9eff55ac
5	9eff55ac					4e5a6b7c	d0a53ed0
6	d0a53ed0					8d9eaaab	5d3b947b
7	5d3b947b					acadaeaf	f1963ad4
8	f1963ad4	963ad4f1	908048a1	02000000	928048a1	9eff55ac	0c7f1d0d
9	0c7f1d0d					d0a53ed0	dcda23dd
10	dcda23dd					5d3b947b	81e1b7a6
11	81e1b7a6					f1963ad4	70778d72
12	70778d72	778d7270	f55d4051	04000000	f15d4051	0c7f1d0d	fd225d5c
13	fd225d5c					dcda23dd	21f87e81
14	21f87e81					81e1b7a6	a019c927
15	a019c927					70778d72	d06e4455
16	d06e4455	6e4455d0	9f1bfc70	08000000	971bfc70	fd225d5c	6a39a12c
17	6a39a12c					21f87e81	4bc1dfad
18	4bc1dfad					a019c927	ebd8168a
19	ebd8168a					d06e4455	3bb652df
20	3bb652df	b652df3b	4e009ee2	10000000	5e009ee2	6a39a12c	34393fce
21	34393fce					4bc1dfad	7ff8e063
22	7ff8e063					ebd8168a	9420f6e9
23	9420f6e9					3bb652df	af96a436

24	af96a436	96a436af	90490579	20000000	b0490579	34393fce	84703ab7
25	84703ab7					7ff8e063	fb88dad4
26	fb88dad4					9420f6e9	6fa82c3d
27	6fa82c3d					af96a436	c03e880b
28	c03e880b	3e880bc0	b2c42bba	40000000	f2c42bba	84703ab7	76b4110d
29	76b4110d					fb88dad4	8d3ccbd9
30	8d3ccbd9					6fa82c3d	e294e7e4
31	e294e7e4					c03e880b	22aa6fef
32	22aa6fef	aa6fef22	aca8df93	80000000	2ca8df93	76b4110d	5a1cce9e
33	5a1cce9e					8d3ccbd9	d7200547
34	d7200547					e294e7e4	35b4e2a3
35	35b4e2a3					22aa6fef	171e8d4c
36	171e8d4c	1e8d4c17	725d29f0	1b000000	695d29f0	5a1cce9e	3341e76e
37	3341e76e					d7200547	e461e229
38	e461e229					35b4e2a3	d1d5008a
39	d1d5008a					171e8d4c	c6cb8dc6
40	c6cb8dc6	cb8dc6c6	1f5db4b4	36000000	295db4b4	3341e76e	1a1c53da
41	1a1c53da					e461e229	fe7db1f3
42	fe7db1f3					d1d5008a	2fa8b179
43	2fa8b179					c6cb8dc6	e9633cbf

Pode-se observar que ao contrário do cifrador do DES, que utilizava-se de operações simples de permuta e rotacionamento de bits, o processo de expansão de chave do AES utiliza-se de operações matemáticas e Caixa-S. A finalidade dessa lógica encontra-se no desempenho e características de difusão nas subchaves. As 4 palavras w de 4 bytes são dispostas da mesma forma que o estado para serem utilizadas durante o processo de cifrar/decifrar conforme demonstrado na tabela 2.11.

Tabela 2.11: Disposição da Chave de Rodada 0 na matriz estado de *Nb*=4.

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \oplus \begin{bmatrix} 0a & 4e & 8d & ac \\ 1b & 5a & 9e & ad \\ 2c & 6b & aa & ae \\ 3d & 7c & ab & af \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

2.8 Processo de Cifrar do AES

O processo de cifrar do AES envolve uma aplicação seqüencial de funções principais, sendo que, todas essas funções foram explicadas nas seções anteriores. Na figura 2.7 pode-se analisar a estrutura de cifrar do AES com uma visão macro.

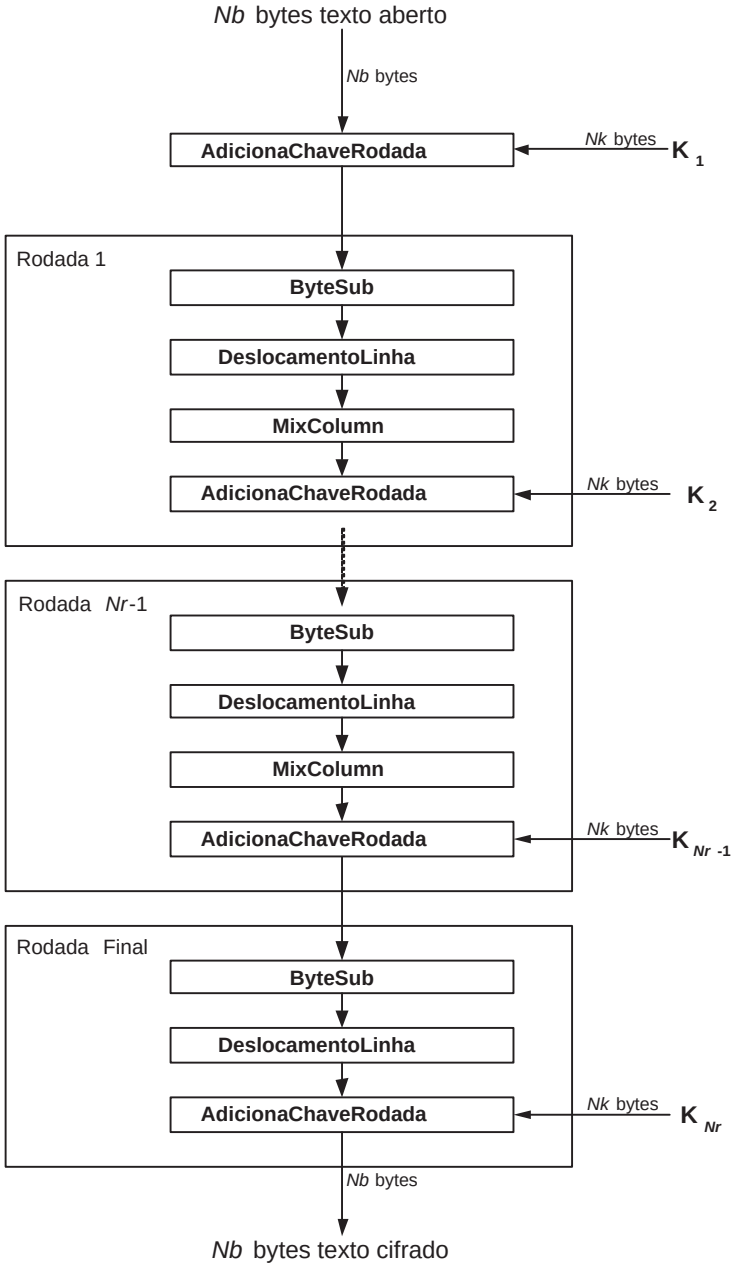

Figura 2.7: Fluxo para cifrar no AES

Ao analisar essa visão macro do AES pode-se perceber os indicadores *Nb*, *Nk* e *Nr* que possuem seus valores de acordo com o tamanho de bloco e chave a serem utilizados. Porém um aspecto que fica mais evidenciado, e que é diferente da maioria dos cifradores da atualidade, é a primeira adição de chaves (K_1) e da mesma forma a finalização do cifrador com uma rodada incompleta[14].

O Algoritmo para cifrar do AES pode ser especificado em pseudo-código da seguinte forma:

```
Cifrar(byte entrada[4*Nb], byte saida[4*Nb], palavra w[Nb*(Nr+1)])
  Inicio
    byte estado[4,Nb]
    estado = entrada
    AdicionaChaveRodada(estado, w)

    Para rodada = 1 ate (Nr  1)
      ByteSub(estado)
      DeslocamentoLinha(estado)
      MixColumns(estado)
      AdicionaChaveRodada(estado, w + rodada * Nb)
      rodada=rodada+1
    fim para

    ByteSub(estado)
    DeslocamentoLinha(estado)
    AdicionaChaveRodada(estado, w + rodada * Nb)
    saida = estado
  Fim
```

Exemplo. Processo de cifrar com *Nb*=4 e *Nk*=4.

Chave: 0a 1b 2c 3d 4e 5a 6b 7c 8d 9e aa ab ac ad ae af

Texto Aberto: 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

Nesse exemplo encontra-se descrito o processo de cifrar conforme o algoritmo descrito.

```
Rodada: 0
                    Estado Inicial: 00 44 88 cc 11 55 99 dd 22 66 aa ee 33 77 bb ff
                   Chave de Rodada: 0a 4e 8d ac 1b 5a 9e ad 2c 6b aa ae 3d 7c ab af

Rodada: 1
                    Estado Inicial: 0a 0a 05 60 0a 0f 07 70 0e 0d 00 40 0e 0b 10 50
```

[14] a operação MixColumn foi omitida

```
                    Apos Caixa-S: 67 67 6b d0 67 76 c5 51 ab d7 63 09 ab 2b ca 53
            Apos Deslocamento Linha: 67 67 6b d0 76 c5 51 67 63 09 ab d7 53 ab 2b ca
                    Apos MixColumns: 64 38 a5 0f 7d 46 04 b6 22 56 0a 47 1a 28 11 54
                    Chave de Rodada: 9e d0 5d f1 ff a5 3b 96 55 3e 94 3a ac d0 7b d4

Rodada: 2
                     Estado Inicial: fa e8 f8 fe 82 e3 3f 20 77 68 9e 7d b6 f8 6a 80
                       Apos Caixa-S: 2d 9b 41 bb 13 11 75 b7 f5 45 0b ff 4e 41 02 cd
            Apos Deslocamento Linha: 2d 9b 41 bb 11 75 b7 13 0b ff f5 45 cd 4e 41 02
                    Apos MixColumns: af 03 f4 1f df 25 71 50 66 d9 c4 24 ec a0 03 84
                    Chave de Rodada: 0c dc 81 70 7f da e1 77 1d 23 b7 8d 0d dd a6 72

Rodada: 3
                     Estado Inicial: a3 df 75 6f a0 ff 90 27 7b fa 73 a9 e1 7d a5 f6
                       Apos Caixa-S: 0a 9e 9d a8 e0 16 60 cc 21 2d 8f d3 f8 ff 06 42
            Apos Deslocamento Linha: 0a 9e 9d a8 16 60 cc e0 8f d3 21 2d 42 f8 ff 06
                    Apos MixColumns: e3 ac b0 5b ee c8 82 02 df 50 09 18 03 e1 b4 22
                    Chave de Rodada: fd 21 a0 d0 22 f8 19 6e 5d 7e c9 44 5c 81 27 55

Rodada: 4
                     Estado Inicial: 1e 8d 10 8b cc 30 9b 6c 82 2e c0 5c 5f 60 93 77
                       Apos Caixa-S: 72 5d ca 3d 4b 04 14 50 13 31 ba 4a cf d0 dc f5
            Apos Deslocamento Linha: 72 5d ca 3d 04 14 50 4b ba 4a 13 31 f5 cf d0 dc
                    Apos MixColumns: a7 03 bc 4a 5a 64 8f 24 1d 97 d7 6b d9 3c bd 9e
                    Chave de Rodada: 6a 4b eb 3b 39 c1 d8 b6 a1 df 16 52 2c ad 8a df

Rodada: 5
                     Estado Inicial: cd 48 57 71 63 a5 57 92 bc 48 c1 39 f5 91 37 41
                       Apos Caixa-S: bd 52 5b a3 fb 06 5b 4f 65 52 78 12 e6 81 9a 83
            Apos Deslocamento Linha: bd 52 5b a3 06 5b 4f fb 78 12 65 52 83 e6 81 9a
                    Apos MixColumns: 90 bd 83 83 ba 34 eb 22 d5 1c 46 49 bf 68 de 78
                    Chave de Rodada: 34 7f 94 af 39 f8 20 96 3f e0 f6 a4 ce 63 e9 36

Rodada: 6
                     Estado Inicial: a4 c2 17 2c 83 cc cb b4 ea fc b0 ed 71 0b 37 4e
                       Apos Caixa-S: 49 25 f0 71 ec 4b 1f 8d 87 b0 e7 55 a3 2b 9a 2f
            Apos Deslocamento Linha: 49 25 f0 71 4b 1f 8d ec e7 55 87 b0 2f a3 2b 9a
                    Apos MixColumns: 87 9d db e7 c2 47 48 e3 a6 6e 15 53 29 78 57 e0
                    Chave de Rodada: 84 fb 6f c0 70 88 a8 3e 3a da 2c 88 b7 d4 3d 0b

Rodada: 7
                     Estado Inicial: 03 66 b4 27 b2 cf e0 dd 9c b4 39 db 9e ac 6a eb
                       Apos Caixa-S: 7b 33 8d cc 37 8a e1 c1 de 8d 12 b9 0b 91 02 e9
            Apos Deslocamento Linha: 7b 33 8d cc 8a e1 c1 37 12 b9 de 8d e9 0b 91 02
                    Apos MixColumns: 88 ec 16 55 ab 31 fc 2c f5 a6 43 fc dc 1b aa f1
```

```
                Chave de Rodada: 76 8d e2 22 b4 3c 94 aa 11 cb e7 6f 0d d9 e4 ef

Rodada: 8
                  Estado Inicial: fe 61 f4 77 1f 0d 68 86 e4 6d a4 93 d1 c2 4e 1e
                    Apos Caixa-S: bb ef bf f5 c0 d7 45 44 69 3c 49 dc 3e 25 2f 72
          Apos Deslocamento Linha: bb ef bf f5 d7 45 44 c0 49 dc 69 3c 72 3e 25 2f
                  Apos MixColumns: 34 e8 e5 b9 a7 24 a9 05 68 4b 46 3c ac cf bd a6
                  Chave de Rodada: 5a d7 35 17 1c 20 b4 1e ce 05 e2 8d 9e 47 a3 4c

Rodada: 9
                  Estado Inicial: 6e 3f d0 ae bb 04 1d 1b a6 4e a4 b1 32 88 1e ea
                    Apos Caixa-S: 9f 75 70 e4 ea f2 a4 af 24 2f 49 c8 23 c4 72 87
          Apos Deslocamento Linha: 9f 75 70 e4 f2 a4 af ea 49 c8 24 2f 87 23 c4 72
                  Apos MixColumns: e6 f6 ea ab 3c 46 9d 28 6d 3f c0 c6 14 b5 88 16
                  Chave de Rodada: 33 e4 d1 c6 41 61 d5 cb e7 e2 00 8d 6e 29 8a c6

Rodada: 10
                  Estado Inicial: d5 12 3b 6d 7d 27 48 e3 8a dd c0 4b 7a 9c 02 d0
                    Apos Caixa-S: 03 c9 e2 3c ff cc 52 11 7e c1 ba b3 da de 77 70
          Apos Deslocamento Linha: 03 c9 e2 3c cc 52 11 ff ba b3 7e c1 70 da de 77
                  Chave de Rodada: 1a fe 2f e9 1c 7d a8 63 53 b1 b1 3c da f3 79 bf
               Apos Chave Rodada: 19 37 cd d5 d0 2f b9 9c e9 02 cf fd aa 29 a7 c8

                   Texto Cifrado: 19 37 cd d5 d0 2f b9 9c e9 02 cf fd aa 29 a7 c8
```

2.9 Processo de Decifrar do AES

O processo de decifrar no AES consiste na execução de transformações diferentes, devido a sua essência matemática. Ao contrário do DES que tem sua estrutura baseada em Feistel, que possui a característica de ser reversível apenas invertendo-se a seqüencia das chaves para decifrar, o AES necessita de inversas matemáticas de suas transformações para realizar o processo de decifrar[15]. Observando o AES em uma visão macro poderia ser considerado como uma sequência de transformações matemáticas e, como uma, o seu processo reverso consiste na aplicação da sequência inversa a da original.

O processo de expansão de chaves continua sendo o mesmo, descrito anteriormente, porém as funções de ByteSub, DeslocamentoLinha e MixColumn necessitam ser as suas inversas matemáticas para realizar o processo de decifrar .

[15]Essa característica faz com que o AES possua tempos diferentes para cifrar e decifrar

Analisando o fluxo demonstrado na figura 2.8 pode-se perceber as diferenças a nível de função do processo utilizado para cifrar.

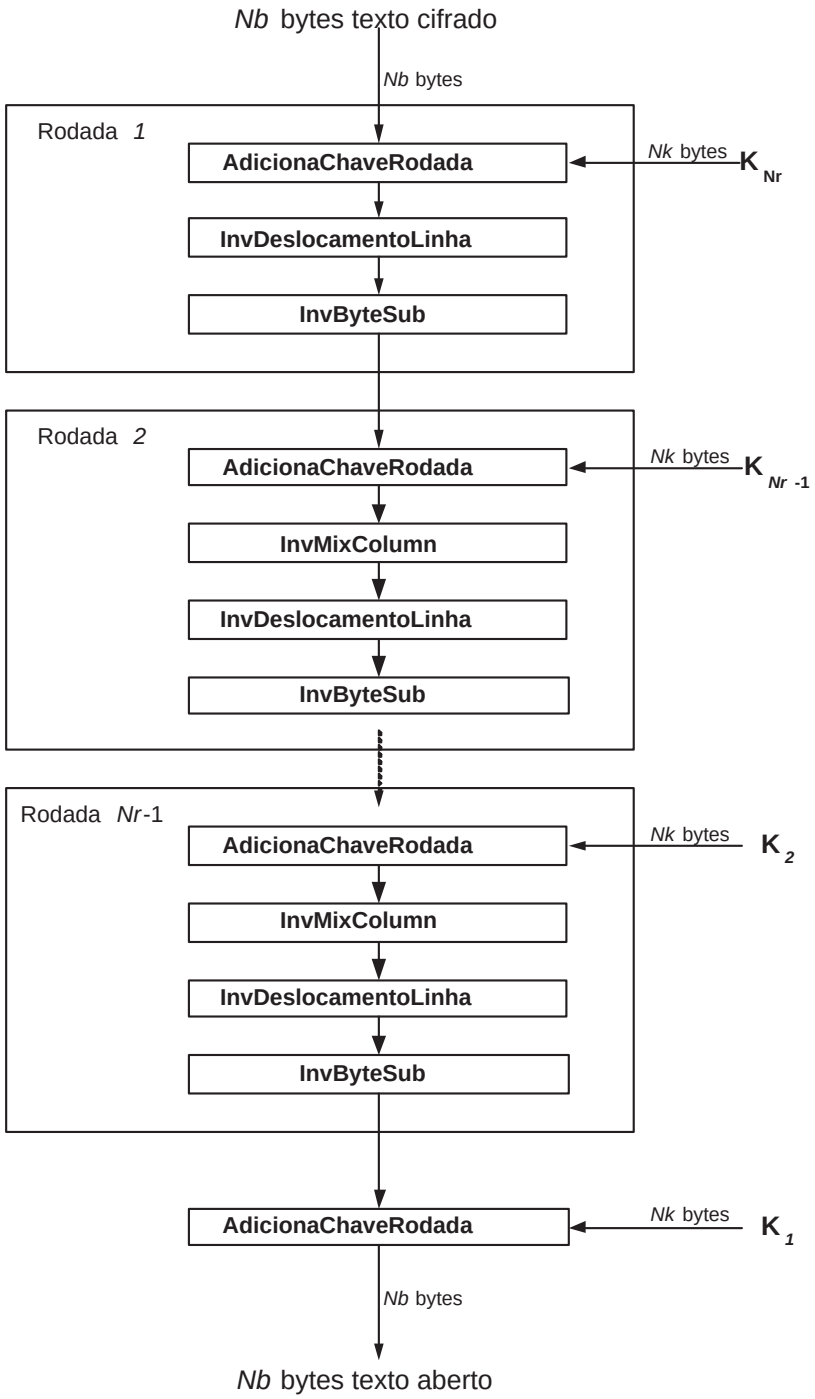

Figura 2.8: Fluxo para decifrar no AES

O Algoritmo para decifrar do AES pode ser especificado em pseudo-código da seguinte forma:

```
Decifrar(byte entrada[4*Nb],byte saida[4*Nb],palavra w[Nb*(Nr+1)])
  Inicio
    byte estado[4,Nb]
    estado = entrada
    AdicionaChaveRodada(estado, w+Nr*Nb)

    Para rodada = (Nr-1) ate 1 passo -1
      Inv_DeslocamentoLinha(estado)
      Inv_ByteSub(estado)
      AdicionaChaveRodada(estado, w + rodada * Nb)
      Inv_MixColumns(estado)
      rodada=rodada-1
    fim para

    Inv_DeslocamentoLinha(estado)
    Inv_ByteSub(estado)
    AdicionaChaveRodada(estado, w)
    saida = estado
  Fim
```

A operação inversa da DeslocamentoLinha é realizada fazendo-se um rotacionamento cíclico a direita na mesma quantidade de bytes da operação de cifrar. Visto que a transformação é feita a nível de bytes.

A operação inversa mais complexa é a MixColumn. Do mesmo modo que o processo de cifrar, na transformação InvMixColumn as colunas dos estado são considerados como polinômios sobre um corpo $GF(2^8)$ e sofrem uma multiplicação módulo $x^4 + 1$ com um polinômio fixo $a^{-1}(x)$, dado por:

$$a^{-1}(x) = 0Bhx^3 + 0Dhx^2 + 09hx + 0Eh$$

Da mesma forma que a operação MixColumn utilizada para cifrar, esta operação pode ser escrita como uma multiplicação de matrizes $s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0eh & 0bh & 0dh & 09h \\ 09h & 0eh & 0bh & 0dh \\ 0dh & 09h & 0eh & 0bh \\ 0bh & 0dh & 09h & 0eh \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Como resultado da multiplicação os 4 bytes da coluna são substituídos por:

$$s'_{0,c} = (0eh \bullet s_{0,c}) \oplus (0bh \bullet s_{1,c}) \oplus (0dh \bullet s_{2,c}) \oplus (09h \bullet s_{3,c})$$
$$s'_{1,c} = (09h \bullet s_{0,c}) \oplus (0eh \bullet s_{1,c}) \oplus (0bh \bullet s_{2,c}) \oplus (0dh \bullet s_{3,c})$$
$$s'_{2,c} = (0dh \bullet s_{0,c}) \oplus (09h \bullet s_{1,c}) \oplus (0eh \bullet s_{2,c}) \oplus (0bh \bullet s_{3,c})$$
$$s'_{3,c} = (0bh \bullet s_{0,c}) \oplus (0dh \bullet s_{1,c}) \oplus (09h \bullet s_{2,c}) \oplus (0eh \bullet s_{3,c})$$

A operação inversa da adição de chave de rodada é a mesma operação utilizada para o processo de cifrar. Isto ocorre devido a chave ser adicionada ao estado através de um *XOR* e essa operação ser reversível.

