UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GIOVANI DUCATTI RINALDI

# Análise do AES e Sua Criptoanálise Diferencial

Trabalho de Graduação.

Prof. Dr. Raul Fernando Weber
Orientador

Porto Alegre, junho de 2012.

# AGRADECIMENTOS

À minha família, pelo apoio, confiança e motivação.
Aos meus amigos e colegas, pela motivação.
Ao meu professor orientador Raul Fernando Weber, pela inspiração e paciência.
À UFRGS e, principalmente, ao Instituto de Informática, pelo excelente ensino.

# SUMÁRIO

**LISTA DE ABREVIATURAS E SIGLAS ..... 6**
**LISTA DE FIGURAS ..... 7**
**LISTA DE TABELAS ..... 9**
**RESUMO ..... 10**
**ABSTRACT ..... 11**
**1 INTRODUÇÃO ..... 12**
**1.1 Estrutura do trabalho ..... 12**
**2 *ADVANCED ENCRYPTION STANDARD* ..... 14**
**2.1 Histórico ..... 14**
**2.2 Noções Preliminares ..... 15**
2.2.1 Entrada e saída ..... 15
2.2.2 Matriz estado ..... 15
2.2.3 Representação numérica ..... 16
2.2.3.1 Adição e subtração ..... 16
2.2.3.2 Multiplicação e divisão ..... 16
2.2.3.3 Polinômios com coeficientes em GF($2^8$) ..... 17
**2.3 Transformações sobre o estado ..... 18**
2.3.1 Adição da chave de rodada ..... 18
2.3.2 Substituição de *bytes* ..... 18
2.3.3 Deslocamento de linhas ..... 20
2.3.4 Mistura de colunas ..... 21
**2.4 Escalonamento da chave ..... 22**
2.4.1 Rotação de palavra ..... 22
2.4.2 Substituição de palavra ..... 22
2.4.3 Constante de rodada ..... 22
2.4.4 Expansão da chave ..... 23
**2.5 Cifragem ..... 23**
**2.6 Decifragem ..... 24**
**2.7 Modos de operação ..... 25**
2.7.1 *Electronic Code Book (ECB)* ..... 26
2.7.2 *Cipher Block Chaining (CBC)* ..... 26
2.7.3 *Cipher Feedback (CFB)* ..... 27
2.7.4 *Output Feedback (OFB)* ..... 27
2.7.5 *Counter Mode (CTR)* ..... 28
**3 INTRODUÇÃO À CRIPTOANÁLISE E ATAQUES AO AES ..... 29**
**3.1 Ataque por força bruta ..... 29**
**3.2 Criptoanálise linear ..... 30**
**3.3 Criptoanálise diferencial ..... 30**
3.3.1 Diferencial Impossível ..... 31

3.3.2 Ataque *Boomerang* ..... 31
**3.4 Criptoanálise integral ..... 31**
**3.5 Ataques Algébricos ..... 32**
**3.6 Ataques *Meet In The Middle* ..... 32**
**4 Criptoanálise Diferencial do AES reduzido ..... 33**
**4.1 Notações ..... 33**
**4.2 Noções Preliminares ..... 34**
4.2.1 Análise Diferencial da *S-box* ..... 34
4.2.2 Relação entre subchaves ..... 35
4.2.3 Equivalência entre transformações de estado ..... 36
**4.3 Ataque ao AES de uma rodada ..... 38**
**4.4 Ataque ao AES de duas rodadas ..... 41**
4.4.1 Primeira Fase ..... 42
4.4.2 Segunda Fase ..... 48
**5 Simulador YAAES ..... 53**
**5.1 Especificações e visão geral da implementação ..... 53**
**5.2 YAAES *Inspector* ..... 54**
**5.3 YAAES *Attacker* ..... 59**
**6 CONCLUSÃO ..... 68**
**REFERÊNCIAS ..... 69**
**GLOSSÁRIO ..... 71**

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AES | Advanced Encryption Standard |
| ARK | Add Round Key |
| bit | Binary digit |
| CBC | Cipher Block Chaining |
| CFB | Cipher Feedback |
| CTR | Counter Mode |
| DES | Data Encryption Standard |
| ECB | Electronic Code Book |
| FIPS | Federal Information Processing Standards |
| GF | Galois Field |
| InvMC | Inverse Mix Columns |
| InvSB | Inverse Substitute Bytes |
| InvSR | Inverse Shift Rows |
| MC | Mix Columns |
| NIST | National Institute of Standards and Technology |
| Nb | Número de colunas de 4 *bytes* de um bloco de entrada/saída |
| Nk | Número de colunas de 4 *bytes* da chave |
| Nr | Número de rodadas |
| OFB | Output Feedback |
| Rcon | Round Constant |
| SB | Substitute Bytes |
| SPN | Substitution Permutation Network |
| SR | Shift Rows |
| XOR | Exclusive Or |
| YAAES | Yet Another AES |

# LISTA DE FIGURAS

Figura 2.1: Representação de um bloco de 16 *bytes*.....15
Figura 2.2: Multiplicação de dois polinômios.....17
Figura 2.3: Multiplicação matricial de dois polinômios.....17
Figura 2.4: Transformação afim da caixa de substituição.....18
Figura 2.5: Caixa de substituição.....19
Figura 2.6: Transformação afim inversa.....19
Figura 2.7: Caixa de substituição inversa.....20
Figura 2.8: Deslocamento de linhas.....20
Figura 2.9: Deslocamento de linhas inverso.....21
Figura 2.10: Mistura de colunas.....21
Figura 2.11: Pseudocódigo da cifragem do AES.....24
Figura 2.12: Pseudocódigo da cifragem inversa (decifragem) do AES.....24
Figura 2.13: Pseudocódigo da cifragem inversa equivalente do AES.....25
Figura 2.14: Cifragem no modo de operação ECB.....26
Figura 2.15: Cifragem no modo de operação CBC.....27
Figura 2.16: Cifragem no modo de operação CFB.....27
Figura 2.17: Cifragem no modo de operação OFB.....28
Figura 2.18: Cifragem no modo de operação CTR.....28

Figura 4.1: Exemplo de relação entre subchaves.....36
Figura 4.2: Cifragem do AES reduzido a uma rodada.....38
Figura 4.3: Primeira fase do ataque ao AES reduzido a duas rodadas.....42
Figura 4.4: Segunda fase do ataque ao AES reduzido a duas rodadas.....48

Figura 5.1: Parte de chamadas das funções da classe FastRijndael.....54
Figura 5.2: Tela principal do YAAES *Inspector*.....55
Figura 5.3: Entrada de dados do texto plano e chave.....55
Figura 5.4: Visualização da expansão de chave.....56
Figura 5.5: Cálculo da primeira coluna de uma subchave do AES 128.....56
Figura 5.6: Cálculo da segunda coluna de uma subchave do AES 128.....57
Figura 5.7: Operação de substituição de *bytes*.....57
Figura 5.8: Operação de deslocamento de *bytes*.....58
Figura 5.9: Operação de mistura de colunas.....58
Figura 5.10: Operação de adição de chave de rodada.....59
Figura 5.11: Entrada de dados para o YAAES *Attacker*.....59
Figura 5.12: Primeiro passo da primeira fase do ataque.....60
Figura 5.13: Segundo passo da primeira fase do ataque.....61
Figura 5.14: Terceiro passo da primeira fase do ataque.....62
Figura 5.15: Quarto passo da primeira fase do ataque.....63

Figura 5.16: *Bytes* conhecidos das subchaves .....63
Figura 5.17: Primeiro passo da segunda fase do ataque .....64
Figura 5.18: Segundo passo da segunda fase do ataque .....65
Figura 5.19: Terceiro passo da segunda fase do ataque .....66
Figura 5.20: Quarto passo da segunda fase do ataque .....67
Figura 5.21: Verificação de dados do ataque .....67

# LISTA DE TABELAS

Tabela 4.1: Análise dos quatro primeiros bytes de $k_0$ .....41
Tabela 4.2: Análise da primeira coluna da *S-box* da segunda rodada dos pares 1 e 2.....44
Tabela 4.3: Análise da primeira coluna da *S-box* da segunda rodada dos pares 1 e 3.....45
Tabela 4.4: Intersecção dos valores para a primeira coluna da *S-box* .....45
Tabela 4.5: Análise completa da terceira coluna da *S-box* da segunda rodada .....50

# RESUMO

Desde seu estabelecimento como novo padrão de cifragem com chave simétrica, o algoritmo AES tem sido um objetivo importante para várias técnicas de criptoanálise, muitas dessas explorando versões com rodadas reduzidas e buscando encontrar novas fraquezas. Dentre estas técnicas, a criptoanálise diferencial tem um importante papel por ser uma base sólida para o desenvolvimento de novos métodos, e o seu entendimento se tornou essencial para qualquer criptoanalista.

Neste trabalho, uma breve explicação sobre o algoritmo, assim como algumas critptoanálises recentes, do AES são apresentadas, seguidas por uma visão geral de um ataque à versão de uma rodada, e um exemplo completo de um ataque à versão de duas rodadas, onde ambos ataques são baseados na criptoanálise diferencial. Dois programas interativos foram desenvolvidos, sendo um para auxiliar no ensino do algoritmo AES e o outro para facilitar, de certa maneira, o entendimento de uma dentre as várias técnicas de criptoanálise diferencial.

**Palavras-chave:** criptologia, criptografia, criptoanálise, criptoanálise diferencial, AES, Advanced Encryption Standard.

**AES Analysis and Its Differential Cryptanalysis**

# ABSTRACT

Since its establishment as the new symmetric-key encryption standard, the AES algorithm has been an important objective of multiple cryptanalysis techniques, many of them exploring round-reduced variants and aiming at finding new weaknesses. Amongst these techniques, differential cryptanalysis plays an important role for being a solid basis in the development of new methods, and its understanding has become essential to any cryptanalyst.