A operação inversa de ByteSub consiste no uso da inversa matemática da Caixa-S utilizada para cifrar. A Caixa-S invertida pode ser analisada na tabela 2.12.

Tabela 2.12: Caixa-S invertida do AES baseada em notação hexadecimal

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	OC	7D

Como pode ser observado a seqüencia de operações para cifrar e decifrar são diferentes, apesar do processo de expansão de chaves continuar sendo o mesmo para os dois processos. Porém através de algumas propriedades algébricas e matemáticas no algoritmo do AES pode-se obter o que se chama de Cifrador Equivalente Inverso, em outras palavras, que utilize a mesma seqüencia de operações utilizadas para cifrar (com as transformações substituídas pelas suas inversas). Contudo isso necessita que o processo de expansão de chaves seja alterado [FIP 01].

As duas propriedades que possibilitam isso:

- A ordem das transformações SubByte e DeslocamentoLinha podem ser alteradas sem que com isso o resultado seja alterado. Isso ocorre devido ao fato dessas duas operações alterarem apenas a posição do byte dentro do estado, não fazendo nenhuma alteração a nível de bit. Sendo que o mesmo acontece com as inversas dessas duas operações.

- As transformações MixColumn e InvMixColumn são lineares relativas a coluna de entrada, o que significa: InvMixcolumn(estado *XOR* ChaveRodada)= InvMixColumn(estado)*XOR* InvMixColumn(ChaveRodada)

Essas propriedades permitem que a ordem de InvSubytes e InvDeslocamentoLinha sejam invertidas. As transformações de AdicionaChaveRodada e InvMixColumn também podem ser invertidas, prevendo-se que as colunas no processo de expansão de chave serão alteradas por uma transformação InvMixColumn[16]. Sendo que essa operação não deve ser realizada nas primeiras e últimas palavras *Nb* no processo de expansão de chave, visto que essas duas subchaves são utilizadas em rodadas que não possuem a transformação InvMixColumn.

Como resultado dessas operações obtém-se o Cifrador Equivalente Inverso, que possui uma estrutura mais similar a estrutura original do cifrador do que o algoritmo decifrador proposto anteriormente. O Algoritmo para decifrar utilizando o Cifrador Equivalente Inverso do AES pode ser especificado em pseudo-código da seguinte forma:

[16]Essas modificações são válidas somente para o processo de decifrar

```
DecifrarEqInvCif(byte entrada[4*Nb],byte sa{\'\i}da[4*Nb],palavra dw[Nb*(Nr+1)])
  Inicio
    byte estado[4,Nb]
    estado = entrada
    AdicionaChaveRodada(estado, dw+Nr*Nb)

    Para rodada = (Nr-1) at{\'e} 1 passo -1
      InvByteSub(estado)
      InvDeslocamentoLinha(estado)
      InvMixColumns(estado)
      AdicionaChaveRodada(estado, dw + rodada * Nb)
      rodada=rodada-1
    fim para

    InvByteSub(estado)
    InvDeslocamentoLinha(estado)
    AdicionaChaveRodada(estado, w)
    saida = estado
  Fim
```

Deve ser observado que a palavra `dw` contém as chaves de rodada para decifrar modificadas. A rotina modificada para expansão de chave consiste na adição do algoritmo abaixo no final do algoritmo padrão de expansão de chave do AES :

```
Para i = 0 ate (Nr+1)*Nb-1 passo 1
    dw[i]=w{i]
fim para

Para rnd= 1 ate Nr-1 passo 1
    InvMixColumn(dw + rnd* Nb )
fim para
```

Exemplo. Processo de decifrar com *Nb*=4 e *Nk*=4.

Chave: 0a 1b 2c 3d 4e 5a 6b 7c 8d 9e aa ab ac ad ae af

Texto Cifrado: 19 37 cd d5 d0 2f b9 9c e9 02 cf fd aa 29 a7 c8

Nesse exemplo encontra-se descrito o processo de decifrar conforme o primeiro algoritmo descrito.

```
Rodada: 10
                          Entrada: 19 37 cd d5 d0 2f b9 9c e9 02 cf fd aa 29 a7 c8
                 Chave de Rodada: 1a fe 2f e9 1c 7d a8 63 53 b1 b1 3c da f3 79 bf
```

```
Rodada: 9
                    Estado Inicial: 03 c9 e2 3c cc 52 11 ff ba b3 7e c1 70 da de 77
   Apos Deslocamento Linha Inverso: 03 c9 e2 3c ff cc 52 11 7e c1 ba b3 da de 77 70
              Apos Caixa-S Inversa: d5 12 3b 6d 7d 27 48 e3 8a dd c0 4b 7a 9c 02 d0
                   Chave de Rodada: 33 e4 d1 c6 41 61 d5 cb e7 e2 00 8d 6e 29 8a c6
                 Apos Chave Rodada: d5 12 3b 6d 7d 27 48 e3 8a dd c0 4b 7a 9c 02 d0
           Apos MixColumns Inversa: 9f 75 70 e4 f2 a4 af ea 49 c8 24 2f 87 23 c4 72
Rodada: 8
                    Estado Inicial: 9f 75 70 e4 f2 a4 af ea 49 c8 24 2f 87 23 c4 72
   Apos Deslocamento Linha Inverso: 9f 75 70 e4 ea f2 a4 af 24 2f 49 c8 23 c4 72 87
              Apos Caixa-S Inversa: 6e 3f d0 ae bb 04 1d 1b a6 4e a4 b1 32 88 1e ea
                   Chave de Rodada: 5a d7 35 17 1c 20 b4 1e ce 05 e2 8d 9e 47 a3 4c
                 Apos Chave Rodada: 6e 3f d0 ae bb 04 1d 1b a6 4e a4 b1 32 88 1e ea
           Apos MixColumns Inversa: bb ef bf f5 d7 45 44 c0 49 dc 69 3c 72 3e 25 2f
Rodada: 7
                    Estado Inicial: bb ef bf f5 d7 45 44 c0 49 dc 69 3c 72 3e 25 2f
   Apos Deslocamento Linha Inverso: bb ef bf f5 c0 d7 45 44 69 3c 49 dc 3e 25 2f 72
              Apos Caixa-S Inversa: fe 61 f4 77 1f 0d 68 86 e4 6d a4 93 d1 c2 4e 1e
                   Chave de Rodada: 76 8d e2 22 b4 3c 94 aa 11 cb e7 6f 0d d9 e4 ef
                 Apos Chave Rodada: fe 61 f4 77 1f 0d 68 86 e4 6d a4 93 d1 c2 4e 1e
           Apos MixColumns Inversa: 7b 33 8d cc 8a e1 c1 37 12 b9 de 8d e9 0b 91 02
Rodada: 6
                    Estado Inicial: 7b 33 8d cc 8a e1 c1 37 12 b9 de 8d e9 0b 91 02
   Apos Deslocamento Linha Inverso: 7b 33 8d cc 37 8a e1 c1 de 8d 12 b9 0b 91 02 e9
              Apos Caixa-S Inversa: 03 66 b4 27 b2 cf e0 dd 9c b4 39 db 9e ac 6a eb
                   Chave de Rodada: 84 fb 6f c0 70 88 a8 3e 3a da 2c 88 b7 d4 3d 0b
                 Apos Chave Rodada: 03 66 b4 27 b2 cf e0 dd 9c b4 39 db 9e ac 6a eb
           Apos MixColumns Inversa: 49 25 f0 71 4b 1f 8d ec e7 55 87 b0 2f a3 2b 9a
Rodada: 5
                    Estado Inicial: 49 25 f0 71 4b 1f 8d ec e7 55 87 b0 2f a3 2b 9a
   Apos Deslocamento Linha Inverso: 49 25 f0 71 ec 4b 1f 8d 87 b0 e7 55 a3 2b 9a 2f
              Apos Caixa-S Inversa: a4 c2 17 2c 83 cc cb b4 ea fc b0 ed 71 0b 37 4e
                   Chave de Rodada: 34 7f 94 af 39 f8 20 96 3f e0 f6 a4 ce 63 e9 36
                 Apos Chave Rodada: a4 c2 17 2c 83 cc cb b4 ea fc b0 ed 71 0b 37 4e
           Apos MixColumns Inversa: bd 52 5b a3 06 5b 4f fb 78 12 65 52 83 e6 81 9a
Rodada: 4
                    Estado Inicial: bd 52 5b a3 06 5b 4f fb 78 12 65 52 83 e6 81 9a
   Apos Deslocamento Linha Inverso: bd 52 5b a3 fb 06 5b 4f 65 52 78 12 e6 81 9a 83
              Apos Caixa-S Inversa: cd 48 57 71 63 a5 57 92 bc 48 c1 39 f5 91 37 41
                   Chave de Rodada: 6a 4b eb 3b 39 c1 d8 b6 a1 df 16 52 2c ad 8a df
                 Apos Chave Rodada: cd 48 57 71 63 a5 57 92 bc 48 c1 39 f5 91 37 41
           Apos MixColumns Inversa: 72 5d ca 3d 04 14 50 4b ba 4a 13 31 f5 cf d0 dc
```

```
Rodada: 3
                    Estado Inicial: 72 5d ca 3d 04 14 50 4b ba 4a 13 31 f5 cf d0 dc
   Apos Deslocamento Linha Inverso: 72 5d ca 3d 4b 04 14 50 13 31 ba 4a cf d0 dc f5
              Apos Caixa-S Inversa: 1e 8d 10 8b cc 30 9b 6c 82 2e c0 5c 5f 60 93 77
                   Chave de Rodada: fd 21 a0 d0 22 f8 19 6e 5d 7e c9 44 5c 81 27 55
                Apos Chave Rodada: 1e 8d 10 8b cc 30 9b 6c 82 2e c0 5c 5f 60 93 77
           Apos MixColumns Inversa: 0a 9e 9d a8 16 60 cc e0 8f d3 21 2d 42 f8 ff 06
Rodada: 2
                    Estado Inicial: 0a 9e 9d a8 16 60 cc e0 8f d3 21 2d 42 f8 ff 06
   Apos Deslocamento Linha Inverso: 0a 9e 9d a8 e0 16 60 cc 21 2d 8f d3 f8 ff 06 42
              Apos Caixa-S Inversa: a3 df 75 6f a0 ff 90 27 7b fa 73 a9 e1 7d a5 f6
                   Chave de Rodada: 0c dc 81 70 7f da e1 77 1d 23 b7 8d 0d dd a6 72
                Apos Chave Rodada: a3 df 75 6f a0 ff 90 27 7b fa 73 a9 e1 7d a5 f6
           Apos MixColumns Inversa: 2d 9b 41 bb 11 75 b7 13 0b ff f5 45 cd 4e 41 02
Rodada: 1
                    Estado Inicial: 2d 9b 41 bb 11 75 b7 13 0b ff f5 45 cd 4e 41 02
   Apos Deslocamento Linha Inverso: 2d 9b 41 bb 13 11 75 b7 f5 45 0b ff 4e 41 02 cd
              Apos Caixa-S Inversa: fa e8 f8 fe 82 e3 3f 20 77 68 9e 7d b6 f8 6a 80
                   Chave de Rodada: 9e d0 5d f1 ff a5 3b 96 55 3e 94 3a ac d0 7b d4
                Apos Chave Rodada: fa e8 f8 fe 82 e3 3f 20 77 68 9e 7d b6 f8 6a 80
           Apos MixColumns Inversa: 67 67 6b d0 76 c5 51 67 63 09 ab d7 53 ab 2b ca
Rodada: 0
                    Estado Inicial : 67 67 6b d0 76 c5 51 67 63 09 ab d7 53 ab 2b ca
Apos Deslocamento de Linha Inverso: 67 67 6b d0 67 76 c5 51 ab d7 63 09 ab 2b ca 53
              Apos Caixa-S Inversa: 0a 0a 05 60 0a 0f 07 70 0e 0d 00 40 0e 0b 10 50
                   Chave de Rodada: 0a 4e 8d ac 1b 5a 9e ad 2c 6b aa ae 3d 7c ab af

                      Texto Aberto: 00 44 88 cc 11 55 99 dd 22 66 aa ee 33 77 bb ff
```

2.10 Ataques ao AES

O AES é um cifrador orientado a byte baseado na estrutura do cifrador Square. Junto com a proposição do cifrador AES foi colocado o Ataque Square como uma referência para um ponto de partida de outras análises. Os tipos de operações de substituição e permutação utilizadas no cifrador são padrão, logo ataques em estrutura básicas não são aplicáveis. A Caixa-S possui um estrutura matemática, baseada na combinação da inversão de um Corpo de Galois (GF) e uma transformação consistindo de uma multiplicação por matriz seguida pela adição de um vetor. Embora o uso de toda essa estrutura matemática concebivelmente ajude um ataque, a estrutura não é obscura como

seria o caso para um *trapdoor*. A especificação do Rijndael afirma que caso a Caixa-S fosse suspeita de conter um *trapdoor*, então a Caixa-S poderia ser substituída[RIJ 00].

Como já foi mencionado, a especificação do AES [JD 99a] especifica um ataque diferencial truncado em variações do cifrador com 4, 5 e 6 rodadas apenas, esse ataque é baseado no ataque do cifrador Square e é conhecido como Ataque Square. Em ”A Collision Attack on 7 Rounds of Rijndael”[HG 00] ataques diferenciais truncados são utilizados para construir um tipo de diferenciador em 4 rodadas, baseado, na existência experimentalmente confirmada de colisões entre algumas funções parciais induzidas pelo cifrador. Este diferenciador gerado permitiu a implementação de uma ataque em uma variante reduzida do Rijndael com 7 rodadas apenas.

Vários dos ataques ao AES foram construídos diretamente sobre variações do Ataque Square. Um Ataque deste tipo é descrito no artigo ”Improved Cryptanalysis of Rijndael”[NF 00]. Porém esses ataques são melhorados através de uma técnica de soma parcial que reduz o número de operações. A técnica de soma parcial também pode ser combinada com a técnica de operações de informação *trading off*, permitindo ataques em variantes do Rijndael com 7 e 8 rodadas requerendo quase todas as codificações. Nesse mesmo artigo é apresentado um ataque de chave baseado em um variante reduzida de 9 rodadas com chaves de 256 bits. Esse ataque não requer apenas cifragem de texto aberto escolhido com a chave, mas também cifrados com outras 255 chaves que são relacionadas, até certo ponto, à chave escolhida pelo adversário. Um ponto interessante sugerido nesse trabalho está na discussão que algumas propriedades do cifrador e sugere que, ao contrário do que é afirmado na especificação do cifrador Rijndael, o processo para geração de chaves não possui uma grande característica de difusão.

Sean Murphy e Matt Robshaw [SM 00], citam que algumas propriedades no aspecto linear de parte da rodada que podem ser exploradas. Especificamente um mapeamento linear dentro de uma função da rodada onde possui a propriedade que em 16 iterações são equivalentes ao mapeamento identidade, com isso, lançam dúvida sobre o grau de difusão atingido em várias rodadas. Em resposta ao artigo os submissores do Rijndael argumentaram que os fatores expostos não influenciam na segurança do Rijndael [JD 00].

Na realidade, a maioria destes ataques em variantes com a quantidade de rodadas reduzidas é mais difícil de executar em prática que ataques por procura exaustiva da chave[17], apesar da exigência de processamento menor, por causa dos requisitos de informações e memória necessárias aos ataques. Além disso, até mesmo se um ataque em uma variante simplificada fosse prático, o algoritmo original poderia permanecer seguro.

Se apenas uma pequena simplificação permitisse o ataque ao algoritmo isso poderia ser encarado como um problema, mas os ataques só conseguem realizar um efeito perceptível após grandes simplificações do cifrador original o que pode ser uma indicação de que o original possui uma boa margem de segurança. A simplificação inclui a redução da quantidade de rodadas, o que não é nada surpreendente, por que muito dos aspectos em que os ataques são elaborados baseiam-se em alguma forma de criptoanálise linear e diferencial que quando o cifrador AES é executado em sua quantidade normal de rodadas inviabilizam os ataques. O número total de rodadas especificado para o algoritmo pode ser comparado com o número máximo de rodadas existente no ataque, que produz uma medida definida pelo NIST como fator de segurança.

O cifrador AES aparenta possuir uma margem adequada de segurança. Essa margem de segurança porém é de difícil análise devido ao fato do número de rodadas variar de acordo com o tamanho da chave. O cifrador recebeu suas maiores críticas em dois aspectos: que a sua margem de segurança foi considerada a menor entre todos os finalistas do processo para a seleção do AES, e que a estrutura essencialmente matemática poderia permitir algum tipo de ataque ou facilitador futuro. Porém o cifrador recebeu uma análise bem elaborada, dentro do período de tempo disponibilizado durante o processo de seleção, devido a sua estrutura simples.

2.11 Conclusão

Neste capítulo foram estudados e analisados os fundamentos matemáticos e de álgebra necessários à interpretação do cifrador AES, assim como o processo para cifrar, decifrar e geração de subchaves. Divergindo do padrão anterior, o DES, o AES não

[17] Força Bruta

utiliza-se de Estruturas de Feistel e sim de operações matemáticas sobre Corpos de Galois (GF) como recurso para cifrar. As operações são simples e permitem uma visão clara do funcionamento do cifrador.

Diferente de outros cifradores de bloco simétricos o AES possui uma quantidade de rodadas variáveis em função da quantidade de blocos de entrada e tamanho de chave. Também difere no fato do processo para decifrar ser diferente do processo para cifrar em função da necessidade de inversão das funções matemáticas utilizadas.

A criptoanálise do AES foi estudada através de versões com a quantidade de rodadas reduzidas baseadas, em geral, no Ataque Square. Essas versões reduzidas não objetivam o aprendizado do AES o que fornece precedentes para a elaboração de um modelo simplificado com esse objetivo.

Capítulo 3

SAES

3.1 Introdução

Neste capítulo serão descritas as etapas e pontos chaves do desenvolvimento do modelo simplificado do cifrador de bloco simétrico AES (SAES). Este capítulo é composto de objetivos, uma descrição da proposta original sugerida no TI[1], etapas do desenvolvimento, modelo final e exemplos de utilização do SAES, feitas a partir da implementação didática do SAES.

3.2 Objetivos

O cifrador AES, embora possua uma estrutura relativamente simples, é de difícil implementação manual devido a quantidade de rodadas, tamanho de chave / bloco e a transformação MixColumn sobre $GF(2^8)$ que é demorada e complexa de ser realizada manualmente. Esses pontos podem desencorajar o aprofundamento no estudo do cifrador AES em fases finais de cursos de graduação e cursos de pós-graduação e pode ser resolvido através do uso de um modelo simplificado. Devido ao fato do AES utilizar conceitos de matemática e álgebra, o desenvolvimento da versão simplificada exige maior estudo desses conceitos, o que torna necessário uma introdução nessas áreas.

[1]Trabalho Individual: Proposta de Modelo Simplificado para o Cifrador AES [MIE 02]

Deste modo definiu-se que o SAES deve atender os seguintes requisitos:

1. Manter as características funcionais do AES com os parâmetros reduzidos;
2. Permitir a fácil compreensão do AES completo, uma vez definidas a relação entre SAES e AES;
3. Permitir a implementação manual dos processos: cifrar / decifrar e geração de chaves;
4. Servir como referência para a proposição de outros modelos simplificados de cifradores;
5. Facilitar a apresentação didática do cifrador AES;
6. Possibilitar a utilização como referência para o aprendizado do cifrador AES.

3.3 Metodologia Utilizada

A partir do estudo realizado no Trabalho Individual [MIE 02], que antecedeu a concepção do modelo simplificado, foram definidas as etapas e metodologia para o desenvolvimento do SAES. Desta forma cada um dos requisitos especificados é estudado mais profundamente. Os principais pontos trabalhados foram:

1. Possuir o mínimo de rodadas possíveis;
2. Redefinir o tamanho da matriz estado;
3. Reduzir o tamanho de $GF(2^8)$;
4. Reduzir o tamanho de bloco de entrada de acordo com a redução de GF;
5. Padronizar os polinômios irredutíveis e outros elementos da versão normal do AES ao novo GF, incluindo a Caixa-S;
6. Reduzir o tamanho da chave e adequar os algoritmos de geração de chaves para o novo tamanho das mesmas;

7. Criar exemplos de cifrar, decifrar e geração de chaves utilizando o SAES que possam ser usados como modelos em sala de aula;

8. Desenvolver uma implementação do SAES na Linguagem C padrão ANSI[2] para ser utilizada como referência para o aprendizado do SAES.

3.3.1 Redução do Número de Rodadas

Conforme demonstrado no Capítulo 2 através da tabela 2.5 o AES possui um número de rodadas variável de acordo com o tamanho de chaves e bloco utilizados. O acréscimo de rodadas decorrentes do aumento do tamanho da chave ou bloco de entrada tem por objetivo manter o nível de difusão durante o processo de cifrar. No SAES tem-se como um dos objetivos reduzir a quantidade de rodadas o máximo possível sem desvincular das características de funcionamento do AES, porém devido ao seu aspecto didático não é prioritário manter graus de difusão elevado para dificultar a criptoanálise.

Uma forma de diminuir o número de rodadas é eliminar as operações repetitivas e seqüenciais. Analisando o fluxo de cifrar do AES através da figura 3.1 pode-se perceber que a única operação redundante e seqüencial identificada na estrutura macro do AES está indicada dentro do retângulo identificado com ”REPETIÇÕES”.

Continuando a análise da figura 3.1 pode-se perceber que a estrutura principal do cifrador consiste em:

1. AdicionaChaveRodada;

2. Rodadas convencionais(ByteSub, DeslocamentoLinha, MixColumn, AdicionaChaveRodada);

3. Rodada final (ByteSub, DeslocamentoLinha, AdicionaChaveRodada) que difere de uma rodada convencional por não possuir a transformação MixColumn.

Deste modo fica claro que a redução da quantidade de rodadas convencionais de uma quantidade variável (9,11 e 13 rodadas) que possui o AES para uma rodada

[2] American National Standards Institute

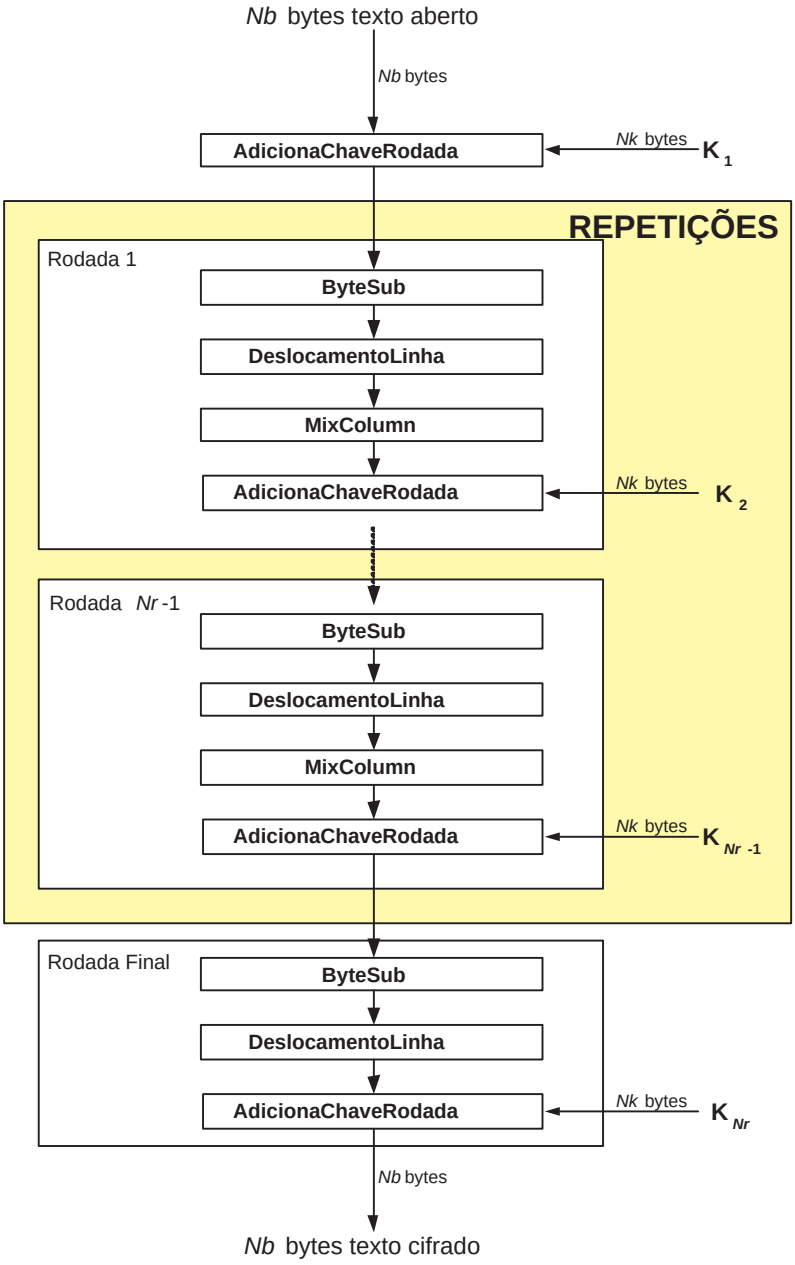

Figura 3.1: Operações repetitivas e seqüenciais na estrutura principal do AES

apenas simplifica a estrutura do cifrador a nível macro. Além da simplificação a nível macro também indica a necessidade de três subchaves de rodadas apenas, em decorrência da retirada das rodadas convencionais extras. Desse modo a figura 3.2 mostra a estrutura macro para o processo de cifrar do SAES derivada dessa redução, sendo indicada pela área cinza a alteração feita em relação ao AES.

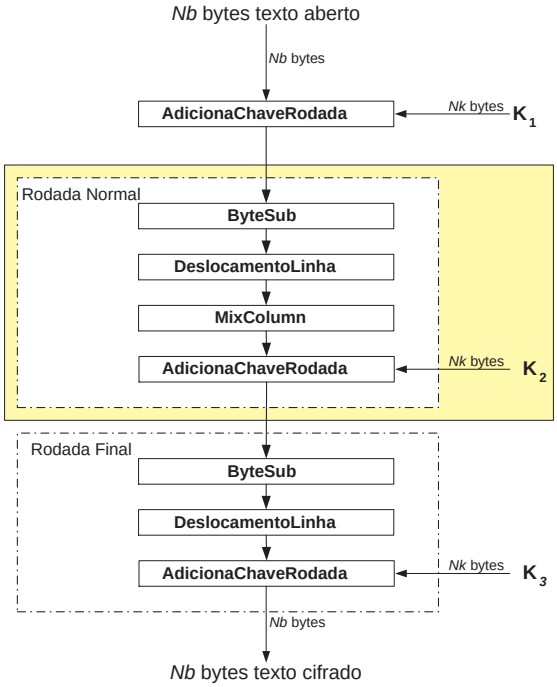

Figura 3.2: Fluxo macro do processo de cifrar com o SAES

3.3.2 Redefinição do Tamanho da Matriz Estado

O AES realiza todas as suas transformações com base na matriz estado[3]. O estado padrão do AES é formado por colunas com quatro elementos (a quantidade de colunas indica o tamanho da chave ou bloco de entrada), essas colunas são chamadas de *Nb* quando referem-se ao bloco de entrada e *Nk* quando referem-se ao tamanho da chave. Sobre a matriz estado é aplicada uma transformação que produz como resultado um novo estado. Desse modo o Novo Estado na realidade é o Estado Inicial da transformação seguinte. Na figura 3.3 pode-se observar um matriz estado com *Nb*=4 e o conceito de transformação sobre o estado no AES.

O AES é um cifrador orientado ao elemento do estado, na figura 3.3 um elemento do estado é representado por $S_{0,0}$, sendo que as transformações AdicionaChave-Rodada, DeslocamentoLinha e ByteSub são feitas com base nos elementos e geram com resultado um elemento $S'_{0,0}$. Desse modo a redução do tamanho da Matriz Estado deve

[3]Conforme explicado no item 2.5 dessa dissertação

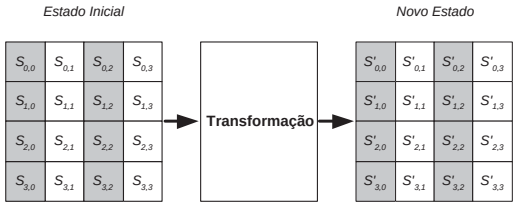

Figura 3.3: Exemplo de transformação sobre a matriz estado no AES

levar em conta as transformações e operações que são realizadas sobre ela. Dessa forma devem ser analisadas as transformações a fim de verificar o número mínimo possível de colunas e elementos por coluna, a transformação MixColumn é a única a não operar sobre um elemento mas sim sobre uma coluna do estado. As transformações podem ser analisadas em um nível macro na figura 3.4.

Figura 3.4: Transformações AdicionaChaveRodada, ByteSub, DeslocamentoLinha e MixColumn sobre uma matriz estado no AES com *Nb*=4 e *Nk*=4

Com base na figura 3.4 que demonstra as transformações sobre uma matriz estado do AES com *Nb*=4 [4] e *Nk*=4 [5] e na definição de estado mostrada na figura 3.3 verifica-se que o estado referido possui 16 elementos agrupados em 4 colunas de 4 elementos ou uma matriz de 4 *x* 4 elementos. O tamanho da Matriz Estado no SAES deve continuar a possuir as mesmas características de funcionamento conforme especificado nos objetivos.

Um modo simples de avaliar o número mínimo de elementos é através do teste progressivo da quantidade de elementos que compõem a Matriz Estado, verificando se mantém as características originais do AES. Porém deve-se observar que a quantidade de elementos por coluna (em outras palavras: linhas) deverá ser igual a fim de manter a característica original da Matriz Estado do AES.

Uma primeira idéia seria a de construir o SAES com 1 elemento apenas, isto é: a Matriz Estado será de dimensão 1x1 elementos. Dessa forma poderia-se avaliar as transformações em um nível macro na figura 3.5 a fim de verificar se mantém as características originais do AES.

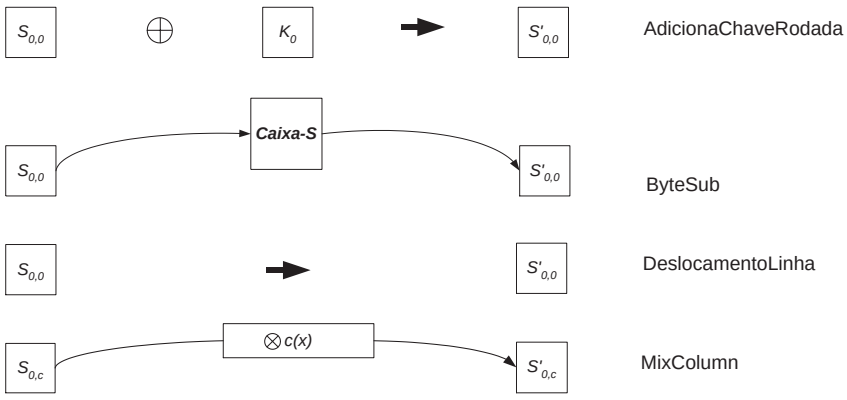

Figura 3.5: Transformações AdicionaChaveRodada, ByteSub, DeslocamentoLinha e MixColumn sobre uma matriz estado no SAES com 1 elemento

A tabela 3.1 mostra quais as transformações que possuem suas características mantidas em relação ao AES. A transformação AdicionaChaveRodada por ser

[4] *Nb*=4 é o tamanho mínimo para *Nb* homologado pela FIPS197 [FIP 01]

[5] *Nk*=4 é o tamanho mínimo para *Nk* homologado pela FIPS197 [FIP 01]

um simples *XOR* do estado com a subchave de rodada não apresenta nenhuma descaracterização em relação ao AES. As transformações ByteSub e MixColumn, embora fiquem muito simplificadas, também mantém as suas características originais por serem simples transformações de um elemento no caso da ByteSub e uma coluna no caso da MixColumn. A transformação DeslocamentoLinha não mantém as características do AES devido a linha possuir apenas um elemento, o que significa uma linha apenas, o que impossibilita qualquer rotação de elemento, as especificações do AES indicam o rotacionamento dos elementos por linha do Estado.

Tabela 3.1: Resultados de uma Matriz Estado com 1 elemento mantém as características macro das transformações do AES

Transformação	Mantém as Característica do AES
AdicionaChaveRodada	Sim
ByteSub	Sim
DeslocamentoLinha	Não
MixColumn	Sim

Através do fator analisado de que a Matriz Estado com a quantidade de um elemento não mantém as características da transformação por possuir apenas uma linha e conseqüentemente um elemento por linha exclui-se automaticamente a possibilidade da Matriz Estado estado possuir dois elementos organizados na disposição 2 *x* 1 elementos[6].

Uma outra disposição possível para a Matriz Estado com dois elementos seria a disposição de 1 *x* 2 elementos. Essa disposição possibilitaria até o deslocamento de elementos da linha para atender uma das premissas da transformação DeslocamentoLinha, porém uma característica dessa transformação que pretende-se manter no SAES é a de que uma linha não sofrerá rotação e uma linha sofrerá rotação. Sendo essa uma premissa definida fica excluída a hipótese da Matriz Estado possuir dois elementos.

Com base nas premissas que impossibilitam o uso de uma Matriz Estado com a quantidade de 1 e 2 elementos pode-se definir um padrão de formato, que independente da quantidade de elementos, não satisfaz os requisitos para a transformação

[6]Linha *x* Coluna

DeslocamentoLinha. Esse padrão de formato ou regra pode ser definido pelas dimensões 1 *x* X e X *x* 1, aonde X é qualquer quantidade de elementos maior que 1.

A próxima quantidade de elementos a ser avaliada seria 3, porém uma Matriz Estado com três elementos pode ser dispostas somente de duas formas: 3 *x* 1 e 1 *x* 3. Verificando o formato da matriz com a regra definida anteriormente pode-se afirmar a inviabilidade de uma Matriz Estado com três elementos.

Uma Matriz Estado com a quantidade de 4 elementos pode ser organizada de três formas: 4 *x* 1, 1 *x* 4 e 2 *x* 2 elementos. Os formatos 4 *x* 1 e 1 *x* 4 ficam eliminados automaticamente pela regra definida anteriormente, porém o formato 2 *x* 2 satisfaz as premissas definidas anteriormente para a transformação DeslocamentoLinha. Desse modo pode-se avaliar uma Matriz Estado com a dimensão de 2 *x* 2 elementos na figura 3.6.

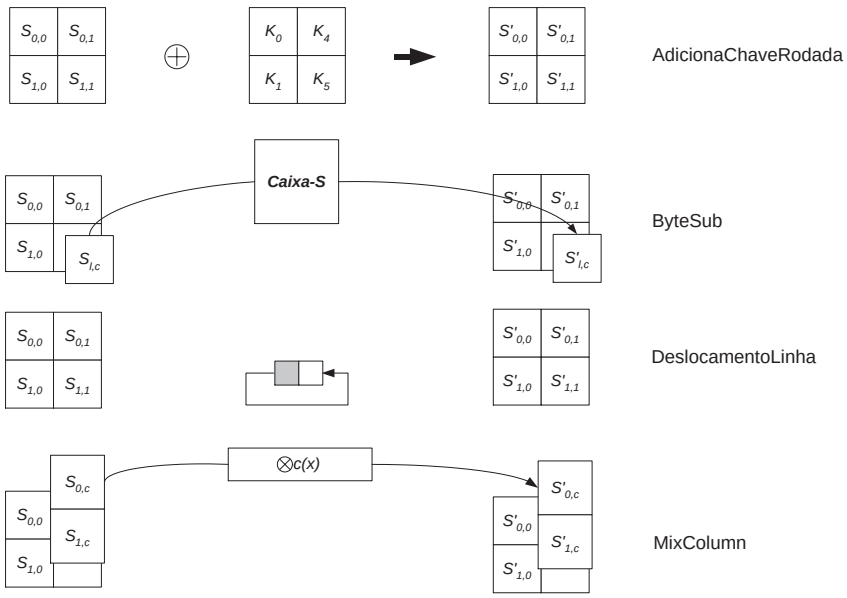

Figura 3.6: Transformações AdicionaChaveRodada, ByteSub, DeslocamentoLinha e MixColumn sobre uma matriz estado no SAES com 4 elementos

Analisando a visão macro das transformações mostradas na figura 3.6 pode-se verificar que satisfaz as necessidades até o momento. Desse modo os resultados de que mantém as características do AES em nível macro através da tabela 3.2.

Tabela 3.2: Resultados se uma Matriz Estado com 4 elementos, 2 *x* 2, mantém as características macro das transformações do AES

Transformação	Mantém as Característica do AES
AdicionaChaveRodada	Sim
ByteSub	Sim
DeslocamentoLinha	Sim
MixColumn	Sim

3.3.3 Redução do Tamanho do Corpo Finito $GF(2^8)$

O tamanho do corpo finito GF[7] é um fator determinante no cifrador AES devido ao grau de complexidade na execução da operação de multiplicação[8] sobre o corpo finito GF. Dentro do conceito de Matriz Estado no cifrador AES cada elemento dessa matriz é um byte, mais precisamente um elemento dentro de um corpo finito $GF(2^8)$. Desta forma a redução do corpo finito deve atender as características do cifrador AES e possibilitar o rápido entendimento de como realizar multiplicações sobre corpos GF.

O AES é dito um cifrador orientado ao byte devido ao fato de cada bit de um byte ser considerado o coeficiente multiplicativo de um polinômio de grau 8 utilizado para representar um elemento no corpo finito $GF(2^8)$. Uma representação de um byte no AES pode ser dada por:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

O tamanho mínimo para o expoente n de um corpo $GF(2^n)$ é 1 o que resulta no corpo finito $GF(2^1)$, ou simplesmente $GF(2)$. Desta forma cada elemento da Matriz Estado é representado por um bit demonstrado por:

$$b_0$$

Porém um elemento simples como um bit não possibilitaria a demonstração de multiplicações que mantivessem as características do AES, sendo dessa forma o corpo finito $GF(2^1)$ considerado inadequado para o SAES.

[7] O conceito de corpo finito GF é descrito no item 2.4.1

[8] O conceito de multiplicação sobre um corpo finito GF é descrito no item 2.4.2.1

O próximo coeficiente n a ser adotado para o corpo $GF(2^n)$ é 2, deste tem-se o corpo finito $GF(2^2)$. Com um corpo $GF(2^2)$ tem-se como cada elemento da Matriz Estado 2 bits representado por:

$$b_1x^1 + b_0$$

A representação genérica do elemento $b_1x^1 + b_0$ para o corpo $GF(2^2)$ pode ser utilizada para demonstrar o funcionamento de operações sobre corpos finitos. Um corpo $GF(2^2)$ é utilizado para explicar operações sobre corpos finitos, pode-se citar o exemplo de Wade Trappe e Laurence C. Washington no livro *Introduction to Cryptography with Code and Theory* [WAS 01]. Observa-se desta forma que a redução do corpo finito $GF(2^8)$ para o corpo $GF(2^2)$ satisfaz os requisitos para ser utilizado no SAES.

Além de manter as características macros é interessante construir as tabelas de resultados para as operações básicas de adição (+) e multiplicação (•) sobre o corpo. A importância das tabelas de resultados está nas premissas matemáticas para grupos descritas no item 2.3.1 e em possibilitar uma visão exemplificada dos elementos que compõe o corpo. Na tabelas 3.3 pode-se observar as operações e seus resultados, os elementos do conjunto $GF(2^2)$ colocados no exemplo foram $\{0, 1, 2, 3\}$.

Tabela 3.3: Tabelas de resultados das operações + e • para o corpo $GF(2^2)$

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

•	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Uma vez definida a representação polinomial para o elemento como $b_1x^1 + b_0$ pode-se construir as tabelas de resultados reais com o corpo finito $GF(2^2)$ que serão utilizadas no SAES. O resultado obtido pode ser observado através da tabela 3.4.

O AES utiliza-se da notação polinomial para representar os bits de um elemento do estado em um corpo finito GF. A finalidade real do cifrador é cifrar dados em bits, logo é interessante especificar as mesmas tabelas de resultados mostradas na

Tabela 3.4 em notação binária a fim de facilitar o uso do cifrador SAES. A tabela de resultados em notação binária poder ser observada na tabela 3.5.

Tabela 3.4: Tabelas de resultados das operações + e • para o corpo $GF(2^2)$ usando a representação polinomial $b_1x^1 + b_0$

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

$\bullet$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Tabela 3.5: Tabelas de resultados das operações + e • para o corpo $GF(2^2)$ usando a representação binária

$+$	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

$\bullet$	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

3.3.4 Redução do Bloco de Entrada/Chave

A redução do bloco de entrada é uma consequência direta do tamanho da Matriz Estado com o tamanho do elemento. Desse modo o tamanho do bloco de entrada em bits pode ser obtido através da seguinte fórmula:

Tamanho Bloco de Entrada = Quantidade Elementos Matriz Estado x bits Elemento

Tamanho Bloco de Entrada = 4 x 2bits

Tamanho Bloco de Entrada = 8 bits

Conforme pode ser observado na transformação AdicionaChaveRodada do AES através da Figura 3.4 e do SAES na Figura 3.6 verifica-se que o tamanho da chave

é idêntico ao do bloco de entrada. Deve-se observar que essa comparação quanto ao AES não é de todo verdadeira, pois na tabela 2.5 pode-se verificar que para o AES o tamanho da chave é igual ou maior de acordo com especificado na tabela.

Deseja-se que o SAES tenha o menor tamanho de chave possível sendo que o tamanho mínimo de chave para o AES é igual ao tamanho mínimo de bloco utilizado pelo cifrador. Sendo o SAES inicialmente projetado para trabalhar com bloco de entrada com o tamanho mínimo de 8 bits a chave possuirá o mesmo tamanho, isto é: 8 bits. A figura 3.7 exemplifica a alocação dos bits dentro do estado padronizado para o SAES.

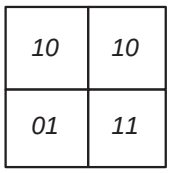

Figura 3.7: Exemplo da representação de estado do SAES para a Entrada '10011011'

3.3.5 Padronização de Polinômios Irredutíveis

A representação polinomial de um elemento através da notação $b_1x^1+b_0$ significa que quando uma multiplicação é feita entre dois elementos o resultado obtido estará fora do corpo por que possuíra um elemento x^2 que não existe no corpo $GF(2^2)$, conforme mostrado no item 3.3.3. A necessidade de polinômios irredutíveis está vinculada a multiplicação de elementos de um corpo finito para que continuem dentro do espaço do corpo[9] GF. A multiplicação de elementos em um corpo finito é indicada pelo operador $\bullet$ porém para que o resultado da operação continue no corpo $GF(2^2)$ é executada a operação de multiplicação módulo.