In this work, a brief explanation about the algorithm and some recent successful cryptanalysis of AES is given, followed by the overview of a one round-reduced attack, and a full example of a two round-reduced variant attack, where both attacks are based in differential cryptanalysis. Two interactive applications have been developed, one to support the teaching of the AES algorithm and the other to facilitate, somehow, the understanding of one of its many differential cryptanalysis techniques.

**Keywords:** cryptology, cryptography, cryptanalysis, differential cryptanalysis, AES, Advanced Encryption Standard.

# 1 INTRODUÇÃO

Desde o início da era da computação, e, principalmente, com a popularização da *internet*, a segurança da informação tornou-se um campo de estudo bastante explorado, tanto academicamente quanto profissionalmente. Na comunicação virtual, a criptologia abrange métodos para garantir privacidade e confiabilidade na troca de informação, através da criptografia, enquanto a criptoanálise busca falhas em tais métodos que possam expor suas fraquezas.

Dentro do ramo da criptografia, pode-se dividir este em dois grupos, de acordo com o tipo de chave utilizada, pública ou simétrica. Nesta última, encontram-se os padrões estabelecidos pelo governo americano, nomeados de *Data Encryption Standard* (DES) e, após provado que este já não poderia mais ser considerado seguro, o *Advanced Encryption Standard* (AES), padrão em utilização até o presente momento.

Este trabalho foi desenvolvido com o intuito de realizar uma análise do algoritmo AES, visando prover um embasamento do mesmo, tanto matemático quanto funcional. A criptoanálise deste algoritmo será, então, o segundo assunto a ser abordado por este trabalho. Uma visão geral sobre métodos e ataques que obtiveram maior sucesso em explorar fraquezas do AES é apresentada, tal como métodos aplicados desde o padrão DES, como as criptoanálises linear e diferencial, e outros ataques, desenvolvidos especificamente para o AES.

Após, dois dentre os vários ataques publicados para o AES, que utilizam em específico propriedades da criptoanálise diferencial assim como características do mesmo, são apresentados. Visando a praticidade da realização dos métodos de ataque, ambos são realizados sobre versões reduzidas no que tange ao número de rodadas do AES.

De modo a exemplificar interativamente o processo de cifragem do AES e o ataque à sua versão reduzida a duas rodadas, dois simuladores foram desenvolvidos, nomeados de *YAAES Inspector* e *YAAES Attacker*.

## 1.1 Estrutura do trabalho

Este trabalho é dividido em seis capítulos, sendo este o primeiro.

O capítulo 2, intitulado *Advanced Encryption Standard*, apresenta o funcionamento do algoritmo AES para os processos de cifragem e decifragem, assim como seus modos de operação.

O capítulo 3 introduz uma gama de criptoanálises publicadas para o AES, dando uma visão geral sobre os ataques que obtiveram maior sucesso até o presente momento.

O capítulo 4 é o tema central deste trabalho, apresentando dois ataques ao AES reduzido a uma e a duas rodadas, ambos baseados na criptoanálise diferencial.

O capítulo 5 demonstra visual e textualmente o funcionamento de dois simuladores desenvolvidos para este trabalho, que visam explicar, de forma didática, o AES e o ataque a versão reduzida a duas rodadas deste algoritmo.

O capítulo 6 conclui este trabalho.

# 2 ADVANCED ENCRYPTION STANDARD

O *Advanced Encryption Standard* (AES) é o padrão de criptografia com chave simétrica adotado desde o ano de 2002 pelo *National Institute of Standards and Technology* (NIST) [FIPS PUB 197, 2001], agência governamental dos Estados Unidos, visando a substituição do *Data Encryption Standard* (DES), padrão até então utilizado desde 1976.

Nas seções que seguem, um breve histórico é apresentado, assim como o funcionamento do algoritmo de cifragem e decifragem do AES.

## 2.1 Histórico

O processo de escolha teve início em janeiro de 1997 quando o NIST veio a público anunciar a necessidade de um algoritmo para sucessão do DES, pedindo, a quem pudesse interessar, opiniões sobre como deveria ser realizado este processo de seleção. Em setembro daquele ano, a agência decide por abrir uma chamada pública por novos algoritmos, devendo estes suportarem blocos de tamanho de 128 bits, e chaves de 128, 192 e 256 bits. Inicialmente, a chamada também exigia que o novo padrão suportasse blocos de tamanho de 192 e 256 bits, porém este requisito foi retirado logo antes da publicação da chamada. Apesar disso, algumas propostas mantiveram este suporte, como uma funcionalidade extra, tal como RC6 e Rijndael.

Na primeira etapa, quinze candidatos foram submetidos. Dentre os aspectos analisados, estavam, além da segurança contra criptoanálise, o desempenho sobre diferentes arquiteturas e processadores, tal como o Intel Pentium II de 32 bits e *smart cards* de 8 bits [SCHNEIER, B., 1999].

Desde agosto de 1998, duas conferências foram realizadas, e em agosto de 1999 foram escolhidos cinco algoritmos finalistas: MARS, RC6, Rijndael, Serpent e Twofish. Quanto ao *design* de funcionamento, MARS, RC6 e Twofish utilizam uma rede Feistel, estrutura de cifragem utilizada pelo DES, enquanto Rijndael e Serpent utilizam uma rede de substituição e permutação (SPN).

Em outubro de 2000, o NIST vem a público anunciar que o Rijndael, sem necessidade de alterações à sua proposta inicial, havia sido selecionado como o novo padrão de criptografia de chave simétrica a ser adotado como sucessor do DES.

Criado por dois criptógrafos belgas, Joan Daemen e Vincent Rijmen, a cifra Rijndael, nome oriundo da junção de seus sobrenomes, adotada como o novo padrão de criptografia de blocos com chave simétrica, é descrita em detalhes nas seções que seguem.

## 2.2 Noções Preliminares

A seguir são apresentadas algumas noções básicas sobre o embasamento matemático, notações e especificação do AES.

### 2.2.1 Entrada e saída

No AES, tanto a entrada e a saída de dados de texto consistem em sequências de 128 *bits*, denominadas blocos, onde cada *bit* ocupa uma posição *n* tal que 0 <= *n* < 128. Para facilitar a visualização dos dados, cada conjunto de oito *bits* é representado como um *byte*, sendo o primeiro *bit*, dentre os 128, o *bit* mais significativo do primeiro *byte* do bloco, e o *bit* 127 sendo o *bit* menos significativo do último *byte* do bloco.

Deste modo, tem-se 16 *bytes*, desde o *byte* de número 00 ao *byte* de número 15. Novamente, para facilitar a representação dos dados, estes são distribuídos em quatro partes, cada parte contendo 32 *bits*, ou 4 *bytes*, formando assim quatro colunas de quatro *bytes* nos blocos de entrada e saída, como mostra a figura 2.1 abaixo.

Pode-se também referenciar cada *byte* de um bloco por dois índices, um para a linha e outro para a coluna nas quais aquele se encontra. Dado que um bloco de 16 *bytes* é representado em uma matriz de duas dimensões, tem-se assim quatro linhas e quatro colunas. Um dado *byte* $b_{l,c}$, por exemplo, está localizado na linha *l* e coluna *c*, sendo que 0 <= *l* < 4 e 0 <= *c* < 4. Assim, o primeiro *byte* de um bloco pode ser referenciado por sua posição ordinal no bloco (*byte* 00), assim como pela linha e coluna que ocupa na matriz do bloco, $b_{0,0}$.

| | | | | |
|---|---|---|---|---|
| Linha 0 | 00 | 04 | 08 | 12 |
| Linha 1 | 01 | 05 | 09 | 13 |
| Linha 2 | 02 | 06 | 10 | 14 |
| Linha 3 | 03 | 07 | 11 | 15 |
| | Coluna 0 | Coluna 1 | Coluna 2 | Coluna 3 |

Fig. 2.1: Representação de um bloco de 16 *bytes*

Diferentemente da entrada e saída de texto, três tamanhos são aceitos para a chave do AES: 128, 192 e 256 *bits*.

### 2.2.2 Matriz estado

A matriz estado, como o nome sugere, representa internamente o estado dos dados sobre os quais são realizadas as operações de cifragem ou decifragem. O tamanho da matriz de estado é de 128 *bits*, dado que a entrada de dados do padrão AES somente aceita este tamanho.

### 2.2.3 Representação numérica

Todos *bytes* no AES são interpretados como elementos pertencentes a um corpo finito (*finite field* ou também *Galois field*), definido por GF($2^8$). Um corpo finito GF(*m*) é definido pelo seu número de elementos *m* de seu conjunto, onde $m = p^n$, para algum *p* inteiro primo e um *n* inteiro. O *p* é considerado a característica do corpo finito.

Como é possível representar os elementos de um corpo finito de diferentes maneiras, os autores do algoritmo escolheram por representá-lo como polinômios [DAEMEN, J., 1999]. Deste modo, um dado *byte* B, que consiste dos *bits* $b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7$, é considerado um polinômio com coeficientes pertencentes ao corpo finito GF(2):

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \qquad (2.1)$$

Por exemplo, um *byte* com valor hexadecimal igual a 57, ou valor binário igual a 01010111, corresponde ao polinômio abaixo:

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

#### *2.2.3.1 Adição e Subtração*

Para realizar operações de adição de dois números na representação polinomial definida acima, coeficientes com mesma potência de *x* são somados em módulo 2, onde os coeficientes representam os *bits* em um *byte*, e a operação de soma módulo 2 é a operação de XOR. A expressão abaixo representa a adição dos valores hexadecimais 0x57 e 0x83, resultando em de acordo com a notação polinomial:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

Esta operação de adição satisfaz as condições para se ter um grupo abeliano, pois é fechada, comutativa, associativa, tem um elemento neutro e para todo elemento existe um elemento inverso, isto é, que anula a adição do elemento. Como todo elemento é seu próprio elemento inverso, a adição e subtração são definidas da mesma forma.