Para cada corpo $GF(2^n)$ existem um ou mais polinômios irredutíveis dependendo do tamanho do corpo. Com base no trabalho "Performance and Security of Block Ciphers using Operations in $GF(2^n)$"escrito por Shiho Moriai e Takeshi Shimoyama [JAP 01] que estuda o desempenho de várias operações algébricas sobre corpos

[9] A explicação de polinômios irredutíveis encontra-se descrita no item 2.4.1.2

GF com base 2 elevados a valores variando entre 2 e 69, foi adotado o polinômio irredutível $x^2 + x + 1$. A definição da adoção do corpo $GF(2^2)$ com o polinômio irredutível $x^2 + x + 1$ também é demonstrada no livro "Introduction to Cryptography with Coding Theory "de Wade Trappe e Lawrence C. Washington [WAS 01]. No corpo $GF(2^2)$ existem os seguintes polinômios possíveis:

x^2

$x^2 + 1$

$x^2 + x$

$x^2 + x + 1$

Verificando os polinômios observa-se:

- O polinômio x^2 não é um polinômio irredutível por que $x^2 = x * x$, isto é: pode ser decomposto;
- O polinômio $x^2 + x$ não é um polinômio irredutível por poder ser decomposto, em $(x + 1) * (x)$;
- O polinômio $x^2 + 1$ não é um polinômio irredutível por poder ser decomposto em $(x) * (x + 1)$
- O polinômio $x^2 + x + 1$ é um polinômio irredutível por não poder ser decomposto.

Com base nas observações acima adotou-se o polinômio $x^2 + x + 1$ como polinômio irredutível para o SAES em operações de multiplicação mod 2 dentro do corpo finito $GF(2^2)$.

3.3.5.1 Padronização de Outros Polinômios

Outra definição necessária para o SAES consiste na operação de multiplicação de polinômios com coeficientes em $GF(2^2)$. Uma definição de um polinômio deste tipo pode ser dada por:

$$a(x) = a_1 x + a_0$$

Para ilustrar as operações de multiplicação e adição de polinômios com coeficientes em $GF(2^2)$ define-se $b(x)$ como o segundo polinômio:

$$b(x) = b_1 x + b_0$$

A operação de adição é realizada somando os valores das bases com o mesmo coeficiente, e a realização de um *XOR* com as bases de mesmo coeficiente. Demonstrando a operação com $a(x) + b(x)$:

$$a(x) + b(x) = (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

A operação de multiplicação é realizada em duas fases. Na primeira fase o produto $c(x) = a(x) \otimes b(x)$ da multiplicação polinomial é expandindo algebricamente:

$$c(x) = c_2 x^2 + c_1 x + c_0$$

com

$c_0 = a_o \bullet b_0$

$c_1 = a_1 \bullet b_0 \oplus b_1 \bullet a_0$

$c_2 = a_1 \bullet b_1$

Pode-se verificar que o resultado de $c(x)$ não representa um polinômio de grau 2. A segunda fase consiste em reduzir $c(x)$ através de uma multiplicação mod com um polinômio de grau 2. No SAES adotou-se, por similaridade ao AES[10], o polinômio $x^2 + 1$ desta forma expressado por:

$$x^i \ mod \ (x^2 + 1) = x^{i \ mod \ 2}$$

O produto modular de $a(x)$ e $b(x)$, denotado por $a(x) \otimes b(x)$, é dado pelo polinômio de dois termos $d(x)$:

$$d(x) = d_1 x + d_0$$

[10]O polinômio utilizado no AES é $x^4 + 1$

com:

$d_0 = (a_o \bullet b_0) \oplus (a_1 \bullet b_1)$

$d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1)$

Do mesmo modo que no AES, quando $a(x)$ é um polinômio fixo, a operação $d(x) = d_1 x + d_0$ pode ser escrita na seguinte forma matricial:

$$\begin{bmatrix} d_1 \\ d_0 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

Devido ao fato de x^2+1 não ser um polinômio irredutível sobre $GF(2^2)$, conforme já demonstrado anteriormente. A multiplicação por um polinômio fixo de grau dois não é necessariamente inversível. Do mesmo modo que no AES, foi adotado um polinômio de grau 2 que possui uma inversa. Na documentação original da especificação do Rijndael [JD 99a] e na especificação do AES [FIP 01] não consta como foi obtido/selecionado esse polinômio e o seu inverso. Devido ao fato do SAES necessitar também de um polinômio inversível e esse ser diferente do empregado do AES[11] tornou-se necessário um estudo para determinar como gerar e identificar um polinômio inversível.

Analisando o funcionamento matemático do polinômio inversível e as premissas de um Corpo GF constata-se que uma das características é de que a multiplicação $a \bullet b$ para ser inversível deve ter seu resultado igual a matriz identidade, o que é expressado por $a(x) \bullet b(x)$ sendo $a_i, b_i \ \in \ GF(2^2)$:

$$\begin{bmatrix} a_0 & a_1 \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 & b_1 \\ b_1 & b_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Realizando a multiplicação acima e simplificando-a obtém-se:

$(a_0 \bullet b_1) = 0$, ou de outra forma: $a_0 \bullet b_1$

[11] em função do tamanho do corpo, matriz estado e tamanho do elemento

Outra característica que pode ser observada de relevância para o processo de decifrar, aonde S representa o elemento do estado:

$$S = a \bullet b \Longrightarrow a^{-1} \bullet S \Longrightarrow Identidade \bullet b \Longrightarrow a^{-1} \bullet S = b$$

Realizando a operação obtém-se:

$$(a_0 \bullet b_0) \oplus (a_1 \bullet b_1) = 1$$

Analisando as informações acima chega-se a conclusão que para um polinômio $a(x)$ possua uma inversa sobre uma multiplicação $a \bullet b$ ele deve satisfazer duas premissas:

$$\begin{cases} (a_0 \bullet b_0) \oplus (a_1 \bullet b_1) = 1 \\ (a_1 \bullet b_0) = 0 \end{cases}$$

Baseado nessas duas equações acima foram testados todos os valores possíveis, dentro do corpo finito $GF(2^2)$, para os coeficientes multiplicativos com o objetivo de encontrar um par de polinômios inversíveis. Nas tabelas abaixo estão contidos os resultados das duas equações citadas anteriormente em um par x, y aonde x corresponde a equação $(a_0 \bullet b_0) \oplus (a_1 \bullet b_1)$ e y a equação $(a_0 \bullet b_1)$, sendo que o resultado $(1, 0)$ identifica os polinômios inversíveis.

$a(0,0) * b(0,0) = (0,0)$	$a(0,1) * b(0,0) = (0,0)$	$a(0,2) * b(0,0) = (0,0)$	$a(0,3) * b(0,0) = (0,0)$
$a(0,0) * b(0,1) = (0,0)$	$a(0,1) * b(0,1) = (1,0)$	$a(0,2) * b(0,1) = (2,0)$	$a(0,3) * b(0,1) = (3,0)$
$a(0,0) * b(0,2) = (0,0)$	$a(0,1) * b(0,2) = (2,0)$	$a(0,2) * b(0,2) = (3,0)$	$a(0,3) * b(0,2) = (1,0)$
$a(0,0) * b(0,3) = (0,0)$	$a(0,1) * b(0,3) = (3,0)$	$a(0,2) * b(0,3) = (1,0)$	$a(0,3) * b(0,3) = (2,0)$
$a(0,0) * b(1,0) = (0,0)$	$a(0,1) * b(1,0) = (0,1)$	$a(0,2) * b(1,0) = (0,2)$	$a(0,3) * b(1,0) = (0,3)$
$a(0,0) * b(1,1) = (0,0)$	$a(0,1) * b(1,1) = (1,1)$	$a(0,2) * b(1,1) = (2,2)$	$a(0,3) * b(1,1) = (3,3)$
$a(0,0) * b(1,2) = (0,0)$	$a(0,1) * b(1,2) = (2,1)$	$a(0,2) * b(1,2) = (3,2)$	$a(0,3) * b(1,2) = (1,3)$
$a(0,0) * b(1,3) = (0,0)$	$a(0,1) * b(1,3) = (3,1)$	$a(0,2) * b(1,3) = (1,2)$	$a(0,3) * b(1,3) = (2,3)$
$a(0,0) * b(2,0) = (0,0)$	$a(0,1) * b(2,0) = (0,2)$	$a(0,2) * b(2,0) = (0,3)$	$a(0,3) * b(2,0) = (0,1)$
$a(0,0) * b(2,1) = (0,0)$	$a(0,1) * b(2,1) = (1,2)$	$a(0,2) * b(2,1) = (2,3)$	$a(0,3) * b(2,1) = (3,1)$
$a(0,0) * b(2,2) = (0,0)$	$a(0,1) * b(2,2) = (2,2)$	$a(0,2) * b(2,2) = (3,3)$	$a(0,3) * b(2,2) = (1,1)$
$a(0,0) * b(2,3) = (0,0)$	$a(0,1) * b(2,3) = (3,2)$	$a(0,2) * b(2,3) = (1,3)$	$a(0,3) * b(2,3) = (2,1)$
$a(0,0) * b(3,0) = (0,0)$	$a(0,1) * b(3,0) = (0,3)$	$a(0,2) * b(3,0) = (0,1)$	$a(0,3) * b(3,0) = (0,2)$
$a(0,0) * b(3,1) = (0,0)$	$a(0,1) * b(3,1) = (1,3)$	$a(0,2) * b(3,1) = (2,1)$	$a(0,3) * b(3,1) = (3,2)$
$a(0,0) * b(3,2) = (0,0)$	$a(0,1) * b(3,2) = (2,3)$	$a(0,2) * b(3,2) = (3,1)$	$a(0,3) * b(3,2) = (1,2)$
$a(0,0) * b(3,3) = (0,0)$	$a(0,1) * b(3,3) = (3,3)$	$a(0,2) * b(3,3) = (1,1)$	$a(0,3) * b(3,3) = (2,2)$

$a(1,0) * b(0,0) = (0,0)$	$a(1,1) * b(0,0) = (0,0)$	$a(1,2) * b(0,0) = (0,0)$	$a(1,3) * b(0,0) = (0,0)$
$a(1,0) * b(0,1) = (0,1)$	$a(1,1) * b(0,1) = (1,1)$	$a(1,2) * b(0,1) = (2,1)$	$a(1,3) * b(0,1) = (3,1)$
$a(1,0) * b(0,2) = (0,2)$	$a(1,1) * b(0,2) = (2,2)$	$a(1,2) * b(0,2) = (3,2)$	$a(1,3) * b(0,2) = (1,2)$
$a(1,0) * b(0,3) = (0,3)$	$a(1,1) * b(0,3) = (3,3)$	$a(1,2) * b(0,3) = (1,3)$	$a(1,3) * b(0,3) = (2,3)$
$a(1,0) * b(1,0) = (1,0)$	$a(1,1) * b(1,0) = (1,1)$	$a(1,2) * b(1,0) = (1,2)$	$a(1,3) * b(1,0) = (1,3)$
$a(1,0) * b(1,1) = (1,1)$	$a(1,1) * b(1,1) = (0,0)$	$a(1,2) * b(1,1) = (3,3)$	$a(1,3) * b(1,1) = (2,2)$
$a(1,0) * b(1,2) = (1,2)$	$a(1,1) * b(1,2) = (3,3)$	$a(1,2) * b(1,2) = (2,0)$	$a(1,3) * b(1,2) = (0,1)$
$a(1,0) * b(1,3) = (1,3)$	$a(1,1) * b(1,3) = (2,2)$	$a(1,2) * b(1,3) = (0,1)$	$a(1,3) * b(1,3) = (3,0)$
$a(1,0) * b(2,0) = (2,0)$	$a(1,1) * b(2,0) = (2,2)$	$a(1,2) * b(2,0) = (2,3)$	$a(1,3) * b(2,0) = (2,1)$
$a(1,0) * b(2,1) = (2,1)$	$a(1,1) * b(2,1) = (3,3)$	$a(1,2) * b(2,1) = (0,2)$	$a(1,3) * b(2,1) = (1,0)$
$a(1,0) * b(2,2) = (2,2)$	$a(1,1) * b(2,2) = (0,0)$	$a(1,2) * b(2,2) = (1,1)$	$a(1,3) * b(2,2) = (3,3)$
$a(1,0) * b(2,3) = (2,3)$	$a(1,1) * b(2,3) = (1,1)$	$a(1,2) * b(2,3) = (3,0)$	$a(1,3) * b(2,3) = (0,2)$
$a(1,0) * b(3,0) = (3,0)$	$a(1,1) * b(3,0) = (3,3)$	$a(1,2) * b(3,0) = (3,1)$	$a(1,3) * b(3,0) = (3,2)$
$a(1,0) * b(3,1) = (3,1)$	$a(1,1) * b(3,1) = (2,2)$	$a(1,2) * b(3,1) = (1,0)$	$a(1,3) * b(3,1) = (0,3)$
$a(1,0) * b(3,2) = (3,2)$	$a(1,1) * b(3,2) = (1,1)$	$a(1,2) * b(3,2) = (0,3)$	$a(1,3) * b(3,2) = (2,0)$
$a(1,0) * b(3,3) = (3,3)$	$a(1,1) * b(3,3) = (0,0)$	$a(1,2) * b(3,3) = (2,2)$	$a(1,3) * b(3,3) = (1,1)$

$a(2,0) * b(0,0) = (0,0)$	$a(2,1) * b(0,0) = (0,0)$	$a(2,2) * b(0,0) = (0,0)$	$a(2,3) * b(0,0) = (0,0)$
$a(2,0) * b(0,1) = (0,2)$	$a(2,1) * b(0,1) = (1,2)$	$a(2,2) * b(0,1) = (2,2)$	$a(2,3) * b(0,1) = (3,2)$
$a(2,0) * b(0,2) = (0,3)$	$a(2,1) * b(0,2) = (2,3)$	$a(2,2) * b(0,2) = (3,3)$	$a(2,3) * b(0,2) = (1,3)$
$a(2,0) * b(0,3) = (0,1)$	$a(2,1) * b(0,3) = (3,1)$	$a(2,2) * b(0,3) = (1,1)$	$a(2,3) * b(0,3) = (2,1)$
$a(2,0) * b(1,0) = (2,0)$	$a(2,1) * b(1,0) = (2,1)$	$a(2,2) * b(1,0) = (2,2)$	$a(2,3) * b(1,0) = (2,3)$
$a(2,0) * b(1,1) = (2,2)$	$a(2,1) * b(1,1) = (3,3)$	$a(2,2) * b(1,1) = (0,0)$	$a(2,3) * b(1,1) = (1,1)$
$a(2,0) * b(1,2) = (2,3)$	$a(2,1) * b(1,2) = (0,2)$	$a(2,2) * b(1,2) = (1,1)$	$a(2,3) * b(1,2) = (3,0)$
$a(2,0) * b(1,3) = (2,1)$	$a(2,1) * b(1,3) = (1,0)$	$a(2,2) * b(1,3) = (3,3)$	$a(2,3) * b(1,3) = (0,2)$
$a(2,0) * b(2,0) = (3,0)$	$a(2,1) * b(2,0) = (3,2)$	$a(2,2) * b(2,0) = (3,3)$	$a(2,3) * b(2,0) = (3,1)$
$a(2,0) * b(2,1) = (3,2)$	$a(2,1) * b(2,1) = (2,0)$	$a(2,2) * b(2,1) = (1,1)$	$a(2,3) * b(2,1) = (0,3)$
$a(2,0) * b(2,2) = (3,3)$	$a(2,1) * b(2,2) = (1,1)$	$a(2,2) * b(2,2) = (0,0)$	$a(2,3) * b(2,2) = (2,2)$
$a(2,0) * b(2,3) = (3,1)$	$a(2,1) * b(2,3) = (0,3)$	$a(2,2) * b(2,3) = (2,2)$	$a(2,3) * b(2,3) = (1,0)$
$a(2,0) * b(3,0) = (1,0)$	$a(2,1) * b(3,0) = (1,3)$	$a(2,2) * b(3,0) = (1,1)$	$a(2,3) * b(3,0) = (1,2)$
$a(2,0) * b(3,1) = (1,2)$	$a(2,1) * b(3,1) = (0,1)$	$a(2,2) * b(3,1) = (3,3)$	$a(2,3) * b(3,1) = (2,0)$
$a(2,0) * b(3,2) = (1,3)$	$a(2,1) * b(3,2) = (3,0)$	$a(2,2) * b(3,2) = (2,2)$	$a(2,3) * b(3,2) = (0,1)$
$a(2,0) * b(3,3) = (1,1)$	$a(2,1) * b(3,3) = (2,2)$	$a(2,2) * b(3,3) = (0,0)$	$a(2,3) * b(3,3) = (3,3)$

$a(3,0) * b(0,0) = (0,0)$	$a(3,1) * b(0,0) = (0,0)$	$a(3,2) * b(0,0) = (0,0)$	$a(3,3) * b(0,0) = (0,0)$
$a(3,0) * b(0,1) = (0,3)$	$a(3,1) * b(0,1) = (1,3)$	$a(3,2) * b(0,1) = (2,3)$	$a(3,3) * b(0,1) = (3,3)$
$a(3,0) * b(0,2) = (0,1)$	$a(3,1) * b(0,2) = (2,1)$	$a(3,2) * b(0,2) = (3,1)$	$a(3,3) * b(0,2) = (1,1)$
$a(3,0) * b(0,3) = (0,2)$	$a(3,1) * b(0,3) = (3,2)$	$a(3,2) * b(0,3) = (1,2)$	$a(3,3) * b(0,3) = (2,2)$
$a(3,0) * b(1,0) = (3,0)$	$a(3,1) * b(1,0) = (3,1)$	$a(3,2) * b(1,0) = (3,2)$	$a(3,3) * b(1,0) = (3,3)$
$a(3,0) * b(1,1) = (3,3)$	$a(3,1) * b(1,1) = (2,2)$	$a(3,2) * b(1,1) = (1,1)$	$a(3,3) * b(1,1) = (0,0)$
$a(3,0) * b(1,2) = (3,1)$	$a(3,1) * b(1,2) = (1,0)$	$a(3,2) * b(1,2) = (0,3)$	$a(3,3) * b(1,2) = (2,2)$
$a(3,0) * b(1,3) = (3,2)$	$a(3,1) * b(1,3) = (0,3)$	$a(3,2) * b(1,3) = (2,0)$	$a(3,3) * b(1,3) = (1,1)$
$a(3,0) * b(2,0) = (1,0)$	$a(3,1) * b(2,0) = (1,2)$	$a(3,2) * b(2,0) = (1,3)$	$a(3,3) * b(2,0) = (1,1)$
$a(3,0) * b(2,1) = (1,3)$	$a(3,1) * b(2,1) = (0,1)$	$a(3,2) * b(2,1) = (3,0)$	$a(3,3) * b(2,1) = (2,2)$
$a(3,0) * b(2,2) = (1,1)$	$a(3,1) * b(2,2) = (3,3)$	$a(3,2) * b(2,2) = (2,2)$	$a(3,3) * b(2,2) = (0,0)$
$a(3,0) * b(2,3) = (1,2)$	$a(3,1) * b(2,3) = (2,0)$	$a(3,2) * b(2,3) = (0,1)$	$a(3,3) * b(2,3) = (3,3)$
$a(3,0) * b(3,0) = (2,0)$	$a(3,1) * b(3,0) = (2,3)$	$a(3,2) * b(3,0) = (2,1)$	$a(3,3) * b(3,0) = (2,2)$
$a(3,0) * b(3,1) = (2,3)$	$a(3,1) * b(3,1) = (3,0)$	$a(3,2) * b(3,1) = (0,2)$	$a(3,3) * b(3,1) = (1,1)$
$a(3,0) * b(3,2) = (2,1)$	$a(3,1) * b(3,2) = (0,2)$	$a(3,2) * b(3,2) = (1,0)$	$a(3,3) * b(3,2) = (3,3)$
$a(3,0) * b(3,3) = (2,2)$	$a(3,1) * b(3,3) = (1,1)$	$a(3,2) * b(3,3) = (3,3)$	$a(3,3) * b(3,3) = (0,0)$

Todos os polinômios da tabela que resultam em $(1,0)$ satisfazem a necessidade para o SAES. Adotou-se um polinômio que possuísse os dois termos $a(x)$ e $b(x)$ com valores diferentes de zero e com valores diferentes para o uso em cifrar e decifrar. Deste modo foram adotados os seguintes polinômios para o SAES: $a(x) = 3x + 1$ e $a^{-1}(x) = x + 2$

3.3.6 Derivação da Caixa-S

A Caixa-S para o SAES foi gerada seguindo o mesmo algoritmo utilizado para a construir a Caixa-S do AES. A Caixa-S poderia ser criada de uma forma aleatória porém pela finalidade didática do SAES será demonstrada a forma de construção sugerida pelos criadores do AES[12] [JD 99a] e utilizada a mesma forma de construção matemática sugerida na especificação do AES [FIP 01]. A Caixa-S no AES é definida com base em duas operações:

1. Encontrar a multiplicativa inversa;

2. Aplicar uma transformada afim;

[12]O processo de construção da Caixa-S do AES está descrito no item 2.6.1

A inversa de a é denotada com a^{-1} e indica o número que multiplicado por a é igual a 1. Desse modo pode-se retirar da tabela de multiplicação todos os multiplicativos inversos com base nos resultados que são igual a 1 conforme ressaltado em negrito na tabela 3.6, sendo que o elemento zero é mapeado sobre si mesmo.

Tabela 3.6: Elementos com multiplicativo inverso

•	01	10	11
01	**01**	10	11
10	10	11	**01**
11	11	**01**	10

Com base na tabela 3.6 encontra-se os multiplicativos inversos:

$01 - 01$

$10 - 11$

$11 - 10$

Para realização da transformada afim segue-se o mesmo princípio do AES, que a define como:

$$b'_i = b_i \oplus b_{(i+4)\ mod\ 8} \oplus b_{(i+5)\ mod\ 8} \oplus b_{(i+6)\ mod\ 8} \oplus b_{(i+7)\ mod\ 8} \oplus c_i$$

O cifrador SAES possui apenas 2 bits, logo:

$$b'_i = b_i \oplus b_{(i+4)\ mod\ 2} \oplus b_{(i+5)\ mod\ 2} \oplus b_{(i+6)\ mod\ 2} \oplus b_{(i+7)\ mod\ 2} \oplus c_i$$

Analisando a equação acima percebe-se que i adicionado a um número par gera o próprio i,sendo que o mesmo acontece para i como número ímpar:

$(0 + 4) = 4\ mod\ 2 = 0$

$(1 + 4) = 5\ mod\ 2 = 1$

Com base nessa comprovação simplifica-se a equação para: $b'_i = b_i \oplus c_i$

Matricialmente pode-se escrever a equação da seguinte forma:

$$b'_i = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} + \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

Simplificando:

$$b'_i = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} + \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

Para definição do elemento c tem-se os seguintes prováveis valores: 00, 01, 10 e 11. Aplicando cada valor provável desses sobre a multiplicativa inversa, tomando como base a tabela de soma, selecionou-se o valor que produz a melhor alteração de bits na saída, isto é: não linearidade.

Nas tabelas 3.7, 3.8, 3.9 e 3.10 foram colocados em negrito os bits alterados na saída para tornar a identificação mais rápida.

Tabela 3.7: Quantidade de elementos modificados utilizando $c = 00$, bits trocados=2

a	$b = a^{-1}$	b‘
00	00	00
01	01	01
10	10	1**1**
11	11	1**0**

Tabela 3.8: Quantidade de elementos modificados utilizando $c = 01$, bits trocados=2

a	$b = a^{-1}$	b‘
00	00	0**1**
01	01	0**0**
10	10	10
11	11	11

Tabela 3.9: Quantidade de elementos modificados utilizando $c = 10$, bits trocados=6

a	$b = a^{-1}$	b‘
00	00	**1**0
01	01	**1**1
10	10	**01**
11	11	**11**

Tabela 3.10: Quantidade de elementos modificados utilizando $c = 11$, bits trocados=6

a	$b = a^{-1}$	b‘
00	00	**11**
01	01	**10**
10	10	**0**0
11	11	**0**1

Com base nos resultados obtidos pelas tabelas 3.7, 3.8, 3.9 e 3.10 foi adotado $c = 10$. A Caixa-S resultante, e sua inversa matemática usada para decifrar, podem ser observadas na figura 3.8, sendo que o primeiro bit indica a linha e o segundo a coluna do elemento que será utilizado como saída da Caixa-S.

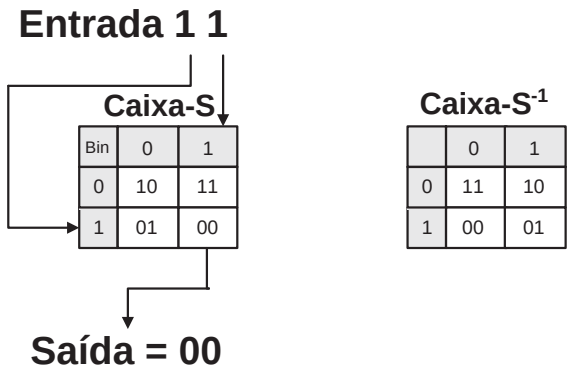

Figura 3.8: Exemplo de uso da Caixa-S no SAES e a Caixa-S inversa

3.4 Transformação AdicionaChaveRodada no SAES

A transformação AdicionaChaveRodada no SAES diferencia-se da utilizada no AES apenas pelo tamanho da Matriz Estado e tamanho do elemento do estado, sendo que a transformação realiza a operação, em nível macro, de forma idêntica. Na figura 3.9 pode-se visualizar a transformação.

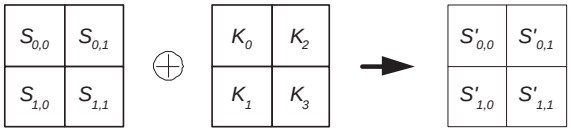

Figura 3.9: Transformação AdicionaChaveRodada sobre a matriz estado no SAES

3.5 Transformação ByteSub no SAES

A transformação ByteSub no SAES diferencia-se da utilizada no AES apenas pelo tamanho da Matriz Estado, tamanho do elemento do estado e Caixa-S utilizada. Da mesma forma que no AES a metade dos bits menos significativa indica a linha e os bits mais significativos a coluna, sendo que a transformação realiza a operação de forma idêntica conforme pode ser verificado através da figura 3.10.

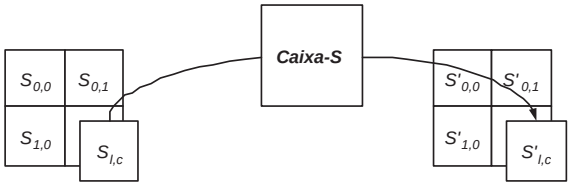

Figura 3.10: Transformação ByteSub sobre a matriz estado no SAES

3.6 Transformação DeslocamentoLinha no SAES

A transformação DeslocamentoLinha pode ser observada na figura 3.11. Da mesma forma que no AES, a primeira linha não sofre rotacionamento, e a linha

seguinte sofre um rotacionamento cíclico à esquerda de um elemento da matriz estado.

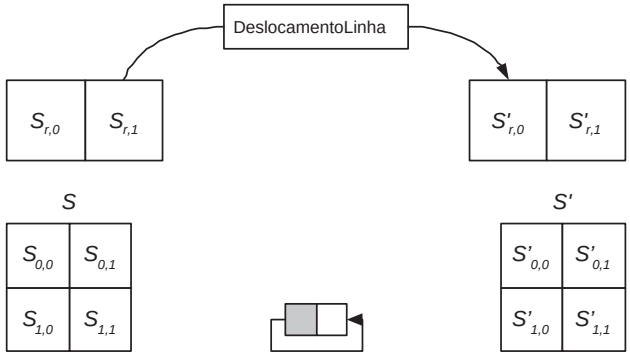

Figura 3.11: Transformação DeslocamentoLinha sobre a matriz estado no SAES

A transformação InvDeslocamentoLinha pode ser observada na figura 3.12. Da mesma forma que no AES, a primeira linha não sofre rotacionamento, e a linha seguinte sofre um rotacionamento cíclico à direita de um elemento da matriz estado.

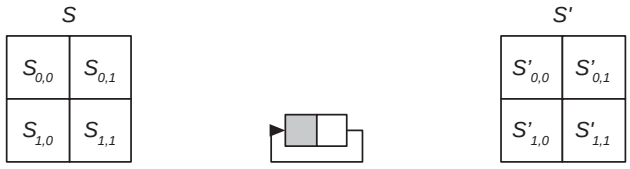

Figura 3.12: Transformação InvDeslocamentoLinha sobre a matriz estado no SAES

3.7 Transformação MixColumn no SAES

A transformação MixColumn no SAES segue os mesmos princípios de operação do AES, diferenciando-se apenas por tratar as colunas como um polinômio de dois termos[13] considerando-as como polinômios sobre um corpo $GF(2^2)$ e fazendo uma multiplicação mod $x^2 + 1$, individualmente, com o polinômio fixo $a(x) = 3x + 1$. Deste modo tem-se: $S'(x) = a(x) \otimes S(x)$. A expressão pode ser representada matricialmente por:

[13] no AES trata como um polinômio de quatro termos

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 3 & 1 \end{bmatrix} \otimes \begin{bmatrix} S_{0,c} \\ S_{1,c} \end{bmatrix}$$

Como resultado dessa multiplicação obtém-se:

$$S'_{0,c} = (01 \bullet S_{0,c}) \oplus (03 \bullet S_{1,c})$$
$$S'_{1,c} = (03 \bullet S_{0,c}) \oplus (01 \bullet S_{1,c})$$

A operação inversa InvMixColumn difere da MixColumn no uso do polinômio $x + 2$ que tem por objetivo fazer a inversão dos valores no processo de decifrar gerados pelo polinômio $3x + 1$ utilizado no processo de cifrar. Deste modo tem-se: $S'(x) = a^{-1}(x) \otimes S(x)$. A expressão pode ser representada matricialmente por:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \otimes \begin{bmatrix} S_{0,c} \\ S_{1,c} \end{bmatrix}$$

Como resultado dessa multiplicação obtém-se:

$$S'_{0,c} = (02 \bullet S_{0,c}) \oplus (01 \bullet S_{1,c})$$
$$S'_{1,c} = (01 \bullet S_{0,c}) \oplus (02 \bullet S_{1,c})$$

Pode-se analisar o escopo da função MixColumn/InvMixColumn, através de uma visão macro, pela figura 3.13.

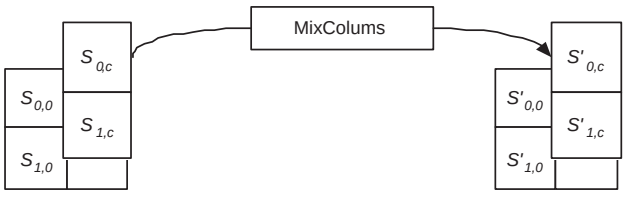

Figura 3.13: Transformação MixColumn sobre a matriz estado no SAES

3.8 Geração de SubChaves no SAES

O processo de geração de subchaves de rodadas utilizado no SAES é muito similar ao empregado no AES. Analisando o processo percebe-se que o mesmo é completamente parametrizável, sendo assim para o SAES será mantido o mesmo algoritmo. As funções SubPalavra e Rotacao permanecem inalteradas. Como resultado da implementação dos pontos de ajuste obtém-se o seguinte algoritmo:

```
ExpansaoChaveSAES(byte Chave[2 * Nk],palavra w[Nb * (Nr + 1)],Nk)
inicio
  i=0
  enquanto (i < Nk)
    w[i] = palavra[Chave[2*i],Chave24*i+1]]
    i = i +1
  fim enquanto
  i = Nk
  enquanto (i < Nb * (Nr + 1))
    temp = w[i - 1]
    se (i mod Nk = 0)
      entao
        temp = SubPalavra(Rotacao(temp)) xor cons[i / Nk]
      senao
        temp = SubPalavra(temp)
    fim se
    w[i] = w[i - Nk] xor temp
    i = i + 1
  fim enquanto
fim
```

Alguns pontos foram ajustados no algoritmo de geração de chaves a fim de torná-lo compatível com o SAES:

- No algoritmo utilizado no AES as variáveis são definidas como byte, porém no SAES em 4 bits apenas;
- A variável Chave consiste no AES de um conjunto de no mínimo 4 palavras, porém no SAES consistem em 2 palavras;
- Devido ao tamanho da chave ser menor o processo de inicialização das subchaves deve ser adaptado;

- *Nk* possui um valor fixo de 2 para o SAES;
- A obtenção do valor de $cons$ que no AES era dado por $[x^{i-1}, 00, 00, 00]$, foi reduzido no SAES para $[x^{i-1}, 00]$

Exemplo. A tabela 3.11 demonstra o processo de geração de chaves de rodadas para: 10 00 01 00 .

Nesse exemplo encontra-se descrito o processo detalhado conforme o algoritmo descrito anteriormente. Devido ao algoritmo não possuir uma execução simples recomenda-se o uso de tabelas de variáveis similares a apresentada na tabela 3.11 para facilitar o processo de execução e assim torná-lo mais rápido. Observa-se que as subchaves são armazenadas no vetor *w* em conjuntos de 4 bits que representam uma coluna da matriz estado, deste modo dois elementos do vetor *w* correspondem à uma subchave de rodada do SAES.

Tabela 3.11: Geração das subchaves de rodadas do SAES para: 10 00 01 00

i	temp	Após RotPalavra	Após Caixa-S	Rcon$[i/Nk]$	Após *XOR* $\oplus$ Rcon	$w[i-Nk]$	$w[i]$
0							10 00
1							01 00
2	01 00	00 01	10 11	01 00	11 11	10 00	01 11
3	11 00					01 00	10 00
4	10 00	00 10	10 01	10 00	00 01	01 11	01 10
5	11 01					10 00	01 01

3.9 Processo de Cifrar com o SAES

O processo de cifrar do SAES envolve uma seqüencia pré-estabelecida das transformações, sendo que todas essas transformações foram explicadas nas seções anteriores. A principal mudança na estrutura macro do algoritmo encontra-se na retirada da estrutura de repetição responsável pela quantidade de rodadas. Na figura 3.14 pode-se analisar a estrutura de cifrar do SAES em uma visão macro.

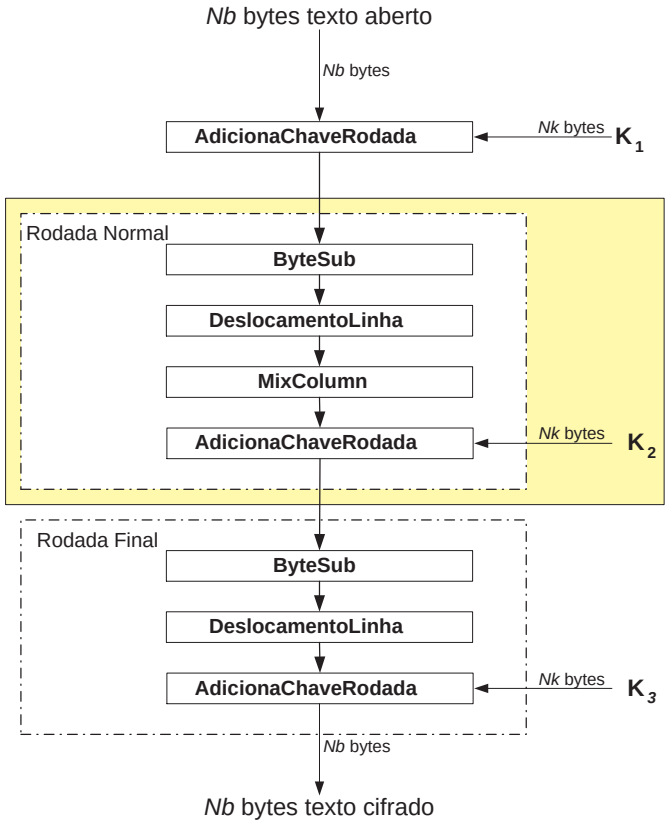

Figura 3.14: Fluxo macro do processo de cifrar com o SAES

O algoritmo resultante para o SAES pode ser observado abaixo:

```
Cifrar(entrada[2*Nb], saida[2*Nb], palavra w[Nb*(Nr+1)])
  Inicio
    estado[2,Nb]
    estado = entrada
    AdicionaChaveRodada(estado, w)

    ByteSub(estado)
    DeslocamentoLinha(estado)
    MixColumn(estado)
    AdicionaChaveRodada(estado, w + rodada * Nb)
    rodada=rodada+1

    ByteSub(estado)
    DeslocamentoLinha(estado)
    AdicionaChaveRodada(estado, w + rodada * Nb)
    saida = estado
  Fim
```

Exemplo. Processo de cifrar com o SAES.

Chave: 10, 00, 01, 00

Texto Aberto: 10, 00, 00, 01

```
Rodada:  0
                      Entrada: 10 00 00 01
              Chave de Rodada: 10 00 01 00
     Apos AdicionaChaveRodada: 00 00 01 01
Rodada:  1
                Valor Inicial: 00 00 01 01
                 Apos Caixa-S: 10 10 11 11
       Apos DeslocamentoLinha: 10 11 11 10
              Apos MixColumns: 00 10 10 00
              Chave de Rodada: 01 11 10 00
     Apos AdicionaChaveRodada: 01 01 00 00
Rodada:  2
                Valor Inicial: 01 01 00 00
                 Apos Caixa-S: 11 11 10 10
       Apos DeslocamentoLinha: 11 10 10 11
              Chave de Rodada: 01 10 01 01
     Apos AdicionaChaveRodada: 10 00 11 10

                Texto Cifrado: 10 00 11 10
```

Através desse exemplo pode-se observar o resultado obtido após a execução de cada transformação. Usando este modo, para realizar o processo de cifrar manualmente, pode ser observado após a execução manual de cada transformação as variações que os bits sofreram e também verificar se o processo está correto através da comparação dos resultados gerados pelo programa de implementação didática do SAES contido no Anexo B.

Uma modo melhor visualizar é através da visualização das operações sobre o estado, na figura 3.15 pode-se observar esse processo para cifrar com o SAES.

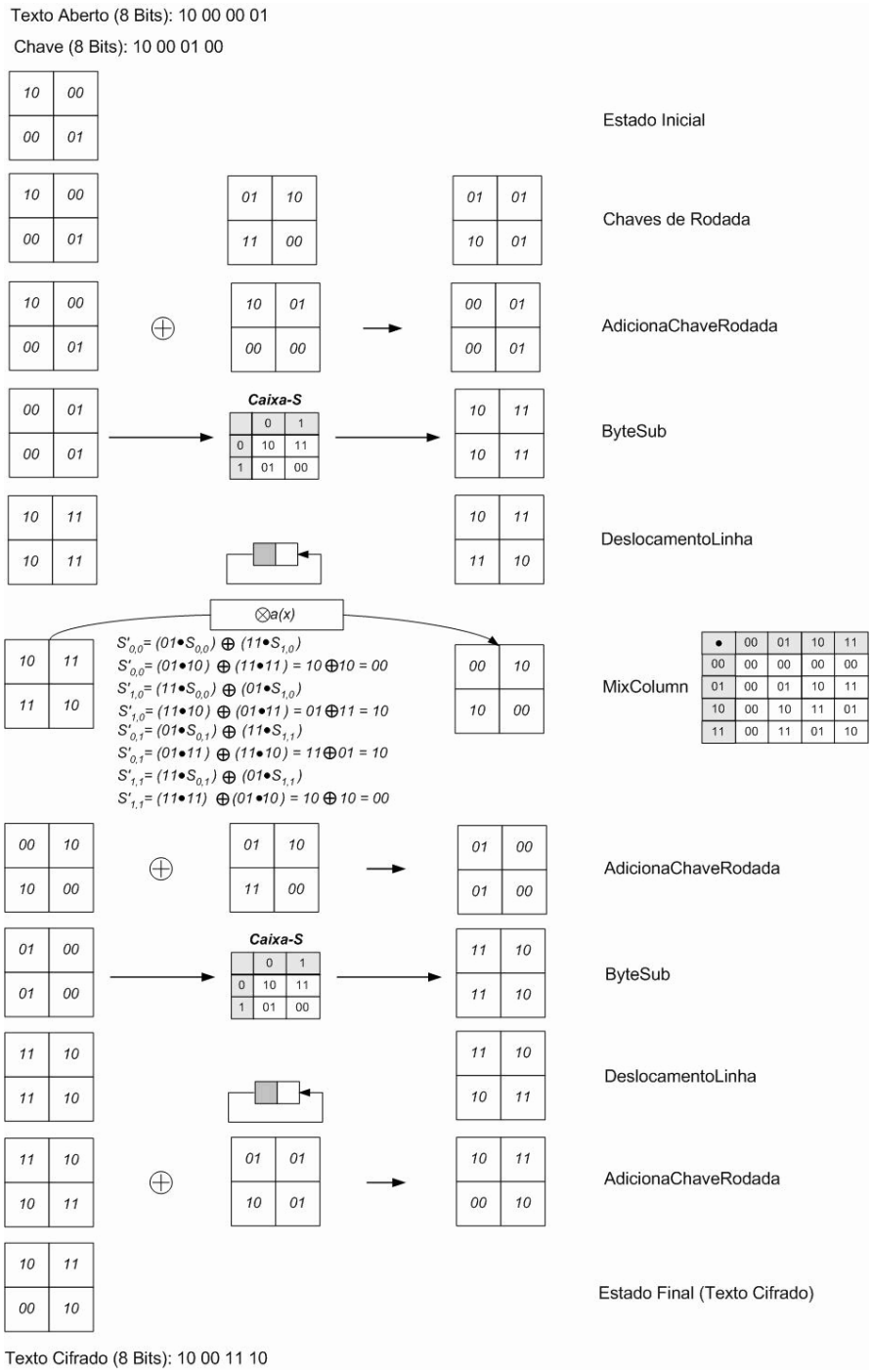

Figura 3.15: Visualização detalhada do processo de cifrar com o SAES

O processo de cifrar pode ser generalizado para uso na construção de exemplos manuais[14] através da figura 3.16.