#### *2.2.3.2 Multiplicação e Divisão*

Multiplicação de polinômios em GF($2^8$), representada aqui pelo símbolo *, corresponde à multiplicação polinomial em módulo com um polinômio irredutível de grau 8. Um polinômio é considerado irredutível se seus divisores são um e ele mesmo. A operação em módulo é necessária para que, além de ser associativa, comutativa e distributiva, a multiplicação seja uma operação fechada sobre o corpo finito GF($2^8$), reduzindo assim a ordem do polinômio resultante da multiplicação.

O polinômio irredutível escolhido pelos autores do AES é:

$$x^8 + x^4 + x^3 + x + 1 \qquad (2.2)$$

Por exemplo, a seguinte multiplicação 0x57 * 0x83 = 0xc1 é exemplificada em formato polinomial como segue:

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1)*(x^7 + x + 1) = \quad & x^{13} + x^{11} + x^9 + x^8 + x^7 + \\ & x^7 + x^5 + x^3 + x^2 + x + \\ & x^6 + x^4 + x^2 + x + 1 \\ = \quad & x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1\end{aligned}$$

Aplicando o módulo sobre o resultado da multiplicação, obtém-se:

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

A multiplicação definida acima tem como identidade multiplicativa o elemento, em formato hexadecimal, 0x01. Para qualquer polinômio b(x) diferente de zero e de grau menor que 8, existe um inverso multiplicativo de b(x), denotado por $b^{-1}(x)$, que pode ser encontrado, por exemplo, através do algoritmo de Euclides estendido, como demonstram os autores do AES em [DAEMEN, J., 2002-b].

*2.2.3.3 Polinômios com coeficientes em GF($2^8$)*

Ao se definir polinômios com coeficientes em um corpo finito GF($2^8$), tem-se um polinômio onde cada coeficiente das potências de *x* representam um *byte* ao invés de um *bit*. Ao limitar que o grau do polinômio seja menor que 4, pode-se representar um vetor de 4 *bytes* em formato polinomial.

Ambas operações de adição e multiplicação são aplicadas a estes polinômios. A adição utiliza operações de XOR, tal como a adição da seção 2.2.3.1, porém ao invés de operar sobre *bits*, opera sobre *bytes* diretamente. Para a multiplicação de polinômios com coeficientes em GF($2^8$), é definido um novo polinômio irredutível para a realização do módulo no resultado da multiplicação dos polinômios:

$$x^4 + 1 \tag{2.3}$$

Assim pode-se calcular o polinômio resultante d(x) da multiplicação modular de dois polinômios a(x) e b(x) aplicando as expressões da figura 2.2 abaixo.

$$d_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$$
$$d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3)$$
$$d_2 = (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3)$$
$$d_3 = (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)$$

Figura 2.2: Multiplicação de dois polinômios (FIPS 197, p. 13).

Caso a(x) seja um polinômio com coeficientes fixos, as operações acima podem ser escritas no formato de multiplicação matricial, onde a matriz à esquerda é uma matriz circulante, como demonstra a figura 2.3.

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

Figura 2.3: Multiplicação matricial de dois polinômios (FIPS 197, p. 13).

## 2.3 Transformações sobre o estado

Sobre a matriz estado são aplicadas transformações, que alteram o valor dos *bytes*, seja de forma linear ou não-linear. Nas seções que seguem, as quatro transformações do AES são apresentadas, assim como suas operações inversas.

### 2.3.1 Adição da chave de rodada

A transformação de adição da chave de rodada (*AddRoundKey)* consiste em modificar a matriz estado realizando um XOR *byte* a *byte* desta com a matriz da subchave da rodada em questão. Cada subchave é composta pelo mesmo número de *bytes* que a matriz estado, isto é, 16. Este conjunto de *bytes* é selecionado a cada rodada a partir da chave expandida, descrita na seção 2.4.

### 2.3.2 Substituição de bytes

A substituição de bytes (*SubBytes*) é a única transformação não linear do algoritmo AES. Ela consiste em aplicar uma caixa de substituição (*S-box*) em cada *byte* da matriz estado. A caixa de substituição é igual para todas as rodadas, e constitui de uma tabela bidimensional de valores, como mostra a figura 2.5.

Para a construção da tabela, os autores utilizaram o inverso multiplicativo do valor procurado, e então o utilizam como entrada para uma transformação afim, definida a seguir na figura 2.4.

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

Figura 2.4: Transformação afim da caixa de substituição (DAEMEN, J., 1999).

Para realizar a substituição de *bytes*, deve-se procurar pela intersecção da linha equivalente ao valor dos quatro *bits* mais significativos do *byte* e da coluna equivalente ao valor dos quatro *bits* menos significativos deste mesmo *byte*. Por exemplo, se um dado *byte* $s_{r,c}$ = 0xC3, procura-se, na figura 2.5, pela linha com índice de linha ‘C’ e índice de coluna ‘3’, obtendo assim o valor ‘2E’, isto é, 0x2E.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| a | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| b | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| c | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| d | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| e | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| f | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

Figura 2.5: Caixa de substituição (FIPS PUB 197, 2001).

A operação inversa à de substituição de *bytes* (*InvSubBytes*) consiste também em aplicar uma caixa de substituição sobre os *bytes* da matriz estado. Porém, para construir a tabela inversa, utiliza-se o inverso do mapeamento afim da figura 2.4, mostrada na figura 2.6, sobre o *byte* de entrada, e calcula-se o inverso multiplicativo do resultado. A tabela para substituição de *bytes* inversa é mostrada na figura 2.7.

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

Figura 2.6: Transformação afim inversa (DAEMEN, J., 2002-b).

| | 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 | 0c | 7d |

Figura 2.7: Caixa de substituição inversa (FIPS PUB 197, 2001).

### 2.3.3 Deslocamento de linhas

A transformação de descolamento de linhas (*ShiftRows)* consiste em uma transposição de deslocamento cíclico dos *bytes* da matriz estado, onde cada linha é deslocada por um número fixo, de acordo com a linha em questão.

Na primeira linha da matriz, o deslocamento não é realizado. Nas seguintes, é aplicado de acordo com a função *deslocamento(l),* onde *l* é o número da linha, demonstrada abaixo.

$$\text{deslocamento}(1) = 1;\ \text{deslocamento}(2) = 2;\ \text{deslocamento}(3) = 3; \qquad (2.4)$$

Pode-se representar tal deslocamento *byte* a *byte* de acordo com a equação abaixo, onde s indica o *byte* da matriz estado, *l* indica a linha, *c* a coluna, e *Nb* o número de colunas de quatro *bytes* da matriz estado.

$$s'_{l,c} = s_{l,\,(c + \text{deslocamento}(l)) \bmod Nb}, \text{ onde } 0 < \boldsymbol{l} < 4 \text{ e } 0 <= \boldsymbol{c} < \boldsymbol{Nb} \qquad (2.5)$$

Na figura que segue, pode-se visualizar o deslocamento de linhas sobre a matriz estado representada pelos *bytes* $s_{l,c}$.

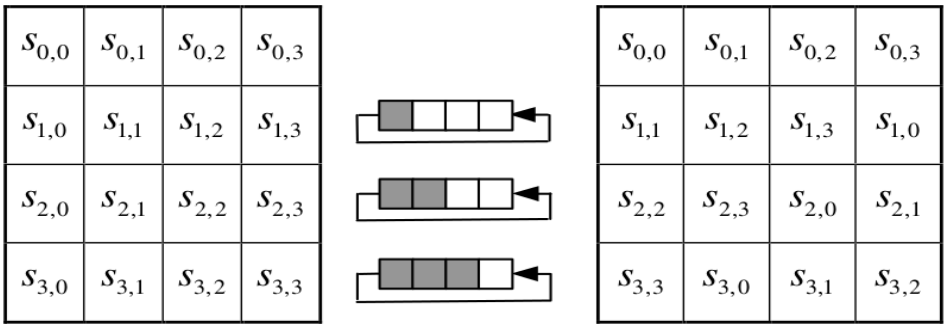


Figura 2.8: Deslocamento de linhas (FIPS PUB 197, 2001).

A transformação inversa do deslocamento de linhas (*InvShiftRows*) consiste em deslocar os *bytes* ciclicamente na ordem inversa, isto é, ao invés de deslocá-los de acordo com o resultado da função *deslocamento(l)* definida acima, subtrai-se do número de colunas do bloco, *Nb*, o resultado da função *deslocamento*(*l*).