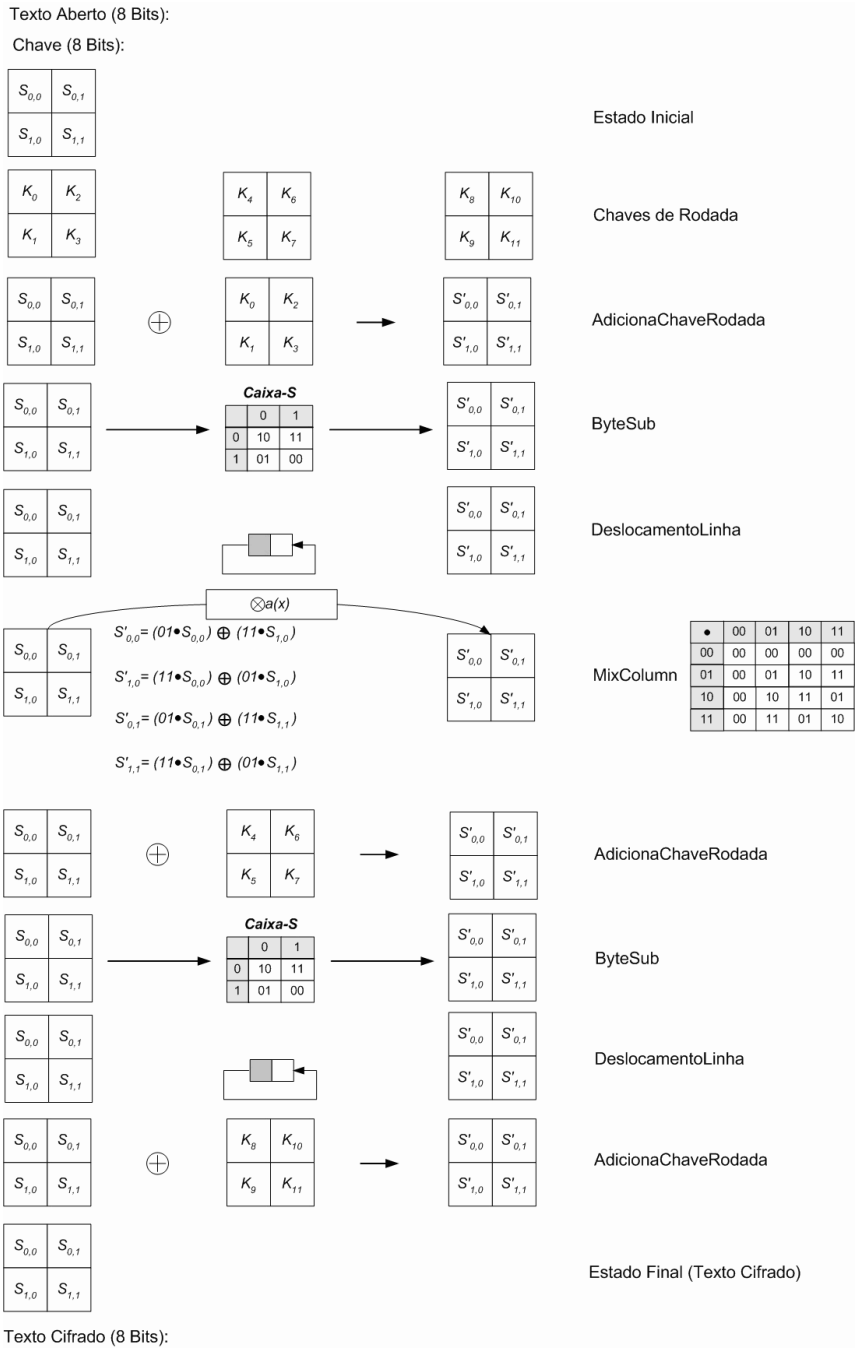

Figura 3.16: Visualização genérica do processo de decifrar com o SAES a nível de estado

[14]Partindo-se das subchaves de rodadas já calculadas

3.10 Processo de Decifrar com o SAES

O processo de decifrar no SAES consiste na execução de transformações diferentes. Do mesmo modo que o AES, o SAES, necessita de inversas matemáticas de suas transformações para realizar o processo de decifrar.

A estrutura de processo de decifrar adotada no SAES baseia-se no primeiro algoritmo citado na FIPS197, isto é, na inversão matemática das transformações. Embora exista uma derivação matemática/algébrica do modelo de inversão matemática para um modelo equivalente de decifrador adotou-se a abordagem matemática por facilitar a identificação de prováveis erros na implementação manual do cifrador.

Analisando o fluxo demonstrado na figura 3.17 pode-se perceber as diferenças a nível de função do processo utilizado para decifrar.

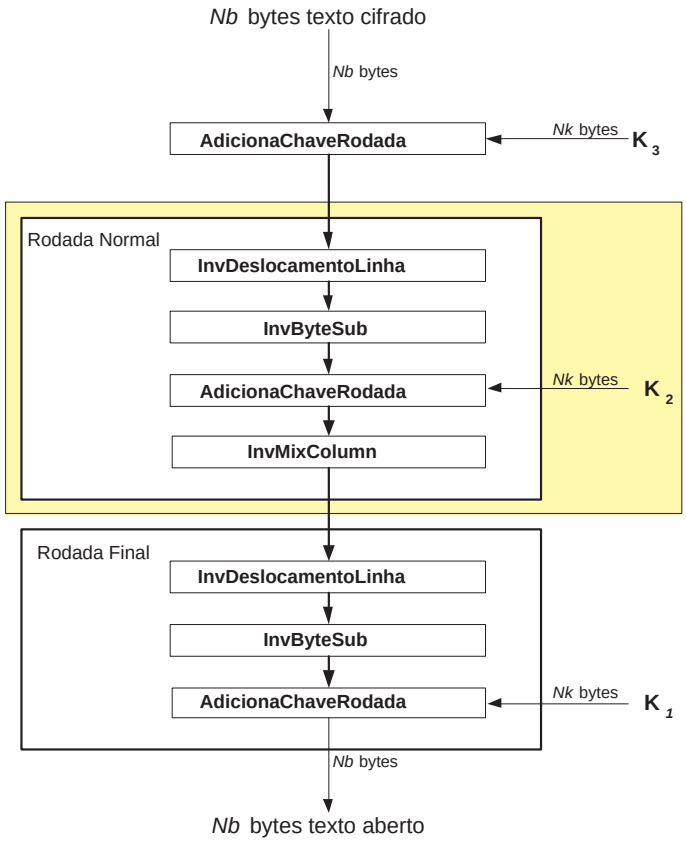

Figura 3.17: Fluxo macro do processo de decifrar com o SAES

O algoritmo resultante para o SAES pode ser observado abaixo:

```
Decifrar(entrada[2*Nb],saida[2*Nb],palavra w[Nb*(Nr+1)])
  Inicio
    byte estado[2,Nb]
    estado = entrada
    AdicionaChaveRodada(estado, w+Nr*Nb)

    Inv_DeslocamentoLinha(estado)
    Inv_ByteSub(estado)
    AdicionaChaveRodada(estado, w + rodada * Nb)
    Inv_MixColumns(estado)
    rodada=rodada-1

    Inv_DeslocamentoLinha(estado)
    Inv_ByteSub(estado)
    AdicionaChaveRodada(estado, w)
    saida = estado
  Fim
```

Exemplo. Processo de decifrar com o SAES.

Chave: 10, 00, 01, 00

Texto Cifrado: 10, 00, 11, 10

```
Rodada:  0
                     Entrada: 10 00 11 10
              Chave de Rodada: 10 00 01 00
     Apos AdicionaChaveRodada: 11 10 10 11
Rodada:  1
                Valor Inicial: 11 10 10 11
    Apos InvDeslocamentoLinha: 11 11 10 10
              Apos InvCaixa-S: 01 01 00 00
              Chave de Rodada: 01 11 10 00
   Apos AdicionaChavedeRodada: 00 10 10 00
           Apos InvMixColumn: 10 11 11 10
Rodada:  2
                Valor Inicial: 10 11 11 10
    Apos InvDeslocamentoLinha: 10 11 11 11
              Apos InvCaixa-S: 00 00 01 01
              Chave de Rodada: 10 00 01 00
          Apos AdicionaChave: 10 00 00 01

                 Texto Aberto: 10 00 00 01
```

Um modo melhor visualizar é através da visualização das operações sobre o estado, na figura 3.18 pode-se observar esse processo para decifrar com o SAES.

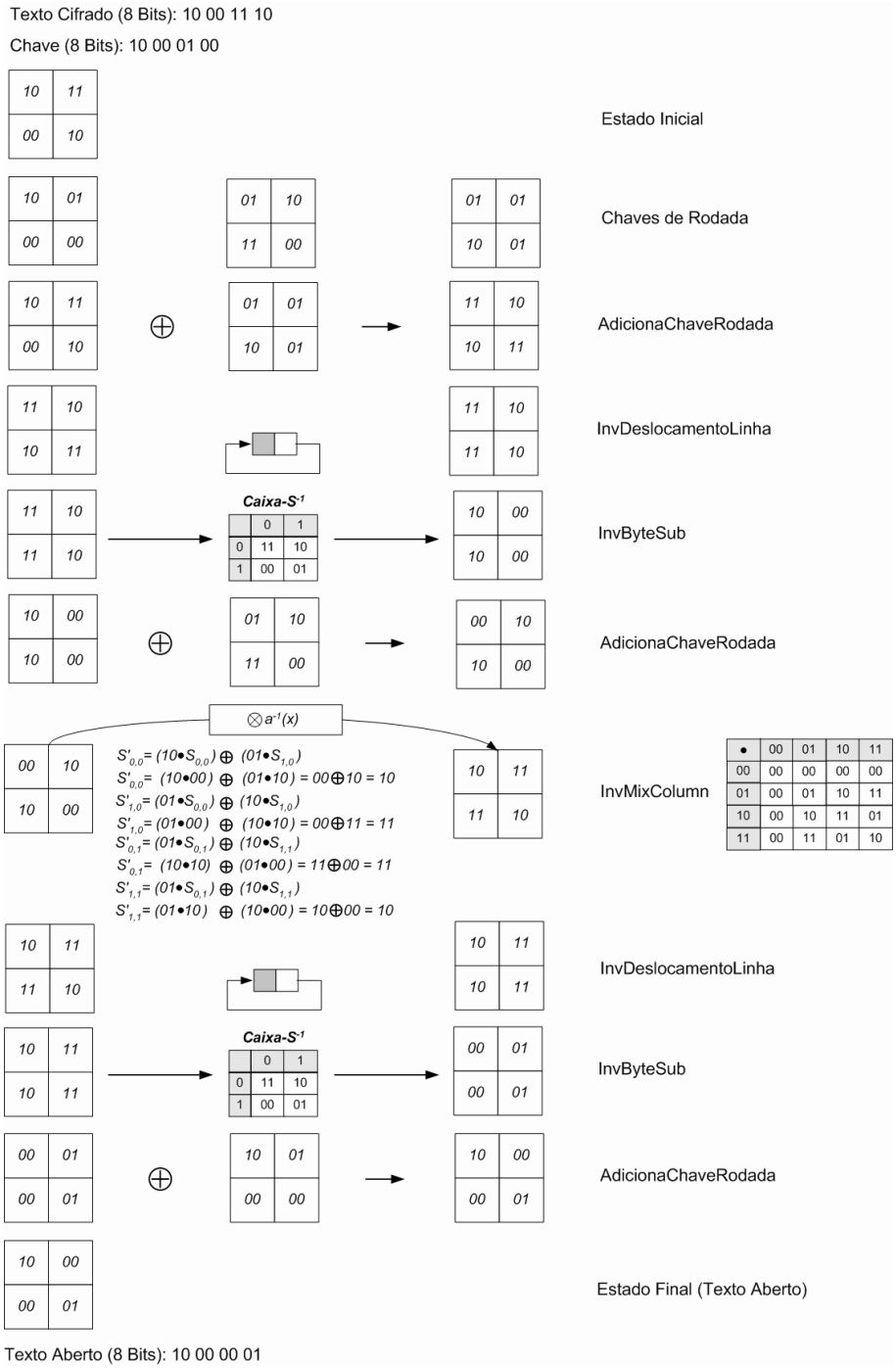

Figura 3.18: Visualização detalhada do processo de decifrar com o SAES

O processo de decifrar pode ser generalizado para uso na construção de exemplos manuais[15] através da figura 3.19.

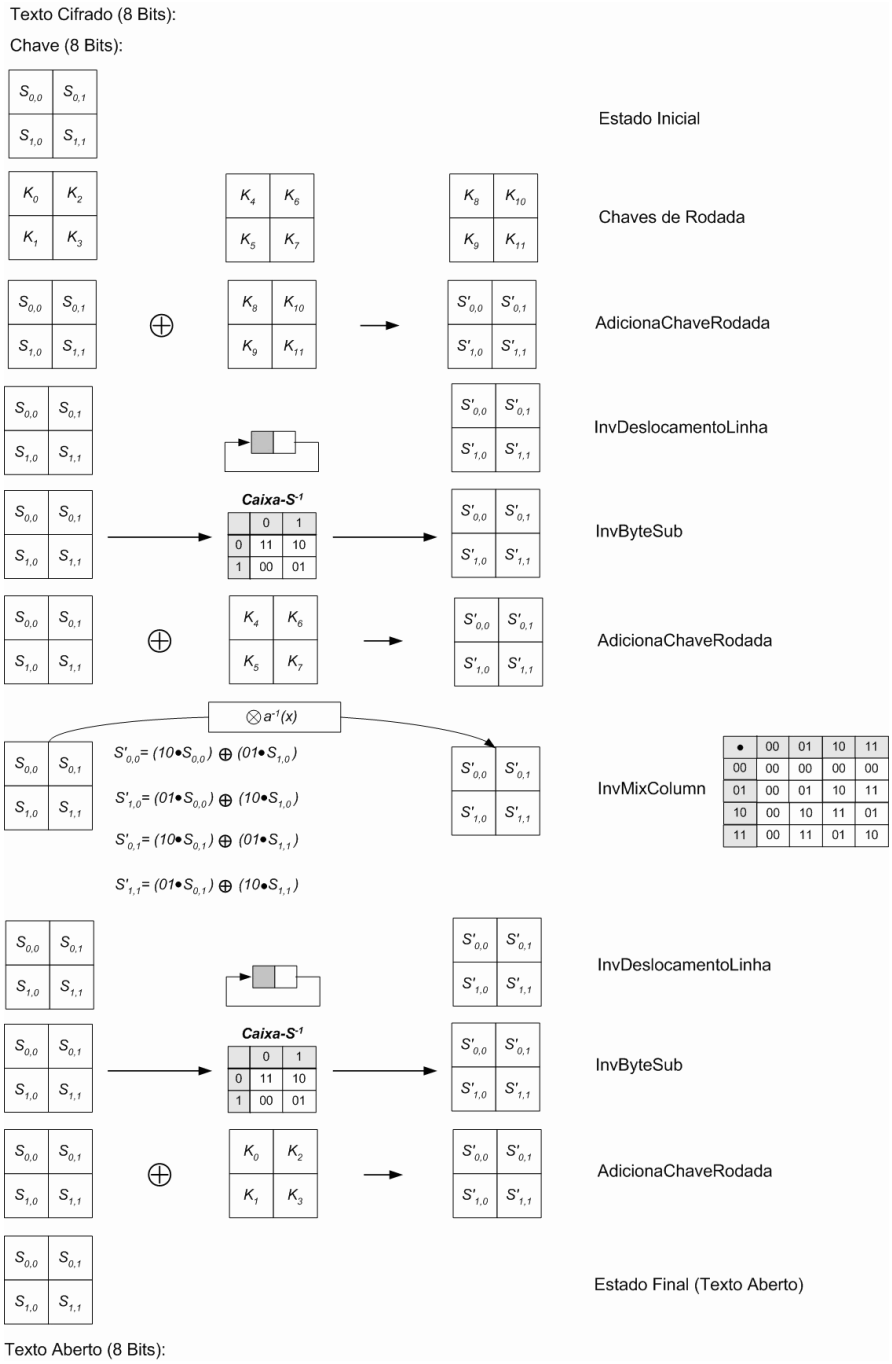

Figura 3.19: Visualização genérica do processo de decifrar com o SAES a nível de estado

[15]Partindo-se das subchaves de rodadas já calculadas

3.11 Relação do AES com o SAES

Através da tabela 3.12 pode-se verificar a redução realizada do AES para o SAES, adotou-se o AES com *Nk*=4 e *Nb*=4 (como base de referência) por ser a menor versão em termos do tamanho de chaves, bloco de entrada e quantidade de rodadas.

Tabela 3.12: Comparativo entre AES (*Nk*=4 e *Nb*=4) e SAES.

Cifrador	AES	SAES
Número de rodadas (cifrar-decifrar)	10	2
Número de rodadas (gerar chaves)	40	4
Bloco de entrada (bits)	128	8
Tamanho de chave (bits)	128	8
Tamanho do Corpo Finito	$GF(2^8)$	$GF(2^2)$
Tamanho da Matriz Estado	4x4	2x2
Quantidade de subchaves necessárias	44	6
Tamanho da Caixa-S	16x16	2x2
Quantidade de Caixas-S	2	2

A operação que menos sofreu alterações estruturais foi a do processo de geração de subchaves, que também pode ser vista como a operação mais complexa no SAES. Embora tenha ocorrido uma redução do número de subchaves necessárias para o processo de cifrar e decifrar essa não decorreu de uma simplificação do algoritmo e sim da redução do número de rodadas padrão. A simplificação do algoritmo implicaria na descaracterização do processo original o que não satisfaria as premissas pretendidas com o SAES. Devido ao fato de ser um algoritmo com características e operações específicas a própria redução é de difícil mensuração.

Analisando a relação entre AES e SAES fica constatado que o SAES cumpre aos pré-requisitos estabelecidos para uma versão simplificada do cifrador AES com fins didáticos. As transformações no SAES preservam as características do AES e possibilitam a compreensão do cifrador AES a partir do estudo do cifrador SAES.

3.12 Conclusão

Neste capítulo foi detalhado a metodologia e o processo de construção do modelo simplificado do cifrador de bloco simétrico AES. Pode-se observar que as simplificações macros (estruturais) não são de difícil realização e que a maior complexidade foi encontrada na simplificação das estruturas matemáticas e algébricas empregadas no cifrador, mais em específico na transformação MixColumn e na derivação da Caixa-S.

A documentação original do Rijndael [JD 99a] e a especificação do AES [FIP 01] não tem por objetivo detalhar os princípios de funcionamento matemático do AES, partindo como premissa que o leitor já possui um conhecimento médio ou avançado em Corpos GF. Desse modo a fundamentação matemática apresentada no Capítulo do AES (Capítulo 2) é necessária para o aprendizado tanto do SAES como do próprio AES. A escolha dos polinômios irredutíveis e inversíveis para a operação MixColumn permitiu um conhecimento maior sobre a forma como fora construído o cifrador. Fundamentado na metodologia definida fora desenvolvido o modelo simplificado buscando uma orientação didática, o que implicou no desenvolvimento das implementações didáticas do AES e SAES em linguagem C.

Capítulo 4

Considerações Finais

Essa dissertação de mestrado demonstra o desenvolvimento de um modelo simplificado do cifrador de blocos simétrico AES, o SAES-Simplified Advanced Encryption Standard. O modelo simplificado é de fácil compreensão e fora elaborado de modo a possuir uma boa didática de forma que possa ser utilizado nas fases finais de cursos de graduação e pós-graduação para o estudo do AES. Embora não conste nos objetivos, dessa dissertação de mestrado, o SAES também poderá ser utilizado como fonte de estudos na busca de possíveis pontos fracos no cifrador AES.

A especificação do Rijndael [JD 99a] e a especificação do AES [FIP 01] explicam o funcionamento matemático e algébrico das operações utilizadas no cifrador. Toda a explicação é fundamentada no princípio de que o leitor já conhece Corpos GF, o que na prática nem sempre é a realidade encontrada. Desse modo a fundamentação matemática apresentada no Capítulo do AES (Capítulo 2) é necessária para o aprendizado tanto do SAES como do próprio AES.

Devido ao aspecto matemático do AES, a abordagem adotada para simplificação depende diretamente das reduções em estruturas matemáticas e algébricas. As operações principais no AES são realizadas em um corpo finito $GF(2^8)$, logo a redução do tamanho do corpo GF para um tamanho menor teve que manter o aspecto das demais transformações de modo que o SAES atenda os objetivos estabelecidos no item 3.3.

Observando o diagrama da figura 2.7 pode-se perceber que o número

mínimo de rodadas é duas, sendo necessário fazer uma expansão de chave para criação de três subchaves. A redução no tamanho do corpo GF faz com que o tamanho do bloco de entrada e chave sejam alterados de forma a adequarem-se ao tamanho do corpo.

Através do uso de tabelas genéricas como a tabela 3.19 e 3.16 para a implementação manual do SAES em sala de aula consegue-se atingir um dos objetivos didáticos dessa dissertação que é o de possibilitar a realização manual do SAES para Cifrar e Decifrar em sala de aula. Aliada a essas tabelas pode-se utilizar os programas contidos no Anexos A e B para servirem como fonte de respostas durante o processo manual. Também pode-se utilizar as implementações como meio para comparação entre o AES e SAES e possibilitar o entendimento do AES através do SAES que é objetivo principal dessa dissertação.

Uma das principais dificuldades encontradas durante o trabalho foram os testes manuais de cifrar, decifrar e geração de chaves pois as implementações que foram encontradas forneciam apenas o texto cifrado/aberto e não os estados intermediários. Apenas com esse tipo de informação tornava-se complicado descobrir o ponto em que uma operacão não estava correta, deixando moroso o processo de correção. Devido a esse fator que fora desenvolvida e implementada uma versão didática do AES cujo o objetivo não é cifrar/decifrar arquivos mas sim mostrar com mais detalhes todas os estados intermediários dos processos de cifrar, decifrar e geração de chaves. A forma de utilização do programa está descrita no Anexo A, sendo que os exemplos apresentados no Capítulo 2 foram gerados a partir dessa implementação didática.

Com base na experiência obtida na implementação didática do AES[1] fora desenvolvida uma implementação didática do SAES com os mesmos objetivos. Porém existe um diferencial na notação utilizada nos elementos da matriz estado, enquanto na implementação do AES utilizou-se a notação hexadecimal, adotou-se a notação decimal na implementação do SAES. A notação decimal foi empregada devido a facilidade de visualização dos elementos, porém nos exemplos contidos no Capítulo3 converteu-se manualmente os valores de decimal para binário devido as transformações adotadas serem mais simples de serem realizadas e interpretadas em binário. Poderia ser adotada

[1] que comprovou a praticidade no estudo e correção dos processos de cifrar, decifrar e geração de chaves

a notação decimal em ambos os casos, porém a maioria das referências bibliográficas do AES já utilizam a notação hexadecimal por padrão, o fato do SAES ser um modelo novo deu a liberdade de adotar para ele uma notação que possibilitasse alunos de graduação e pós-graduação converter mais rapidamente os valores para binário. Não fora desenvolvido uma versão do SAES para cifrar e decifrar arquivos por entender que o objetivo da dissertação não era construir um novo cifrador e sim explicar os processos e transformações do AES.

As versões reduzidas do AES existentes não tem como objetivo a didática ou ensino do cifrador AES, apenas demonstram técnicas de criptoanálise. Estes estudos foram elaborados durante o processo de seleção do AES para demonstrar questões relativas a segurança do cifrador e sempre foram elaborados buscando uma aproximação do cifrador real tendo como objetivo demonstrar ao NIST e todos participantes do processo de seleção os resultados alcançados. Levando-se em consideração esses aspectos o desenvolvimento de uma versão simplificada (ou versão reduzida) do AES para fins didáticos consiste em um bom trabalho no qual é pretendido que torne-se referência no desenvolvimento de outros modelos de cifradores simplificados e no estudo do AES.

Referências Bibliográficas

[AE 00] AJ. ELBIRT, W. YIP, B. C. An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists, 2000.

[BS 98] B. SCHNEIER, J. KELSEY, D. W. D. W. Two fish: A 128-bit block cipher. in First Advanced Encryption Standard (AES) Conference, 1998.

[CB 98] C. BURWICK, D. COPPERSMITH, E. D. R. G. S. H. C. J. S. M. M. L. O. M. P. D. S. Mars - a candidate cipher for AES. in First Advanced Encryption Standard (AES) Conference, 1998.

[DAE 95] DAEMEN, J. **Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis**. Katholieke Universiteit Leuven, 1995. Tese de Doutorado.

[DW 00] DAVE WRESKI, V. R. LinuxSecurity.com speaks with AES winner. Disponível em <http://www.linuxsecurity.com/feature_stories/interview-aes-1.html>, 27 de outubro, 2000. Entrevista.

[EB 00] ELI BIHAM, N. K. Cryptanalysis of reduced variants of rijndael, 2000.

[FIP 01] FIPS197: AES - advanced encryption standar, November, 2001.

[FK 99] FRANÇOIS KOEUNE, J.-J. Q. A timing attack against rijndael. Univeristé Catholique de Louvain, 1999. Relatório TécnicoCG-1999/1.

[HER 70] HERNSTEIN, I. N. **Tópicos Em Algebra (Topics in Algebra)**. Av. Brig. Luís Antonio, 3035 - Sao Paulo / SP:Editora Polígono S.A., 1. ed., 1970.

[HG 00] HENRI GILBERT, M. M. A collision attack on 7 rounds of rijndael, 2000.

[HHD 82] HYGINO H. DOMINGUES, G. I. **Álgebra Moderna**. Av. Marquês São Vicente, 1697 - Barra Funda - São Paulo/SP:Atual Editora, 1982.

[JAP 01] Japan, T. A., editor. **Perfomance and Security of Block Ciphers Using Operations in** $GF(2^n)$, 1-1-32 Shinùrashima, Kanaagawa-ku, Yokohama, 221, Japan, 2001. Telecomunications Advancement Organizatrion of Japan.

[JD 99a] JOAN DAEMEM, V. R. The rijndael block cipher, 1999.

[JD 99b] JOAN DAEMEN, LARS KNUNDSEN, V. R. The block cipher square, 1999.

[JD 00] JOAN DAEMEM, V. R. Answer to ”new observations on rijndael”, august, 2000.

[KAW 00] KAWAMURA, F. S. M. K. S. Performance evaluation of AES finalists on the high-end smart card, 2000.

[MCE 89] MCELIECE, R. J. **Finite Fields for Computer Scientists and Engineers**. 101 Philip Drive, Asinippi Park, Norwell, Massachusets, 02061, USA:Kluwer Academic Publishers, 2. ed., 1989.

[MEN 97] MENEZES, P. V. O. S. V. A. **HandBook of Applied Criptography**. Boca Raton, FL - USA:CRC Press, 1. ed., 1997.

[MEN 01] MENDES, M. A. L. Modelo simplificado do cifrador RC6. UFSC-Universidade Federal de Santa Catarina, 2001. Dissertação de Mestrado.

[MIE 02] MIERS, C. C. Proposta de um modelo simplificado do cifrador AES. Trabalho Individual-CPGCC/UFSC, 2002.

[NF 00] NIEL FERGUNSON, JOHN KELSEY, S. L. B. S. M. S. D. W. D. W. Improved cryptanalysis of rijndael. In: AES, editor, SEVENTH FAST SOFTWARE ENCRYPTION WORKSHOP, 2000. **Proceedings...** Berlin, Germany / Heidelberg, Germany / London, UK / etc.:Springer-Verlag, 2000. p.19.

[RA 98] R. ANDERSON, E. B. Serpent: A proposal for the advanced encryption standard. in First Advanced Encryption Standard (AES) Conference, 1998.

[RIJ 00] RIJMEN, V. Eficient implementation of the rijndael s-box, 2000.

[ROT 94] ROTMAN, J. J. **An Introduction to the Theory of Groups**. 175 Fifth Avenue, New York, NY10010, USA:Springer-Verlag, 4. ed., 1994.

[RR 98] R. RIVEST, M. ROBSHAW, R. S. The RC6 block cipher. in First Advanced Encryption Standard (AES) Conference, 1998.

[SCH 96] SCHAEFER, E. F. A simplified data encryption algorithm. **Journal of Cryptologia**, [S.l.], v.20, n.20, p.77–84, 1996.

[SHA 49] SHANNON, C. Communication theory of secrecy systems. **Bell Systems Technical Journal**, [S.l.], , n.4, 1949.

[SM 00] SEAN MURPHY, M. R. New observations on rijndael, 2000.

[STA 98] STALLINGS, W. **Criptography And Network Security: Principles And Practice**. Upper Saddle River, New Jersey, 07458, USA:Prentice-Hall Internacional, 2. ed., 1998.

[WAS 01] WASHINGTON, W. T. L. C. **Introduction to Cryptography with Coding Theory**. Upper Saddle River, NJ 07458:Prentice Hall, 2001.

[WIE 93] Wiener, M. J., editor. **Efficient DES Key Search**. Springer-Verlag, 1993. Crypto 93.

Apêndice A

Implementação em C do AES

Neste anexo está contida uma implementação do cifrador AES com fins didáticos através do qual foram gerados os exemplos utilizados nessa dissertação. Essa implementação não tem por finalidade o uso para cifrar arquivos nem operar em modos (CBC, CFB, etc.) apenas auxiliar no aprendizado do cifrador.

Instruções para o uso do AES.exe:

- Criar um arquivo com o Texto Aberto, como por exemplo: 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff, aonde os valores devem ser escritos em notação Hexadecimal. O programa converte para binário, o arquivo deve ser salvo como padrão ASCII.
- Criar um arquivo com a Chave, por exemplo: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f , aonde os valores devem ser escritos em notação Hexadecimal. O programa converte para binário, o arquivo deve ser salvo como padrão ASCII.
- Executar via shell o seguinte comando AES <arquivo com o texto aberto> <arquivo com a chave>, por exemplo AES Aberto.txt Chave.txt.

O programa criará no mesmo diretório um arquivo com o nome state.txt, no formato ASCII puro, que contém passo a passo o processo de cifrar e decifrar assim como um arquivo chamado keyexp.txt, no formato ASCII, contendo o processo de geração de subchaves de rodadas.

aes.h

```
#define Nk 4
#define Nb 4
#define Nr 10

#define BYTE unsigned char

union word {
  unsigned long ulw;
  BYTE ucw[4];
};

#define RE(x,i)  ((x)<<((i)*8))|((x)>>((sizeof(x)-(i))*8))

struct state {
  union word w[4];
};

BYTE vezesx(BYTE byte){
  BYTE resultado = byte;
  //Polinomio irredutivel utilizado (m)
  //x^8+x^4+x^3+x+1 ou {1 0001 1011} ou {01} {1b}
  //objetivo reduzir o polinomio ao grau 7 (1 byte)
  //byte pode ser representado por
  //b7*x^7+b6*x^6+b5*x^5+b4*x^4+b3*x^3+b2*x^2+b1*x^1+b0
  //que vezes x de
  //b7*x^8+b6*x^7+b5*x^6+b4*x^5+b3*x^4+b2*x^3+b1*x^2+b0*x^1
  //que nada mais  que byte deslocado de 1 bit a esquerda
  if( ! (byte & 0x80) ) //se b7 = 0 o polinomio ja esta reduzido
    resultado = byte << 1;
  else //se b7 diferente de 0,  necessario aplicar o modulo com m
  //que nada mais  que um xor
    resultado = (byte << 1) ^ 0x1b;
  return resultado;
}

BYTE mulbyte(BYTE b1, BYTE b2) {
  BYTE resultado ;
  BYTE parc = b1;
  int i;
  resultado = b1 * ( b2 & 1 ) ;
  for(i=1;i<8;i++) {
    parc = vezesx( parc);
    resultado = resultado ^ ( parc * ( (b2>>i) & 1 ) );
  }
  return resultado;
}

struct state MixColumns(struct state st) {
  struct state stl;
  int i;
  for(i=0;i<4;i++) {
    stl.w[0].ucw[i] = mulbyte(0x02,st.w[0].ucw[i])^mulbyte(0x03,st.w[1].ucw[i])^st.w[2].ucw[i]^st.w[3].ucw[i];
    stl.w[1].ucw[i] = mulbyte(0x02,st.w[1].ucw[i])^mulbyte(0x03,st.w[2].ucw[i])^st.w[3].ucw[i]^st.w[0].ucw[i];
    stl.w[2].ucw[i] = mulbyte(0x02,st.w[2].ucw[i])^mulbyte(0x03,st.w[3].ucw[i])^st.w[0].ucw[i]^st.w[1].ucw[i];
    stl.w[3].ucw[i] = mulbyte(0x02,st.w[3].ucw[i])^mulbyte(0x03,st.w[0].ucw[i])^st.w[1].ucw[i]^st.w[2].ucw[i];
  }
  return stl;
} /*unsigned char SBox(unsigned char byte) { #define bit(byte,x)
((byte)&(unsigned char)(1 << (x)))
```

```
  unsigned char b=0;
  int i;
  for(i=0;i<8;i++)
    b^=bit(byte,i)^bit(byte,(i+4)%8)^bit(byte,(i+5)%8)^bit(byte,(i+6)%8)^bit(byte,(i+7)%8)^((0x63>>i)&1);
  return b;
}

unsigned char MDC(unsigned char a, unsigned char b) {
  unsigned char temp;
  printf("\n%i,%i",a,b);
  while(b) {
    temp = a % b;
    a = b;
    b = temp;
    printf("\n%i,%i",a,b);
  }
  return a;
} */

BYTE sbox[] = {
 0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
 0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
 0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
 0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,
 0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,
 0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,
 0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,
 0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,
 0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,
 0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,
 0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,
 0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,
 0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,
 0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,
 0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,
 0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16
};

BYTE Rcon(int pot) {
  BYTE temp = 0x01;
  int i;
  for( i=1;i<pot;i++ )
    temp = vezesx(temp);
  return temp;
}

void ExpandirChave( BYTE Chave[4*Nk], union word *w) {

  union word temp;
  int i,j=0;
  //carrega os primeiros 4 caracteres da chave na chave de rodada
  //mais especificamente a primeira palavra (word)
  for(i=0;i<Nk;i++)
    for(j=0;j<4;j++)
      w[i].ucw[3-j]=Chave[4*i+j];
  while(i<Nb*(Nr+1)) {
    temp.ulw=w[i-1].ulw;
    if (! (i%Nk) ) {
      //Aplica RotWord
      temp.ulw = RE(temp.ulw,1);
      //aplica SBox
```

```
      for(j=0;j<4;j++)
    temp.ucw[j] = sbox[ temp.ucw[j] ] ;
      //Aplica RCon
      temp.ucw[3]^=Rcon(i/Nk);
    }
    else
      if ( (Nk>6) && ( (i%Nk) ==4 ) )
    temp.ulw = sbox[temp.ulw];
    w[i].ulw = w[i-Nk].ulw ^ temp.ulw;
    i++;
  }
}
```

aes.h

```
#include<conio.h>
#include<stdio.h>
#include"aes.hpp"

main(int argc, void *argv[]) {
  FILE *Input;
  FILE *Key;
  FILE *Out;
  BYTE Entrada[16];
  BYTE Chave[16];
  BYTE temp[2];
  union word RK[Nb*(Nr+1)];
  state st,sttemp;
  int i,j,round;
  if(argc<3){
    printf("Formato AES ArqTextoPlano ArqSenha.");
    return 1;
  }
  if( (Input=fopen((char *) argv[1],"r" ))==NULL ) {
    printf("Erro ao abrir arquivo %s.",argv[1]);
    return 2;
  }
  if( (Key=fopen((char *) argv[2],"r" ))==NULL ) {
    printf("Erro ao abrir arquivo %s.",argv[2]);
    return 3;
  }
  if( (Out=fopen("Estado.txt","wt+" ))==NULL ) {
    printf("Erro ao criar arquivo Estado.txt.");
    return 4;
  }
  fclose(Out);
  if( (Out=fopen("ExpChave.txt","wt+" ))==NULL ) {
    printf("Erro ao criar arquivo ExpChave.txt.");
    return 4;
  }
  fclose(Out);

  //Carrega o arquivo de Texto Plano na Variavel Entrada
  for(i=0;i<16;i++){
    fscanf(Input,"%2s ",&temp);
    Entrada[i]= temp[0]>='a'?temp[0]-'a'+10:temp[0]-'0';
    Entrada[i]<<=4;
    Entrada[i]|= temp[1]>='a'?temp[1]-'a'+10:temp[1]-'0';
  }
  //Carrega o arquivo de chave na Variavel chave
```

```
for(i=0;i<16;i++){
  fscanf(Key,"%2s ",&temp);
  Chave[i]= temp[0]>='a'?temp[0]-'a'+10:temp[0]-'0';
  Chave[i]<<=4;
  Chave[i]|= temp[1]>='a'?temp[1]-'a'+10:temp[1]-'0';
}
//Inicia processo de cifragem
ExpandirChave(Chave,RK);
//state = in
//regra state[lin,col] = entrada[lin+4*col)
for(i=0;i<4;i++)
  for(j=0;j<4;j++)
    STATE(st,i,j) = Entrada[i + 4*j];
ImpMsg("Inicio do Processo de Cifrar");
ImpMsg("");
ImpState(st,"Texto Aberto");
ImpChave(Chave);
ImpRound(0);
ImpState(st,"Entrada");
//Adicionar Chave de Rodada
st = Adicionar_Chave_de_Rodada(st,RK,0);
ImpChaveRodada(RK,0);
//Rodadas
for( round = 1; round < Nr; round++ ) {
  ImpRound(round);
  ImpState(st,"Estado Inicial");
  st = AplicaSbox(st);
  ImpState(st,"Apos Caixa-S");
  st = DeslocaLinha(st);
  ImpState(st,"Apos Deslocamento Linha");
  st = MixColumns(st);
  ImpState(st,"Apos MixColumns");
  ImpChaveRodada(RK,round);
  st = Adicionar_Chave_de_Rodada(st,RK,round);
  ImpState(st,"Apos Chave Rodada");
}
ImpRound(round);
ImpState(st,"Estado Inicial");
st = AplicaSbox(st);
ImpState(st,"Apos Caixa-S");
st = DeslocaLinha(st);
ImpState(st,"Apos Deslocamento Linha");
ImpChaveRodada(RK,round);
st = Adicionar_Chave_de_Rodada(st,RK,round);
ImpState(st,"Apos Chave Rodada");
ImpMsg("");
ImpState(st,"Texto Cifrado");
ImpMsg("Fim Processo de Cifrar");
//Fim do cifrador  st contm o estado cifrado do texto plano
//Inicio do decifrador
ImpMsg("");
ImpMsg("Inicio do Processo de Decifrar");
ImpMsg("");
ImpState(st,"Texto Cifrado");
ImpRound(10);
ImpState(st,"Entrada");
//Adicionar Chave de Rodada
st = Adicionar_Chave_de_Rodada(st,RK,Nr);
ImpChaveRodada(RK,Nr);
//Rodadas
for( round = Nr-1; round >0; round-- ) {
```

```
    ImpRound(round);
    ImpState(st,"Estado Inicial");
    st = InvDeslocaLinha(st);
    ImpState(st,"Apos Deslocamento Linha Inverso");
    st = AplicaInvSbox(st);
    ImpState(st,"Apos Caixa-S Inversa");
    ImpChaveRodada(RK,round);
    ImpState(st,"Apos Chave Rodada");
    st = Adicionar_Chave_de_Rodada(st,RK,round);
    st = InvMixColumns(st);
    ImpState(st,"Apos MixColumns Inversa");
  }
  ImpRound(round);
  ImpState(st,"Estado Inicial ");
  st = InvDeslocaLinha(st);
  ImpState(st,"Apos Deslocamento de Linha Inverso");
  st = AplicaInvSbox(st);
  ImpState(st,"Apos Caixa-S Inversa");
  ImpChaveRodada(RK,round);
  st = Adicionar_Chave_de_Rodada(st,RK,round);
  ImpMsg("");
  ImpState(st,"Texto Aberto");
  ImpMsg("Fim do Processo de Decifrar");
  fclose(Input);
  fclose(Key);
  return 0;
}
```