Especificamente, o deslocamento de linhas inverso pode ser definido como segue, onde a figura 2.7 representa visualmente a operação sobre a matriz estado.

$$s'_{l,(c + \text{deslocamento}(l)) \bmod Nb} = s_{l,c}\text{, onde } 0 < l < 4 \text{ e } 0 <= c < Nb \qquad (2.6)$$

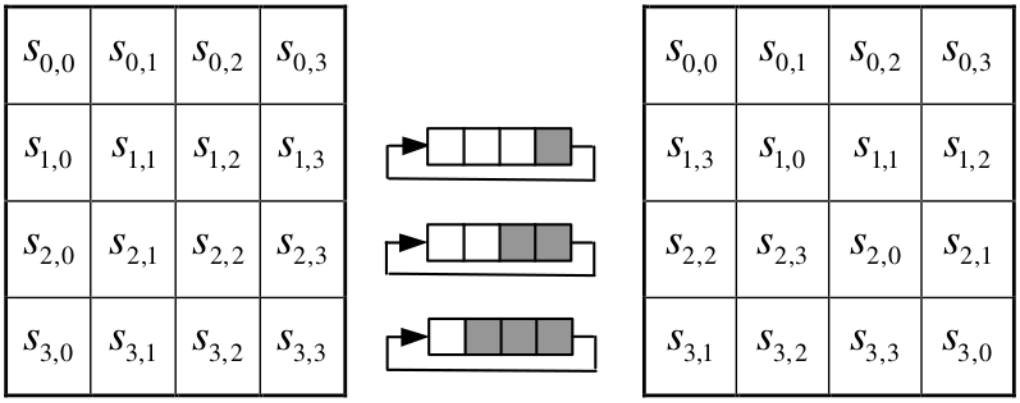


Figura 2.9: Deslocamento de linhas inverso (FIPS PUB 197, 2001).

### 2.3.4 Mistura de colunas

A transformação de mistura de colunas (MixColumns) é uma permutação linear e opera sobre as colunas da matriz de estado, tratando cada coluna como um polinômio de quatro termos sobre o corpo finito GF($2^8$). Então, este polinômio é multiplicado em módulo $x^4+1$ com um polinômio fixo *c(x)* escolhido pelos autores do algoritmo, definido como:

$$c(x) = 0x03*x^3 + 0x01*x^2 + 0x01*x + 0x02 \qquad (2.7)$$

Segundo [DAEMEN, J., 2002-b], este polinômio foi escolhido por ser coprimo do polinômio $x^4+1$ e, portanto, possuir inverso.

De acordo com a subseção 2.2.1.3, utilizando uma matriz circulante, tal multiplicação de polinômios pode ser escrita na forma matricial, como mostra a figura 2.10 abaixo.

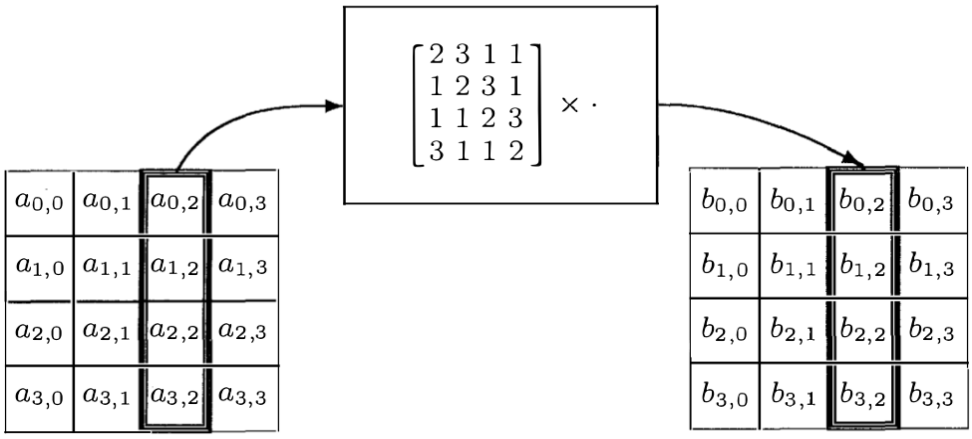


Figura 2.10: Mistura de colunas (DAEMEN, J., 2002-b).

A operação inversa da mistura de colunas (*InvMixColumns*) utiliza o polinômio inverso do polinômio *c(x)*, representado por *d(x)*, e pode ser definido de acordo com a expressão:

$$(0x03*x^3 + 0x01*x^2 + 0x01*x + 0x02) \text{ XOR } d(x) = 01.$$

O polinômio *d(x)* resultante é:

$$d(x) = 0x0B*x^3 + 0x0D*x^2 + 0x09*x + 0x0E \quad (2.8)$$

## 2.4 Escalonamento da chave

Como parte de sua entrada, o AES recebe uma chave para realizar o processo de cifragem ou decifragem. O tamanho da chave define a quantidade de rodadas que o AES efetuará sobre o bloco de entrada. Para isso, a chave deve passar pelo processo de escalonamento, que expande sua quantidade de *bits* para suportar um número *Nr* de rodadas.

Os três tamanhos de chaves aceitas pelo padrão AES são 128, 192 e 256 *bits*. Do mesmo modo que os dados de entrada e saída, representa-se a chave em *bytes*, no formato hexadecimal, separando-a em palavras de quatro *bytes*, obtendo assim uma visualização da chave com *Nk* colunas. Tem-se, então, para cada respectivo tamanho de chave, 16, 24 e 32 *bytes*, cada um tendo um valor *Nk* de colunas igual a 4, 6 e 8, respectivamente.

Além do conjunto de colunas *Nk* da chave passada como entrada, cada uma das *Nr* rodadas do AES requer *Nb* colunas de quatro *bytes* da chave expandida para realizar todas operações de adição de chave da rodada. Assim, a expansão da chave gera um total de *Nb*(Nr+1)* colunas de quatro *bytes*, cada uma indicada por $w_i$, onde $0 <= i < Nb*(Nr+1)$.

### 2.4.1 Rotação de palavra

A função rotação de palavra (*RotWord)* recebe como parâmetro de entrada uma palavra $[b_0,b_1,b_2,b_3]$, isto é, um vetor de 4 *bytes*, e aplica uma permutação cíclica, retornando um novo vetor de *bytes* $[b_1,b_2,b_3,b_0]$.

### 2.4.2 Substituição de palavra

A função de substituição de palavra (*SubWord*) recebe uma palavra em sua entrada, e, para cada um dos quatro *bytes* da palavra, aplica a caixa de substituição (Fig. 2.3), produzindo um novo vetor na saída.

### 2.4.3 Constante de Rodada

Visando eliminar a simetria entre as subchaves, valores constantes para cada rodada (*RC*) são gerados de acordo com a regra recursiva abaixo, definida sobre $GF(2^8)$:

$$RC[1] = x^0$$
$$RC[2] = x$$
$$RC[i] = x * RC[i - 1] = x^{i-1}, \text{ para todo } i > 2 \quad (2.9)$$

Os valores gerados por *RC* são utilizados para ocuparem o primeiro *byte* em uma palavra, definida pela variável *Rcon* abaixo:

$$Rcon[i] = (RC[i], 0x00, 0x00, 0x00)$$

### 2.4.4 Expansão da chave

A expansão da chave começa por copiar as primeiras *Nk* colunas da chave passada como entrada para a chave expandida. Para todas as colunas $w_i$ seguintes, realiza-se o XOR da coluna anterior $w_{i-1}$ com a coluna $w_{i-Nk}$, isto é, a coluna que antecede em *Nk* posições a coluna $w_i$.

Caso a coluna seja para uma posição *i* múltipla do valor de *Nk*, uma transformação deve ser aplicada a esta coluna antes de realizar o XOR descrito acima. Tal transformação consiste em realizar um deslocamento cíclico dos *bytes* (*RotWord*), seguido de uma operação de substituição dos *bytes* resultantes (*SubWord*), e então um XOR com uma constante de rodada (*Rcon*).

Deste modo, obtém-se a chave expandida quando a chave de entrada tem 128 ou 192 *bits*. Para 256 *bits*, ainda é necessário que, antes do XOR de $w_{i-1}$ com a coluna $w_{i-Nk}$ para gerar a coluna $w_i$, deve-se aplicar uma substituição de palavra (SubWord) na coluna $w_{i-1}$, caso *i* seja um múltiplo de *Nk*.

Tem-se, ao final do processo, na chave expandida resultante, um total de 1408, 1664 e 1920 *bits* para os tamanhos de chave de entrada de 128, 192 e 256 *bits*, respectivamente.

## 2.5 Cifragem

O processo de cifragem do AES consiste em, após realizar a expansão da chave, copiar a entrada de dados para a matriz estado, e aplicar transformações sobre esta matriz durante um número *r* de rodadas, onde $0 <= r <= Nr$. Lembrando que *r* é definido de acordo com o tamanho da chave passada na entrada, para tamanhos de chave com 128, 192 e 256 *bits*, tem-se $r = 10$, 12 e 14, respectivamente.

Na primeira rodada (também chamada de *whitening round*), isto é, onde $r = 0$, é realizada apenas uma operação de adição de chave de rodada à matriz estado. Nas rodadas que seguem, até a rodada $Nr - 1$, aplica-se as quatro transformações (*SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*) na matriz estado. Na última rodada, onde $r = Nr$, aplica-se todas as transformações tal como na transformação intermediária, com exceção da operação de mistura de colunas. Então, ao final, tem-se na matriz estado a saída do processo de cifragem.

O pseudocódigo na figura 2.11 demonstra o processo de cifragem do AES.

```
Cifragem(
        byte entrada[Nb*4],
        byte saida[Nb*4],
        byte chaveExpandida[(Nb*(Nr+1))*4]
        ){

    byte estado[Nb*4] = entrada;

    //Primeira rodada
    AddRoundKey(estado, chaveExpandida[0]);

    //Rodadas intermediárias
    para rodada = 1 até Nr-1, rodada++
    {
         SubBytes(estado);
         ShiftRows(estado);
         MixColumns(estado);
         AddRoundKey(estado, chaveExpandida[Nb*rodada*4]);
    }

    //Última rodada
    SubBytes(estado);
    ShiftRows(estado);
    AddRoundKey(estado, chaveExpandida[Nb*Nr*4]);

    saida = estado;
}
```

Figura 2.11: Pseudocódigo da cifragem do AES.