aes.hpp

```
#include<stdio.h>
#define Nk 4
#define Nb 4
#define Nr 10

#define STATE(st,r,c) st.w[r].ucw[3-c]

#define BYTE unsigned char

union word {
  unsigned long ulw;
  BYTE ucw[4];
};
//rotacao a esquerda <<
#define RE(x,i)  ((x)<<((i)*8))|((x)>>((sizeof(x)-(i))*8))
//rotacao a direita
#define RD(x,i)  ((x)>>((i)*8))|((x)<<((sizeof(x)-(i))*8))

struct state {
  union word w[4];
};

BYTE vezesx(BYTE byte){
  BYTE resultado = byte;
  //Polinomio irredutivel utilizado (m)
  //x^8+x^4+x^3+x+1 ou {1 0001 1011} ou {01} {1b}
  //objetivo reduzir o polinomio ao grau 7 (1 byte)
  //byte pode ser representado por
  //b7*x^7+b6*x^6+b5*x^5+b4*x^4+b3*x^3+b2*x^2+b1*x^1+b0
  //que vezes x da
```

```
  //b7*x^8+b6*x^7+b5*x^6+b4*x^5+b3*x^4+b2*x^3+b1*x^2+b0*x^1
  //que nada mais  que byte deslocado de 1 bit a esquerda
  if( ! (byte & 0x80) ) //se b7 = 0 o polinomio ja esta reduzido
    resultado = byte << 1;
  else //se b7 diferente de 0,  necessario aplicar o modulo com m
  //que nada mais  que um xor
    resultado = (byte << 1) ^ 0x1b;
  return resultado;
}

BYTE mulbyte(BYTE b1, BYTE b2) {
  BYTE resultado ;
  BYTE parc = b1;
  int i;
  resultado = b1 * ( b2 & 1 ) ;
  for(i=1;i<8;i++) {
    parc = vezesx( parc);
    resultado = resultado ^ ( parc * ( (b2>>i) & 1 ) );
  }
  return resultado;
}

struct state MixColumns(struct state st) {
  struct state stl;
  int i;
  for(i=0;i<4;i++) {
    stl.w[0].ucw[i] = mulbyte(0x02,st.w[0].ucw[i])^mulbyte(0x03,st.w[1].ucw[i])^st.w[2].ucw[i]^st.w[3].ucw[i];
    stl.w[1].ucw[i] = mulbyte(0x02,st.w[1].ucw[i])^mulbyte(0x03,st.w[2].ucw[i])^st.w[3].ucw[i]^st.w[0].ucw[i];
    stl.w[2].ucw[i] = mulbyte(0x02,st.w[2].ucw[i])^mulbyte(0x03,st.w[3].ucw[i])^st.w[0].ucw[i]^st.w[1].ucw[i];
    stl.w[3].ucw[i] = mulbyte(0x02,st.w[3].ucw[i])^mulbyte(0x03,st.w[0].ucw[i])^st.w[1].ucw[i]^st.w[2].ucw[i];
  }
  return stl;
}

struct state InvMixColumns(struct state st) {
  struct state stl;
  int i;
  for(i=0;i<4;i++) {
    stl.w[0].ucw[i] = mulbyte(0x0e,st.w[0].ucw[i])^mulbyte(0x0b,st.w[1].ucw[i])^mulbyte(0x0d,st.w[2].ucw[i])^mulbyte(0x09,st.w[3].ucw[i]);
    stl.w[1].ucw[i] = mulbyte(0x0e,st.w[1].ucw[i])^mulbyte(0x0b,st.w[2].ucw[i])^mulbyte(0x0d,st.w[3].ucw[i])^mulbyte(0x09,st.w[0].ucw[i]);
    stl.w[2].ucw[i] = mulbyte(0x0e,st.w[2].ucw[i])^mulbyte(0x0b,st.w[3].ucw[i])^mulbyte(0x0d,st.w[0].ucw[i])^mulbyte(0x09,st.w[1].ucw[i]);
    stl.w[3].ucw[i] = mulbyte(0x0e,st.w[3].ucw[i])^mulbyte(0x0b,st.w[0].ucw[i])^mulbyte(0x0d,st.w[1].ucw[i])^mulbyte(0x09,st.w[2].ucw[i]);
  }
  return stl;
}


/*unsigned char SBox(unsigned char byte) { #define bit(byte,x)
((byte)&(unsigned char)(1 << (x)))
  unsigned char b=0;
  int i;
  for(i=0;i<8;i++)
    b^=bit(byte,i)^bit(byte,(i+4)%8)^bit(byte,(i+5)%8)^bit(byte,(i+6)%8)^bit(byte,(i+7)%8)^((0x63>>i)&1);
  return b;
}

unsigned char MDC(unsigned char a, unsigned char b) {
  unsigned char temp;
  printf("\n%i,%i",a,b);
  while(b) {
    temp = a % b;
```

```
    a = b;
    b = temp;
    printf("\n%i,%i",a,b);
  }
  return a;
} */

BYTE sbox[] = {
 0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
 0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
 0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
 0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,
 0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,
 0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,
 0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,
 0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,
 0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,
 0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,
 0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,
 0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,
 0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,
 0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,
 0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,
 0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16
};

BYTE Invsbox[] = {
0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb,
0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb,
0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e,
0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25,
0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92,
0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84,
0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06,
0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b,
0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73,
0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e,
0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b,
0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4,
0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f,
0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef,
0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61,
0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d
};

struct state AplicaSbox(struct state st) {
  int i,j;
  struct state temp;
  for(i=0;i<4;i++)
    for(j=0;j<4;j++)
      temp.w[i].ucw[j] = sbox[st.w[i].ucw[j]];
  return temp;
}

struct state AplicaInvSbox(struct state st) {
  int i,j;
  struct state temp;
  for(i=0;i<4;i++)
    for(j=0;j<4;j++)
      temp.w[i].ucw[j] = Invsbox[st.w[i].ucw[j]];
  return temp;
```

```
}

struct state DeslocaLinha(struct state st) {
  struct state temp;
  int i;
  for(i=0;i<4;i++)
    temp.w[i].ulw = RE(st.w[i].ulw,i);
  return temp;
}

struct state InvDeslocaLinha(struct state st) {
  struct state temp;
  int i;
  for(i=0;i<4;i++)
    temp.w[i].ulw = RD(st.w[i].ulw,i);
  return temp;
}

BYTE Rcon(int pot) {
  BYTE temp = 0x01;
  int i;
  for( i=1;i<pot;i++ )
    temp = vezesx(temp);
  return temp;
}

struct state Adicionar_Chave_de_Rodada(struct state st,union word
w[Nb*(Nr+1)], int rodada) {
  struct state aux;
  int i,j;
  for(i=0;i<4;i++)
    for(j=0;j<4;j++)
      STATE(aux,i,j) = STATE(st,i,j) ^ w[rodada*Nb+j].ucw[3-i];
  return aux;
}
//rotinas de Impress{\AE}o
void ImpState(struct state st,char str[20]) {
  int i,j;
  FILE *k=fopen("Estado.txt","at+");
  fprintf(k,"%34s: ",str);
  for(i=0;i<4;i++)
    for(j=0;j<4;j++)
      fprintf(k,"%02x ",STATE(st,i,j));
  fprintf(k,"\n");
  fclose(k);
} void ImpChaveRodada(union word w[Nb*(Nr+1)], int rodada) {
  int i,j;
  FILE *k=fopen("Estado.txt","at+");
  fprintf(k,"%34s: ","Chave de Rodada");
  for(i=0;i<4;i++)
    for(j=0;j<4;j++)
      fprintf(k,"%02x ",w[rodada*Nb+j].ucw[3-i]);
  fprintf(k,"\n");
  fclose(k);
}

void ImpRound(int round) {
  FILE *k=fopen("Estado.txt","at+");
  fprintf(k,"Rodada: %0i\n",round);
  fclose(k);
}
```

```
void ImpMsg(char str[]) {
  FILE *k=fopen("Estado.txt","at+");
  fprintf(k,"%s\n",str);
  fclose(k);
}

void ImpChave(BYTE Chave[16]) {
  int i;
  FILE *k=fopen("Estado.txt","at+");
  fprintf(k,"%34s: ","Chave");
  for(i=0;i<16;i++)
    fprintf(k,"%02x ",Chave[i]);
  fprintf(k,"\n");
  fclose(k);
}

//Impressao da Chave
void ImpChaveKey(BYTE Chave[16]) {
  int i;
  FILE *k=fopen("ExpChave.txt","at+");
  fprintf(k,"%s: ","Chave");
  for(i=0;i<16;i++)
    fprintf(k,"%02x ",Chave[i]);
  fprintf(k,"\n");
  fclose(k);
}
//Impress{\AE}o da Chave de Rodada
void ImpRoundKey(int round) {
  FILE *k=fopen("ExpChave.txt","at+");
  fprintf(k,"Rodada: %0i\n",round);
  fclose(k);
}

void ImpMsgKey(char str[]) {
  FILE *k=fopen("ExpChave.txt","at+");
  fprintf(k,"%s\n",str);
  fclose(k);
}

void ImpByteKey(BYTE b,char str[20]) {
  int i,j;
  FILE *k=fopen("ExpChave.txt","at+");
  fprintf(k,"%18s: ",str);
  fprintf(k,"%02x000000 ",b);
  fprintf(k,"\n");
  fclose(k);
}

void ImpWordKey(union word w,char str[20]) {
  int i,j;
  FILE *k=fopen("ExpChave.txt","at+");
  fprintf(k,"%18s: ",str);
  fprintf(k,"%08lx ",w.ulw);
  fprintf(k,"\n");
  fclose(k);
}

void ImpAllWordKey(union word w[Nb*(Nr+1)]) {
  int i,j;
  FILE *k=fopen("ExpChave.txt","at+");
```

```
  for(i=0;i<Nb*(Nr+1);i++) {
    if(i%4==0)
      fprintf(k,"Chave Rodada[%2i]\n",i/4);
    fprintf(k,"w[%2i]: ",i);
    fprintf(k,"%08lx ",w[i].ulw);
    fprintf(k,"\n");
  }
  fclose(k);
}
//Rotina para geracao da chave de rodada
void ExpandirChave( BYTE Chave[4*Nk], union word *w) {

  union word temp;
  int i,j=0;
  char texto[50];
  ImpChaveKey(Chave);
  //carrega os primeiros 4 caracteres da chave na chave de rodada
  //mais especificamente a primeira palavra (word)
  for(i=0;i<Nk;i++) {
    ImpRoundKey(i);
    for(j=0;j<4;j++)
      w[i].ucw[3-j]=Chave[4*i+j];
    sprintf(texto,"Carga Inicial w[%i]",i);
    ImpWordKey(w[i],texto);
  }
  while(i<Nb*(Nr+1)) {
    ImpRoundKey(i);
    temp.ulw=w[i-1].ulw;
    ImpWordKey(temp,"Temp");
    if (! (i%Nk) ) {
      //Aplica RotWord
      temp.ulw = RE(temp.ulw,1);
      ImpWordKey(temp,"Apos Rot Palavra");
      //aplica SBox
      for(j=0;j<4;j++)
    temp.ucw[j] = sbox[ temp.ucw[j] ] ;
      ImpWordKey(temp,"Apos Caixa-S");
      //Aplica RCon
      sprintf(texto,"Rcon[%i]",i/Nk);
      ImpByteKey(Rcon(i/Nk),texto);
      temp.ucw[3]^=Rcon(i/Nk);
      ImpWordKey(temp,"Apos Xor com Rcon");
    }
    else
      if ( (Nk>6) && ( (i%Nk) ==4 ) ) {
    temp.ulw = sbox[temp.ulw];
    ImpWordKey(temp,"Apos Caixa-S");
      }
    sprintf(texto,"w[%i]",i-Nk);
    ImpWordKey(w[i-Nk],texto);
    w[i].ulw = w[i-Nk].ulw ^ temp.ulw;
    sprintf(texto,"w[%i]=temp^w[%i]",i,i-Nk);
    ImpWordKey(w[i],texto);
    i++;
  }
  ImpMsgKey("");
  ImpMsgKey("Lista das Chaves de Rodada");
  ImpMsgKey("");
  ImpAllWordKey(w);
}
```

Apêndice B

Implementação em C do SAES

Neste anexo está contida uma implementação do cifrador SAES com fins didáticos através do qual foram gerados os exemplos utilizados nessa dissertação. Essa implementação não tem por finalidade o uso para cifrar arquivos nem operar em modos (CBC, CFB, etc.) apenas auxiliar no aprendizado do cifrador.

Instruções para o uso do SAES.exe:

- Criar um arquivo com duas linhas, onde a primeira linha corresponde à chave e a segunda ao texto aberto. Os valores devem ser especificados em notação decimal (valores possíveis: 0,1, 2, 3), por exemplo na primeira linha: 2 0 1 0 e na segunda linha: 2 0 0 1. O arquivo deve ser salvo como padrão ASCII.

- Executar via shell o seguinte comando SAES <arquivo com o texto aberto e Chave>, por exemplo SAES exemplo.txt .

O programa criará no mesmo diretório um arquivo com o nome SAES.txt, no formato ASCII, que contém passo a passo o processo de cifrar e decifrar assim como um arquivo chamado keyexpsi.txt, no formato ASCII puro, contendo o processo de geração de subchaves de rodadas.

SAES.cpp

```
#include<stdio.h>
#include<conio.h>
#define Nk 4
#define Nb 4
#define Nr 10

#define BYTE unsigned char

struct word {
  BYTE w[2];
};

struct state {
  word st[2];
};
//rotacao a esquerda <<
#define RE(x,i)  ((x)<<((i)*8))|((x)>>((sizeof(x)-(i))*8))
//rotacao a direita
#define RD(x,i)  ((x)>>((i)*8))|((x)<<((sizeof(x)-(i))*8))

/*struct state {
  union word w[4];
};*/

/*Matriz multiplicacao*/ /*Utilizada com o polinomio
caracteristico
  x^2+x+1*/
BYTE MUL[4][4] = { /*00*/ /*01*/ /*10*/ /*11*/
      /*00*/ {   0   ,  0   ,  0   ,  0   } ,
      /*01*/ {   0   ,  1   ,  2   ,  3   } ,
      /*10*/ {   0   ,  2   ,  3   ,  1   } ,
      /*11*/ {   0   ,  3   ,  1   ,  2   } };

/*Matriz Soma*/ BYTE ADD[4][4] = { /*00*/ /*01*/ /*10*/ /*11*/
      /*00*/ {   0   ,  1   ,  2   ,  3   } ,
      /*01*/ {   1   ,  0   ,  3   ,  2   } ,
      /*10*/ {   2   ,  3   ,  0   ,  1   } ,
      /*11*/ {   3   ,  2   ,  1   ,  0   } };
/*Esta matriz retorna o inverso de um numero definido como
 X * x^-1 (inverso) = 1 (resto X^2+x+1) */
BYTE INV[4] = { /*00*/ 0, /*01*/ 1, /*10*/ 3, /*11*/ 2 };

BYTE sbox[4] = { /*00*/ 2, /*01*/ 3, /*10*/ 1, /*11*/ 0 }; BYTE
invsbox[4] = { /*00*/ 3, /*01*/ 2, /*10*/ 0, /*11*/ 1 };


#define SOMA(x,y) ADD[x][y] #define MULT(x,y) MUL[x][y] #define
INVE(x) INV[x] #define SBOX(l,c) sbox[(l)*2+(c)]

void MULPOLI(BYTE a[], BYTE b[], BYTE resp[]) {
  resp[1] = MULT(a[0],b[1])^MULT(a[1],b[0]);
  resp[0] = MULT(a[0],b[0])^MULT(a[1],b[1]);
}

word RotWord( word A ) {
  word resp;
  resp.w[0] = A.w[1];
  resp.w[1] = A.w[0];
  return resp;
```

```
}

word SBoxKey( word A) {
  word resp;
  resp.w[0] = sbox[ A.w[0] ];
  resp.w[1] = sbox[ A.w[1] ];
  return resp;
}

word XOR( word A, word B ) {
  word resp;
  resp.w[0] = A.w[0]^B.w[0];
  resp.w[1] = A.w[1]^B.w[1];
  return resp;
}

void MostraWord( word A , char mens[]) {
  FILE *arq = fopen("KeyExpSi.txt", "a");
  fprintf( arq, "%25s %i %i\n", mens, A.w[0], A.w[1]);
  fclose(arq);
}

void InicializaKey() {
  FILE *arq = fopen("KeyExpSi.txt", "w");
  fclose(arq);
}

void MostraMensagem( char mens[], int i) {
  FILE *arq = fopen("KeyExpSi.txt", "a");
  if( i==-1 ) {
    fprintf( arq, "%s\n", mens);
  } else {
    fprintf( arq, "%s %2i\n", mens, i);
  }
  fclose(arq);
}

/*Numero de rodadas 3, com 12 valores ou 6 words*/

void GeraChaveRodada(BYTE chave[4], word k[6]) {
  word temp;
  word RCon;
  /*os primeiros 4 valores sao copiados da chave*/
  k[0].w[0] = chave[0];
  k[0].w[1] = chave[1];
  MostraMensagem("Rodada:", 0);
  MostraWord( k[0], "Carga Inicial w[0]:");
  k[1].w[0] = chave[2];
  k[1].w[1] = chave[3];
  MostraMensagem("Rodada:", 1);
  MostraWord( k[1], "Carga Inicial w[1]:");
  /*k[2]*/
  /*Calculo de temp
  temp = k[1]
  temp = SBOX( RotWord(temp) ) ^ Rcon - 1 */
  RCon.w[0] = 1;
  RCon.w[1] = 0;
  temp = k[1];
  MostraMensagem("Rodada:", 2);
  MostraWord( temp, "Temp:");
  temp = RotWord(temp);
```

```
  MostraWord( temp, "Apos Rot Palavra:");
  temp = SBoxKey( temp );
  MostraWord( temp, "Apos Rot Palavra:");
  MostraWord( RCon, "RCon[0]:");
  temp = XOR( temp, RCon );
  MostraWord( temp, "Apos XOR com RCon:");
  MostraWord( k[0], "w[0]:");
  k[2] = XOR( k[0], temp);
  MostraWord( k[2], "w[2]=Temp XOR w[0]:");
  /*k[3]*/
  MostraMensagem("Rodada:", 3);
  temp = SBoxKey( k[2] );
  MostraWord( temp, "Temp:");
  MostraWord( k[1], "w[1]:");
  k[3] = XOR( k[1], temp );
  MostraWord( k[3], "w[3]=Temp XOR w[1]:");
  /*k[4]*/
  MostraMensagem("Rodada:", 4);
  RCon.w[0] = 2;
  RCon.w[1] = 0;
  temp = k[3];
  MostraWord( temp, "Temp:");
  temp = RotWord(temp);
  MostraWord( temp, "Apos Rot Palavra:");
  temp = SBoxKey( temp );
  MostraWord( temp, "Apos Rot Palavra:");
  MostraWord( RCon, "RCon[0]:");
  temp = XOR( temp, RCon );
  MostraWord( temp, "Apos XOR com RCon:");
  MostraWord( k[2], "w[2]:");
  k[4] = XOR( k[2], temp);
  MostraWord( k[4], "w[4]=Temp XOR w[2]:");
  /*k[5]*/
  MostraMensagem("Rodada:", 5);
  temp = SBoxKey( k[4] );
  k[5] = XOR( k[3], temp );
  MostraWord( temp, "Temp:");
  MostraWord( k[3], "w[3]:");
  MostraWord( k[5], "w[5]=Temp XOR w[3]:");
}

void Inicializa() {
  FILE *arq = fopen("SAES.TXT","w");
  fclose(arq);
}

void ImprimeState( word st[2] , char mens[]) {
  FILE *arq = fopen("SAES.TXT","a");
  fprintf( arq, "%30s %i %i %i %i\n", mens,
      st[0].w[0],st[0].w[1],st[1].w[0],st[1].w[1]);
  fclose(arq);
}

void Imprime_Mensagem( char mens[], int i) {
  FILE *arq = fopen("SAES.TXT","a");
  if( i==-1 ) {
    fprintf( arq, "%s\n", mens);
  } else {
    fprintf( arq, "%s %2i\n", mens, i);
  }
    fclose(arq);
```

```
}

void Imprime_Word( char mens[], word k[6], int pi) {
  FILE *arq = fopen("SAES.TXT","a");
  fprintf( arq, "%30s %i %i %i %i\n", mens,
      k[pi].w[0],k[pi].w[1],k[pi+1].w[0],k[pi+1].w[1]);
  fclose(arq);
}

state XorRoundKey( state s, word k[6], BYTE or) {
  state resp;
  resp.st[0].w[0] = s.st[0].w[0]^k[or].w[0];
  resp.st[0].w[1] = s.st[0].w[1]^k[or].w[1];
  resp.st[1].w[0] = s.st[1].w[0]^k[or+1].w[0];
  resp.st[1].w[1] = s.st[1].w[1]^k[or+1].w[1];
  return resp;
}

state CaixaS( state s) {
  state resp;
  resp.st[0].w[0] = sbox[ s.st[0].w[0] ];
  resp.st[0].w[1] = sbox[ s.st[0].w[1] ];
  resp.st[1].w[0] = sbox[ s.st[1].w[0] ];
  resp.st[1].w[1] = sbox[ s.st[1].w[1] ];
  return resp;
}

state InvCaixaS( state s ) {
  state resp;
  resp.st[0].w[0] = invsbox[ s.st[0].w[0] ];
  resp.st[0].w[1] = invsbox[ s.st[0].w[1] ];
  resp.st[1].w[0] = invsbox[ s.st[1].w[0] ];
  resp.st[1].w[1] = invsbox[ s.st[1].w[1] ];
  return resp;
}

state ShiftRows( state s) {
  state resp;
  resp.st[0].w[0] = s.st[0].w[0];
  resp.st[0].w[1] = s.st[1].w[1];
  resp.st[1].w[0] = s.st[1].w[0];
  resp.st[1].w[1] = s.st[0].w[1];
  return resp;
}

word MulPoli( word b , word a){
  word resp;
  /*a[0]*b[1]^a[1]*b[0]*/
  resp.w[0] = MULT(a.w[1],b.w[0])^MULT(a.w[0],b.w[1]);
  /*a[0]*b[0]^a[1]*b[1]*/
  resp.w[1] = MULT(a.w[1],b.w[1])^MULT(a.w[0],b.w[0]);
  return resp;
}

state MixColumns( state s ) {
  state resp;
  /*Polinomio escolhido*/
  /*3*x+2*/
  word a = { { 3 , 1 } };
  resp.st[0] = MulPoli( s.st[0] , a);
  resp.st[1] = MulPoli( s.st[1] , a);
```

```
  return resp;
}

state InvMixColumns( state s ) {
  state resp;
  /*Polinomio escolhido*/
  /*2*x+3*/
  word a = { { 1 , 2 } };
  resp.st[0] = MulPoli( s.st[0] , a);
  resp.st[1] = MulPoli( s.st[1] , a);
  return resp;
}

void cifrador(BYTE chave[4], BYTE PT[4], BYTE CT[4], word k[6]) {
  state s;
  Imprime_Mensagem("Inicio do Processo de Cifrar", -1);
  Imprime_Mensagem("Rodada:", 0);
  GeraChaveRodada( chave, k);
  /*Carrega entrada*/
  s.st[0].w[0] = PT[0];
  s.st[0].w[1] = PT[1];
  s.st[1].w[0] = PT[2];
  s.st[1].w[1] = PT[3];
  ImprimeState(s.st, "Entrada:");
  Imprime_Word("Chave de Rodada:", k, 0);
  s = XorRoundKey( s, k, 0);
  ImprimeState(s.st, "Apos AdicionaChaveRodada:");
  Imprime_Mensagem("Rodada:", 1);
  ImprimeState(s.st, "Valor Inicial:");
  /*Inicio da primeiras rodada*/
  s = CaixaS( s );
  ImprimeState(s.st, "Apos Caixa-S:");
  s = ShiftRows( s );
  ImprimeState(s.st, "Apos DeslocamentoLinha:");
  s = MixColumns( s );
  ImprimeState(s.st, "Apos MixColumns:");
  Imprime_Word("Chave de Rodada:", k, 2);
  s = XorRoundKey( s, k, 2);
  ImprimeState(s.st, "Apos AdicionaChaveRodada:");
  Imprime_Mensagem("Rodada:", 2);
  ImprimeState(s.st, "Valor Inicial:");
  /*ultima rodada*/
  s = CaixaS( s );
  ImprimeState(s.st, "Apos Caixa-S:");
  s = ShiftRows( s );
  ImprimeState(s.st, "Apos DeslocamentoLinha:");
  Imprime_Word("Chave de Rodada:", k, 4);
  s = XorRoundKey( s, k, 4);
  ImprimeState(s.st, "Apos AdicionaChaveRodada:");
  CT[0] = s.st[0].w[0];
  CT[1] = s.st[0].w[1];
  CT[2] = s.st[1].w[0];
  CT[3] = s.st[1].w[1];
  Imprime_Mensagem("Fim do Processo de Cifrar\n",-1);
}

void decifrador(BYTE chave[4], BYTE CT[4], BYTE PT[4], word k[6])
{
  state s;
  /*Carrega entrada*/
  s.st[0].w[0] = CT[0];
```

```
  s.st[0].w[1] = CT[1];
  s.st[1].w[0] = CT[2];
  s.st[1].w[1] = CT[3];
  Imprime_Mensagem("Inicio do Processo de Decifrar", -1);
  Imprime_Mensagem("Rodada:", 0);
  ImprimeState(s.st, "Entrada:");
  s = XorRoundKey( s, k, 4);
  Imprime_Word("Chave de Rodada:", k, 0);
  ImprimeState(s.st, "Apos AdicionaChaveRodada:");
  /*Inicio da primeiras rodada*/
  Imprime_Mensagem("Rodada:", 1);
  ImprimeState(s.st, "Valor Inicial:");
  s = ShiftRows( s );
  ImprimeState(s.st, "Apos InvDeslocamentoLinha:");
  s = InvCaixaS( s );
  ImprimeState(s.st, "Apos InvCaixa-S:");
  s = XorRoundKey( s, k, 2);
  Imprime_Word("Chave de Rodada:", k, 2);
  ImprimeState(s.st, "Apos AdicionaChavedeRodada:");
  s = InvMixColumns( s );
  ImprimeState(s.st, "Apos InvMixColumn:");
  /*ultima rodada*/
  Imprime_Mensagem("Rodada:", 2);
  ImprimeState(s.st, "Valor Inicial:");
  s = ShiftRows( s );
  ImprimeState(s.st, "Apos InvDeslocamentoLinha:");
  s = InvCaixaS( s );
  ImprimeState(s.st, "Apos InvCaixa-S:");
  s = XorRoundKey( s, k, 0);
  Imprime_Word("Chave de Rodada:", k, 0);
  ImprimeState(s.st, "Apos AdicionaChave");
  Imprime_Mensagem("Fim do Processo de Decifrar", -1);
}

void main(int argc, void *argv[]) {
  clrscr();
  word k[6];
  BYTE chave[5];
  BYTE PT[5];
  int i;
  FILE *arq;
  arq = fopen((char *)argv[1],"r");
  //Leitura da chave
  for(i=0; i<4; i++)
    fscanf(arq, "%uc",&chave[i]);
  //leitura do texto aberto
  for(i=0; i<4; i++)
    fscanf(arq, "%uc",&PT[i]);

  fclose(arq);
  BYTE CT[4];
  Inicializa();
  InicializaKey();
  cifrador( chave, PT, CT, k );
  decifrador( chave, CT, PT, k );
}
```