## 2.6 Decifragem

As transformações descritas no processo de cifragem na seção 2.5 podem ser invertidas e realizadas em ordem inversa, produzindo assim a cifragem inversa (decifragem) do AES. Deste modo, utiliza-se as operações de substituição de *bytes* inversa, deslocamento de linhas inversa, mistura de colunas inversa, e adição de chave de rodada, aplicando-as em ordem reversa ao processo de cifragem. As chaves de rodada também devem ser selecionadas na ordem inversa, isto é, para a primeira chave de rodada do processo de decifragem a ser selecionada é equivalente à última utilizada pelo processo de cifragem. Abaixo, o pseudocódigo exemplifica o processo.

```
CifragemInversa(
        byte entrada[Nb*4],
        byte saida[Nb*4],
        byte chaveExpandida[(Nb*(Nr+1))*4]
        ){

    byte estado[Nb*4] = entrada;

    //Primeira rodada
    AddRoundKey(estado, chaveExpandida[Nb*Nr*4]);

    //Rodadas intermediárias
    para rodada = Nr-1 até 1, rodada--
    {
         InvShiftRows(estado);
         InvSubBytes(estado);
         AddRoundKey(estado, chaveExpandida[Nb*rodada*4]);
         InvMixColumns(estado);
    }

    //Última rodada
    InvShiftRows(estado);
    InvSubBytes(estado);
    AddRoundKey(estado, chaveExpandida[0]);

    saida = estado;
}
```

Figura 2.12: Pseudocódigo da cifragem inversa (decifragem) do AES.

É possível ainda implementar a decifragem do AES mantendo a ordem das transformações sobre a matriz estado igual ao processo de cifragem, apenas utilizando suas operações inversas. De acordo com as propriedades algébricas do AES [DAEMEN, J., 2002-b], a ordem das operações de deslocamento de linhas inversa e substituição de *bytes* inversa comutam, e, portanto, sua ordem é indiferente no resultado da matriz estado. Quanto às outras duas operações, adição de chave de rodada e mistura de colunas inversa, estas podem ser invertidas em sua ordem de execução, desde que a chave de rodada seja adaptada para ser utilizada antes da transformação de mistura de colunas inversa. Este método de decifragem é chamado de cifragem inversa equivalente, por manter a ordem das transformações igual ao processo de cifragem, como pode-se verificar na figura 2.13 abaixo, demonstrando o processo em pseudocódigo.

```
CifragemInversaEquivalente(
        byte entrada[Nb*4],
        byte saida[Nb*4],
        byte chaveExpandidaModificada[(Nb*(Nr+1))*4]
        ){

   byte estado[Nb*4] = entrada;

   //Primeira rodada
   AddRoundKey(estado, chaveExpandidaModificada[Nb*Nr*4]);

   //Rodadas intermediárias
   para rodada = Nr-1 até 1, rodada--
   {
        InvSubBytes(estado);
        InvShiftRows(estado);
        InvMixColumns(estado);
        AddRoundKey(estado, chaveExpandidaModificada[Nb*rodada*4]);
   }

   //Última rodada
   InvSubBytes(estado);
   InvShiftRows(estado);
   AddRoundKey(estado, chaveExpandidaModificada[0]);

   saida = estado;
}
```

Figura 2.13: Pseudocódigo da cifragem inversa equivalente do AES.

## 2.7 Modos de Operação

Algoritmos de cifragem/decifragem de blocos, como o DES e o AES, operam somente sobre blocos de tamanho fixo. Quando se faz necessário encriptar dados com um tamanho maior de *bits* suportado por bloco, deve-se utilizar algum modo de operação, que possa garantir confidencialidade ou integridade da mensagem criptografada. Além disso, caso o tamanho da mensagem não seja um múltiplo exato do tamanho do bloco de entrada, é necessário realizar o *padding* da mensagem, isto é, completar o final da mensagem para que seu tamanho seja aceito pela cifra.

Nas seções que seguem, são apresentados 5 modos de operação, que visam prover confidencialidade na cifragem de dados maiores que um bloco. A entrada do algoritmo de cifragem é referenciada pela expressão texto plano (*plaintext*) e saída por texto cifrado (*ciphertext*).

### 2.7.1 *Electronic Code Book* (ECB)

A maneira mais simples de cifrar mensagens longas é utilizar o modo de operação Livro Eletrônico de Códigos (*Electronic Code Book*), onde a mensagem é divida em blocos de texto plano, e cada bloco é cifrado separadamente. A figura 2.14 ilustra o processo de cifragem.

Apesar de ser o mais simples e eficiente, em desempenho, dentre os modos de operação disponíveis, não é aconselhável utilizar o ECB [SP 800-38A, 2001]. A principal desvantagem deste modo é que, dados dois textos planos idênticos, a saída da cifragem de ambos será também idêntica, para uma mesma chave. Desta maneira, um atacante que esteja interceptando a troca de texto cifrado pode verificar facilmente repetições de padrão nas mensagens, o que é indesejável visto que a confidencialidade da comunicação cifrada é comprometida.

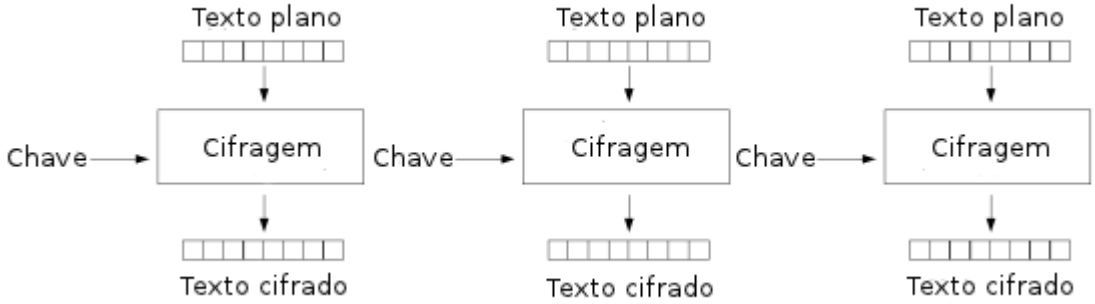


Figura 2.14: Cifragem no modo de operação ECB.

### 2.7.2 *Cipher Block Chaining* (CBC)

O modo de Encadeamento de Bloco Cifrado (*Cipher Block Chaining*) é um dos modos de operação mais utilizados por algoritmos de cifragem, e consiste em combinar ("encadear") os textos planos com os textos cifrados anteriormente, através de uma operação de XOR *byte* a *byte*. Assim, o problema de falta de confidencialidade do modo ECB é evitado, pois o texto plano acaba por ser alterado ("aleatorizado") a cada cifragem de bloco.

Como, antes da cifragem do primeiro bloco, não existe um texto cifrado disponível para se realizar a combinação com o texto plano, um vetor de inicialização (VI) é necessário para dar início ao processo do CBC. Tal vetor não precisa ser secreto, de acordo com [SP 800-38A, 2001], porém deve ser imprevisível e único durante uma troca de mensagens cifradas. Existem diversas maneiras de gerar um vetor de inicialização. Pode-se utilizar um valor fixo, um contador, um vetor aleatório, ou um *Nonce* (*Number used once*). Mais detalhes sobre geração de vetores de inicialização podem ser encontrados em [FERGUSON, N., 2010].

A figura 2.15 a seguir demonstra a cifragem no modo CBC.

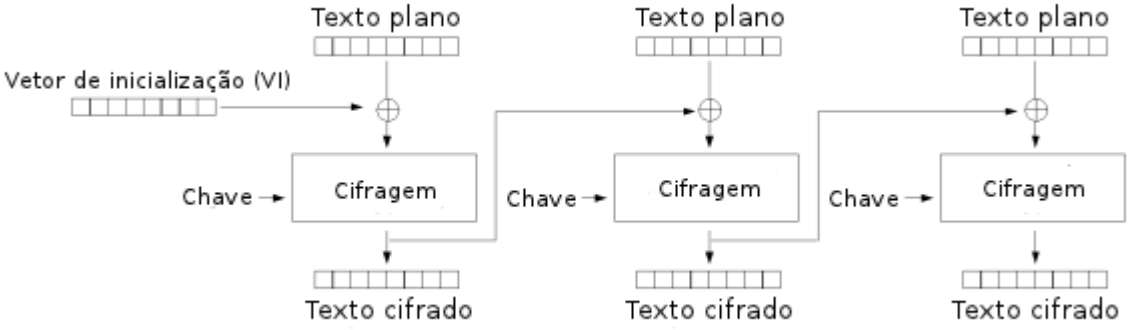


Figura 2.15: Cifragem no modo de operação CBC.

### 2.7.3 *Cipher Feedback* (CFB)

Diferentemente dos modos ECB e CBC, o modo de Realimentação da Cifragem (*Cipher Feedback*) é um modo de operação que utiliza a operação de cifragem para gerar uma saída pseudorrandômica de dados (chamada de fluxo da chave, do inglês *key stream*). Após a cifragem, é realizado um XOR da saída com o texto plano, gerando assim o texto cifrado, que realimenta o processo.

Uma vantagem em relação aos modos de operação anteriores é não necessitar o *padding* da mensagem, caso seu tamanho não for múltiplo do tamanho do bloco de entrada do AES.

A figura 2.16 abaixo demonstra o modo CFB.

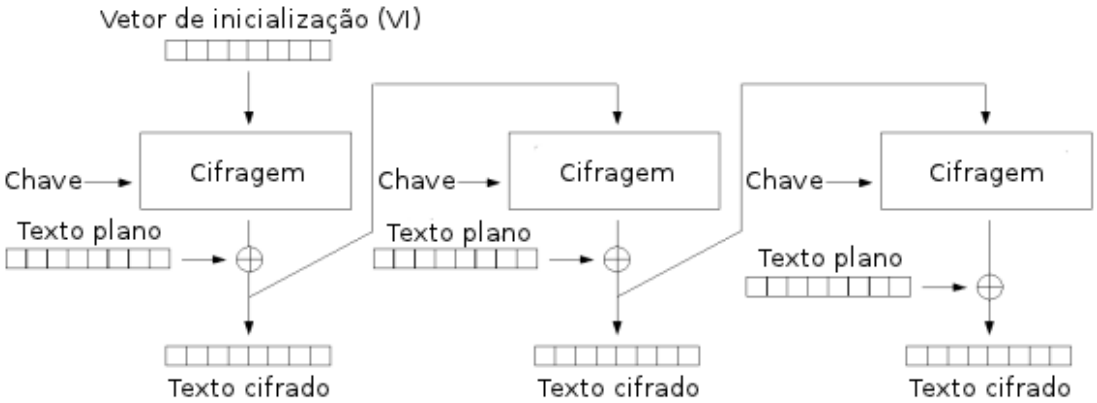


Figura 2.16: Cifragem no modo de operação CFB.

### 2.7.4 *Output Feedback* (OFB)

Muito semelhante ao modo de operação CFB, a Realimentação de Saída (*Output Feedback*) utiliza da mesma geração pseudorrandômica de dados da cifragem para gerar o primeiro fluxo, através de um vetor de inicialização. Porém, para realimentar o sistema de cifragem, utiliza a saída diretamente da cifragem do bloco, e não o texto cifrado, como acontece no modo CFB.

A seguir, na figura 2.17, pode-se ver o processo.

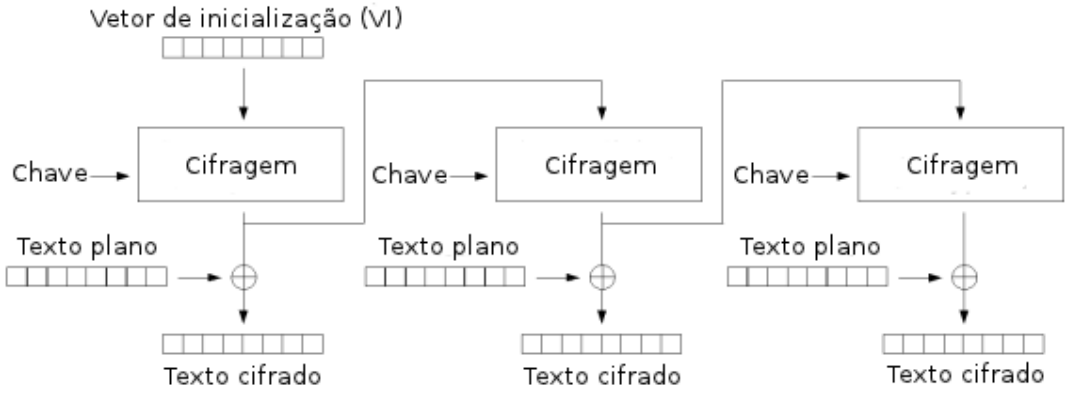


Figura 2.17: Cifragem no modo de operação OFB.

### 2.7.5 *Counter Mode* (CTR)

O modo de operação Contador (*Counter Mode*) aplica uma sequência de blocos de entrada na cifragem, chamados de contadores, para produzir blocos cifrados que então são utilizados para realizar a operação de XOR com o texto plano, ou com o texto cifrado, no caso da decifragem. A única condição imposta por este modo é que cada bloco na sequência, para uma chave igual, seja único. Por não utilizar a mensagem em si para realizar a cifragem, o modo Contador pertence ao conjunto dos modos de fluxo de chave, não necessitando de *padding* ao final da mensagem cifrada.

A figura 2.18 abaixo demonstra a utilização do modo CTR com um *nonce* concatenado ao contador.

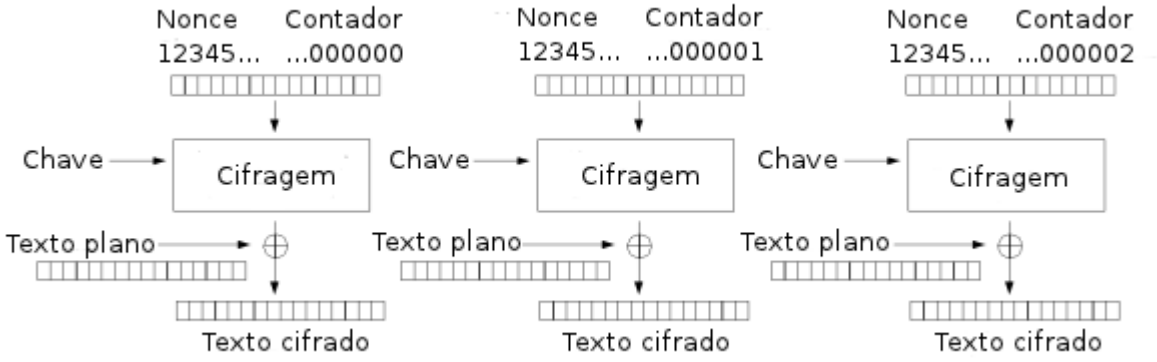


Figura 2.18: Cifragem no modo de operação CTR.

# 3 INTRODUÇÃO À CRIPTOANÁLISE E ATAQUES AO AES

Criptoanálise é o estudo de métodos que exploram possíveis falhas em sistemas criptográficos, que possam comprometer, isto é, "quebrar" o método de cifragem, expondo dados secretos, como a mensagem original, antes de ser cifrada, ou a chave utilizada para cifrá-la.

Pode-se dividir o campo de abrangência da criptoanálise entre análises matemáticas (linear, diferencial, integral, entre outras) e ataques laterais (*Side-channel attacks*). Na primeira categoria, encontram-se as criptoanálises mais utilizadas e de maior importância, no que tange à verificação de possíveis fraquezas em um algoritmo de cifragem. Quanto aos ataques laterais, estes se baseiam em obter informações da implementação (seja lógica ou física) de um sistema de criptografia como, por exemplo, medindo o tempo de partes do processo de cifragem/decifragem, ou até mesmo inserindo falhas durante a computação da cifragem, e observando seu comportamento na saída do processo.

Além do método de criptoanálise ou ataque ao sistema de criptografia, deve-se escolher o modelo de dados que será utilizado para a análise. Isto significa definir o conjunto de informações que um suposto atacante terá para que, quando aliado a algum método, possa ameaçar a confiabilidade de um sistema criptográfico. Dentre os principais, pode-se citar: somente texto cifrado (*ciphertext-only*), texto plano e texto cifrado conhecido (*known-plaintext*), texto plano e/ou texto cifrado escolhido (*chosen plaintext/ciphertext*), e ainda chaves relacionadas (*related key*).

Nas seções que seguem, alguns dos métodos de criptoanálises mais utilizados no AES são apresentados.

## 3.1 Ataque por força bruta

A maneira mais simples para determinar qual chave um algoritmo de cifragem utilizou para transformar um texto plano em um texto cifrado é testar exaustivamente todas combinações possíveis para a chave. Dado que um atacante tenha conhecimento do texto de entrada e do texto de saída de uma cifragem, ele pode testar qual dentre todas possíveis chaves gerou, dada uma entrada, a saída esperada.

Com o avanço do poder computacional, durante as últimas décadas, vários algoritmos criptográficos ficaram suscetíveis a este ataque, como o é o caso do DES, que utiliza uma chave de 56 *bits.*

Visando o rápido desenvolvimento tecnológico, novos sistemas criptográficos já preveem ataques de força bruta, e por este motivo exigem tamanhos de chave que tornam o ataque impraticável, como o AES. Por exemplo, para a chave de 128 *bits*,

deve-se realizar o processo de cifragem, e uma comparação entre a saída e o texto cifrado original, $2^{128}$ vezes, uma complexidade em tempo de execução que duraria mais que o tempo de existência do universo, dado o poder computacional atual.

Porém, apesar de não realizar todas combinações de *bits* possíveis para uma chave, a tentativa por força bruta é ainda utilizada, normalmente apenas para uma parte da chave, complementando o ataque com algum outro método criptoanalítico.

## 3.2 Criptoanálise linear

Desenvolvida como uma nova técnica de ataque à cifra FEAL e, pouco tempo após, ao DES [MATSUI, M., 1994], a criptoanálise linear consiste em encontrar aproximações lineares ao processo de cifragem de um algoritmo, de acordo com a expressão abaixo:

$$P[i_i,i_2,..,i_a] \text{ XOR } C[j_i,j_2,..,j_b] = K[k_i,k_2,..,k_c]$$

onde $i_a$, $j_b$ e $k_c$ denotam posições fixas dos *bits* do texto plano, texto cifrado e da chave, respectivamente. Utilizando textos planos randômicos, e seus respectivos textos cifrados, a magnitude da probabilidade *p* da expressão acima de satisfazer os pares de textos, onde *p* deve ser diferente de ½, define a efetividade da expressão linear. Maiores detalhes sobre a criptoanálise linear podem ser encontrados em [MATSUI, M., 1994] e [HEYS, H. M., 2002].

Para prevenir possíveis ataques através da criptoanálise linear, os autores do algoritmo Rijndael desenvolveram-no utilizando a estratégia *Wide Trail* [DAEMEN, J., 2002-b]. Esta estratégia visa limitar correlações de *bits* entre as rodadas, diminuindo assim expressões lineares com grande probabilidade de satisfazer a cifragem.

Por este motivo, e dado também que, para um ataque a quantidade de textos planos necessários acaba por não ser prática, até o momento não foram desenvolvidas criptoanálises lineares visando o AES.

## 3.3 Criptoanálise Diferencial

Descoberta publicamente desde o fim da década de 80 pelos criptoanalistas Eli Biham e Adi Shamir, a criptoanálise diferencial é um método de criptoanálise que pode ser aplicada tanto a cifras de bloco, de fluxo e funções de *hash*. De forma geral, este método estuda como diferenças constantes entre pares de textos planos e pares de textos cifrados podem ser utilizadas para detectar padrões estatísticos que determinem a chave utilizada. No início, a aplicação do método se restringia a utilizar pares de textos escolhidos (*chosen plaintext*) pelo criptoanalista, onde a diferença é usualmente gerada através de uma simples operação de XOR entre os *bits* dos textos.

Por se tratar de uma criptoanálise já amplamente utilizada, assim como a linear, a estratégia de *Wide Trail*, introduzida na seção 3.2, prevê um limite superior no que tange à probabilidade de rastros diferenciais para apenas quatro rodadas na cifra Rijndael [DAEMEN, J., 2002-c]. Em combinação com o número total de rodadas, é possível provar que existe uma boa margem de segurança contra ataques através da criptoanálise diferencial clássica.

Após a publicação do AES, métodos diferenciais mais genéricos foram aplicados a este. A seguir, são apresentados alguns dos métodos que obtiveram, até o momento, maior avanço na criptoanálise do AES.

### 3.3.1 Diferencial Impossível

Enquanto a criptoanálise diferencial comum procura por diferenças que propagam com grande probabilidade através do processo de cifragem, a criptoanálise diferencial impossível explora diferenças que são impossíveis de acontecer, isto é, com probabilidade zero, em algum estado da cifragem.

Por exemplo, em [CHEON, J. H., 2002], a operação de mistura de colunas do AES é explorada com o intuito de gerar uma condição diferencial impossível. Segundo seus autores, o ataque é capaz de quebrar cinco rodadas do AES de 128 *bits*, utilizando $2^{29.5}$ textos escolhidos e uma complexidade de tempo de $2^{31}$ operações de cifragem.

Utilizando chaves relacionadas (*related key*) para um ataque ao AES de 192 *bits* reduzido à 8 rodadas, Biham et al. demonstram três tipos de criptoanálises diferenciais impossíveis, obtendo uma complexidade de dados de entrada do ataque variando entre $2^{68.5}$ a $2^{116}$, proporcionalmente à complexidade temporal, que varia entre $2^{184}$ a $2^{134}$ [BIHAM, E., 2006].

### 3.3.2 Ataque *Boomerang*

Com o intuito de estender o número de rodadas que um rastro probabilístico da criptoanálise diferencial consegue quebrar, o método de ataque *Boomerang* divide a cifragem em subpartes, normalmente duas, e destas obtém seus sub-rastros diferenciais.

Basicamente, o método se baseia em realizar cifragens parciais de pares de textos planos escolhidos, com uma diferença conhecida, e decifragens de seus respectivos textos cifrados adicionando uma diferença provável àquela introduzida nos textos planos.

No que se refere ao AES, em [BIRYUKOV, A., 2009] é apresentado um ataque para as versões completas, isto é, com todas as devidas rodadas, do AES de 192 e 256 *bits*, utilizando a técnica de ataque Boomerang. Ambos os ataques utilizam relações entre diferentes chaves (*related key*) na cifragem de textos planos escolhidos (*chosen plaintext*). Para as 14 rodadas do AES 256, o método diz-se capaz de realizar o ataque com uma complexidade de dados de $2^{119}$ textos, e utilizando $2^{119}$ cifragens.

## 3.4 Criptoanálise Integral

Originalmente desenvolvida por Lars Knudsen como um ataque para a cifra SQUARE [DAEMEN, J., 1997], a criptoanálise integral consiste em utilizar conjuntos de textos planos escolhidos, ao invés de pares, como a linear ou diferencial. Destes textos planos, escolhe-se uma parte de *bytes* que sempre mantém valores constantes, e outra parte que varia com todas possiblidades de valores.

Para o AES, por exemplo, com um tamanho de bloco de 16 *bytes*, variando apenas o primeiro *byte* e mantendo todos outros com um valor constante, tem-se um conjunto de 256 textos planos escolhidos para se realizar a criptoanálise integral.

Os autores do Rijndael, em sua proposta para o AES [DAEMEN, J., 1999], descrevem o método de criptoanálise integral, na época nomeado de ataque SQUARE devido à sua origem. Demonstram também que, para até seis rodadas do algoritmo, o ataque é mais rápido que um ataque de força bruta para achar a chave, obtendo uma complexidade de execução de cifragens de $2^{72}$, utilizando $2^{32}$ textos planos escolhidos.

## 3.5 Ataques Algébricos

Na criptoanálise, ataques algébricos consistem em construir equações matemáticas para representar o processo de cifragem de um algoritmo, buscando assim um sistema de equações baseado, normalmente, nos *bits* ou *bytes* do texto plano, cifrado, da chave desconhecida, e também dos estados entre as transformações de rodada.

Em [FERGUSON, N., 2001], uma fórmula fechada é apresentada para a representação do algoritmo Rijndael, onde os *bytes* de cinco rodadas podem ser expressados na fórmula. Apesar de sua simplicidade, em comparação à representação algébrica de outras cifras, segundo os autores do Rijndael [DAEMEN, J., 2002-c], um ataque algébrico deste porte teria duas equações, cada uma com $2^{25}$ termos desconhecidos, não existindo atualmente um algoritmo prático de resolução de um sistema deste tamanho.

No ano de 2002, Courtois e Pieprzyck [COURTOIS, N., 2002] introduziram um ataque chamado XSL (*Extended Sparse Linearization*), onde, de forma geral, o Rijndael é derivado em um sistema de equações quadráticas contendo por exemplo 8000 equações e 1600 variáveis para o AES de 128 *bits*. Um algoritmo especializado, o XSL, é aplicado então para resolver tal sistema de equações.

Apesar destas descobertas, até o momento nenhum ataque prático utilizando as simples expressões algébricas do AES foi demonstrado.

## 3.6 Ataques *Meet In The Middle*

O ataque Encontro no Meio (*Meet in The Middle*) é uma técnica de ataque genérica, isto é, pode ser aplicada a qualquer sistema criptográfico, a qual tenta encontrar valores para possíveis chaves que encriptem um texto plano até uma parte da cifragem, e possíveis chaves para a decifragem também parcial do texto cifrado, literalmente se encontrando no meio do processo. Uma tabela guarda as possíveis chaves, e, quando os dois textos forem equivalentes no meio do processo, tem-se um candidato para a chave.

Apesar de sua complexidade de dados ser baixa em comparação a outras criptoanálises, requerendo usualmente um ou poucos pares de textos, a não linearidade entre as subchaves do AES se mostra como um obstáculo para realizar o ataque.

Visando a aplicação deste ataque, juntamente com outro método chamado de biclique, [BOGDANOV, A., 2011] propõem o primeiro ataque para as três versões completas do AES, isto é, para os três tamanhos de chave, com todas rodadas necessárias para cada chave. A principal diferença deste ataque para os anteriores, como citam os autores, é o não requerimento de chaves relacionadas para a realização do ataque. Mesmo assim, a complexidade, apesar de menor que por força bruta, não é prática. Tem-se uma complexidade de $2^{126}$ para o AES-128, uma redução por um fator de 4 em comparação à força bruta.

# 4 CRIPTOANÁLISE DIFERENCIAL DO AES REDUZIDO

Como introduzido no capítulo anterior, sobre criptoanálise diferencial, esta consiste em adquirir informações do estado, de cifragem ou decifragem, de textos analisando diferenças entre pares conhecidos de dados.

Normalmente, quando um ataque deste tipo é realizado sobre uma cifragem completa, isto é, apenas obtendo diferenças na entrada e na saída de textos, probabilidades devem ser atribuídas ao processo, visando estabelecer seu potencial para "quebrar" a cifra. No AES, como apresentado anteriormente, seus autores, e outros criptoanalistas, já comprovaram sua resistência contra ataques diferenciais, analisando as probabilidades diferenciais geradas pelas transformações entre cada rodada [KELIHER, L., 2005].

Apesar disso, ainda é possível, para um número reduzido de rodadas, explorar propriedades diferenciais. No capítulo que segue, são apresentadas notações que serão utilizadas no decorrer do texto, assim como noções preliminares que darão embasamento para que, quando utilizadas concomitantemente, resultarão em dois ataques ao algoritmo AES, sendo um deles ao AES reduzido a uma rodada e outro ao AES reduzido a duas rodadas.

Tais ataques foram escolhidos pois, dentre a gama de ataques conhecidos publicamente contra o AES, estes podem ser realizados com baixa complexidade de tempo, de dados (pares de textos planos/cifrados), e utilização mínima de memória. Em suma, são ataques de cunho prático, e facilmente demonstráveis, visando o ensino da aplicação da criptoanálise diferencial.

## 4.1 Notações

No decorrer deste capítulo, algumas notações serão utilizadas para facilitar a visualização e entendimento do processo de criptoanálise. As notações a seguir são consistentes com as encontradas em [BOUILLAGUET, C., 2010].

*Números:* quando necessário, números serão representados no formato hexadecimal, indicados por um zero ('0') e um 'x' no início do valor. Por exemplo, o número de valor 99 em base decimal será denotado pelo valor hexadecimal 0x63.

*Texto plano:* também podendo ser denotado como texto claro, é composto por 128 *bits*, representados por um bloco de 16 *bytes*, e consiste no texto de entrada do processo de cifragem, isto é, o texto original da mensagem a ser cifrada, e será representado pela letra *P*.

*Texto cifrado:* de mesmo tamanho do texto plano, consiste na saída do processo de cifragem. Será denotado pela letra *C*.

*Transformações:* cada transformação aplicada sobre a matriz estado poderá ser denota pela sua sigla, e sua operação inversa (com exceção da adição de subchaves, onde a inversa é a própria transformação) será indicada pelo número '-1' sobrescrito ou pelo prefixo "Inv". Por exemplo, a operação de mistura de colunas pode ser referenciada como MC (inicias da expressão em inglês, *MixColumns*), e a operação inversa por $MC^{-1}$ ou *InvMixColumns*. As operações de deslocamento de linhas e substituição de *bytes* serão referenciadas por SR (*ShiftRows*) e SB (*SubBytes)*, e suas inversas, respectivamente, por $SR^{-1}$ ou *InvShiftRows* e $SB^{-1}$ ou *InvSubBytes*, enquanto a operação de adição de subchaves somente pela sigla ARK (*AddRoundKey*).

*Subchaves de rodada:* para cada rodada *i*, com 0 <= i <= Nr, denotar-se-á cada subchave pela letra *k* com o número da rodada em subscrito. Tem-se, então, a subchave $k_0$ para a rodada de início (*whitening round)*, $k_1$ para a primeira rodada, e assim por diante. No caso especial de uma subchave ser alterada a partir de seu valor original através de uma transformação, como ocorre na seção 4.1.3 a seguir, denotar-se-á a subchave modificada por *u* e o respectivo subscrito da subchave original *k*, e.g., $u_i = MC^{-1}(k_i)$.

*Valores da matriz estado:* de modo a representar os valores intermediários que a matriz estado assume entre uma e outra transformação, define-se quatro variáveis com subscritos da rodada em questão: $x_i$, $y_i$, $w_i$ e $z_i$. A matriz estado (no caso, seus valores) são representados, no início da rodada *i*, por $x_i$, enquanto $y_i$, $w_i$ e $z_i$ representam a matriz estado após a aplicação das transformações de substituição de *bytes* (SB), descolamento de colunas (SR) e mistura de colunas (MC), respectivamente.

*Indexação de bytes*: quando se fizer necessário indicar um *byte* em específico em um dos conjuntos definidos acima, como por exemplo nos valores intermediários da matriz estado, indica-se a posição ordinal do *byte* no bloco dentre os 16 *bytes*, calculado através de sua linha e coluna, como mostrado pela figura 2.1. O valor do índice do *byte* será a segunda parte do subscrito. Por exemplo, o *byte* $x_{i,l+4*c}$ denota o *byte* na posição *l+4*c*, onde *l* indica a linha e *c* a coluna, da matriz estado no início da rodada *i*. Quando necessário denotar uma coluna inteira, isto é, um vetor de quatro *bytes*, ao invés da posição dos *bytes* será utilizada a expressão *Col(c)*, onde *c* é o índice da coluna desejada, para 0 <= c < 4.

## 4.2 Noções preliminares

Para a realização dos ataques diferenciais às versões reduzidas do AES, algumas propriedades devem ser exploradas com o intuito de obter informações entre os estados das transformações, seja dos textos planos, cifrados, ou entre as diferenças dos pares de ambos. A seguir, três características do algoritmo AES são apresentadas, e serão utilizadas para compor os ataques.

### 4.2.1 Análise diferencial da *S-box*

Provendo a característica diferencial dos ataques escolhidos, a análise diferencial da caixa de substituição de *bytes* do AES consiste em obter valores *x* e *y* que geram diferenças conhecidas antes e após a aplicação da transformação de substituição de *bytes*.

Especificamente, dado um par de diferenças (α, β), onde α deve ser diferente de zero, α é a diferença na entrada e β é a diferença na saída de uma caixa de substituição, tenta-se encontrar dois valores *x* e *y* que satisfaçam as expressões abaixo:

$$x \text{ XOR } y = \alpha \quad (4.1)$$

$$\text{Sbox}(x) \text{ XOR } \text{Sbox}(y) = \beta \quad (4.2)$$

Em outros termos, dadas as diferenças de entrada e saída, a caixa de substituição de *bytes* do AES assume um comportamento linear.

Exemplificando, para:

$$\alpha = 0\text{xaa e } \beta = 0\text{x21}$$

tem-se um par de valores possível igual a (0xd8, 0x72). Pode-se verificar este resultado aplicando-o em 4.1 e 4.2:

$$0\text{xd8 XOR } 0\text{x72} = 0\text{xaa} = \alpha$$

$$\text{Sbox}(0\text{xd8}) = 0\text{x61}$$

$$\text{Sbox}(0\text{x72}) = 0\text{x40}$$

$$\text{Sbox}(0\text{xd8}) \text{ XOR } \text{Sbox}(0\text{x72}) = 0\text{x61 XOR } 0\text{x40} = 0\text{x21} = \beta$$

Visto que a operação de XOR comuta ($x$ XOR $y$ = $y$ XOR $x$), o par de valores possíveis encontrado não define uma ordem para os resultados, isto é, não é possível precisar qual valor pertence a $x$ e a $y$.

Para obter os valores de $x$ e $y$, é utilizada uma tabela pré-computada, chamada de tabela de distribuição diferencial da *S-box* (DDT, do inglês *differential distribution table*). A tabela é gerada a partir de todas as $2^{16}$ possibilidades para os valores de $x$ e $y$, isto é, iterando estas variáveis de 0x00 a 0xff, aplicando-as nas expressões 4.1 e 4.2, e guardando seus valores em uma posição da tabela indexada pelo valor das diferenças $\alpha$ e $\beta$.

De acordo com [BOUILLAGUET, C., 2010] e com [MENDEL, F., 2009], analisando a tabela DDT da *S-box*, verifica-se que para 129/256, isto é, aproximadamente pouco mais da metade, os pares de diferenças de entrada/saída da *S-box* não têm um mapeamento para um par ($x$, $y$), ou seja, não existe valor que satisfaz as expressões 4.1 e 4.2. Para 126/256 (aproximadamente pouco mais de 49%) dos pares de diferenças existem dois pares ordenados ($x$, $y$) que satisfazem as relações, e para 1/256 (menos de 1%) existem quatro pares ordenados ($x$, $y$) que satisfazem.

Deste modo, a utilização desta análise diferencial torna o ataque probabilístico, dependendo do número de *S-box* exploradas e podendo exigir um maior número de textos planos/cifrados conhecidos do que o mínimo necessário. Apesar disso, por se tratar de diferenças entre os *bytes* dos textos, quando utilizado um número maior que dois pares de textos, pode-se trocar a ordem destes, buscando resultados diferenciais que possam ser calculados pela tabela DDT.

### 4.2.2 Relação entre subchaves

Como explicado na seção 2.4, a chave recebida na entrada do processo de cifragem/decifragem do AES é expandida e dividida entre subchaves, de acordo com seu tamanho, de modo que cada subchave é utilizada somente uma vez, em uma rodada específica, durante todo processo.

Para a versão de 128 *bits* do AES, por exemplo, somente os *bytes* da primeira subchave (utilizada na rodada inicial, denotada por $k_0$) contém os *bytes* da chave

original, recebida na entrada, e normalmente sua obtenção é o único objetivo da maioria dos ataques criptográficos.

Porém é fácil verificar que, sabendo como o processo de expansão de chave é realizado, é possível reconstruir as subchaves, revertendo a aplicação das operações de expansão. Por exemplo, a partir dos 16 *bytes* da subchave $k_1$ pode-se calcular a subchave $k_0$, a partir de $k_2$ pode-se achar $k_1$, e assim por diante.

Um simples exemplo é demonstrado abaixo na figura 4.1, onde se deseja saber qual o valor do primeiro *byte* de $k_0$, ou, de acordo com a notação definida anteriormente, o *byte* $k_{0,0}$, através dos *bytes* conhecidos de $k_1$.

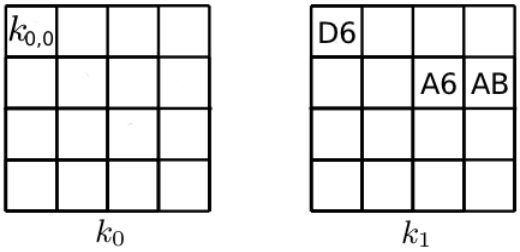


Figura 4.1: Exemplo de relação entre subchaves.

Sabe-se que o *byte* $k_{0,0}$ é utilizado para o cálculo do *byte* $k_{1,0}$, de acordo com a expressão abaixo:

$$k_{1,0} = \text{RotWord}(\text{SubWord}(k_{0,\text{Col}(3)}))_{,0} \text{ XOR } \text{Rcon}[1]_{,0} \text{ XOR } k_{0,0} \quad (4.3)$$

onde $\text{Rcon}[1]_{,0}$ é o *byte* na posição zero do vetor Rcon (definido em 2.4.3) para subchave $k_1$, e tem valor 0x01. *SubWord* consiste na aplicação de cada *byte* de um vetor, com quatro *bytes* na caixa de substituição do AES. *RotWord* é a função de rotação de palavra, definida na seção 2.4.1 da expansão de chave, e, na expressão acima, recebe como parâmetro o resultado da aplicação da função *SubWord* na terceira coluna da subchave $k_0$. O *byte* 0 do resultado desta composição de funções é então selecionado. Isso significa aplicar o valor do segundo *byte* da última coluna de $k_0$ na caixa de substituição de *bytes*, e então a expressão em 4.3 pode ser simplificada por:

$$k_{1,0} = \text{Sbox}[k_{0,13}] \text{ XOR } 0x01 \text{ XOR } k_{0,0} \quad (4.4)$$

Pelo fato do XOR ser sua própria operação inversa, pode-se passar os termos para o outro lado da equação, isolando a variável $k_{0,0}$. O valor de $k_{1,0}$ é 0xD6, de acordo com a figura 4.1. Tem-se então a seguinte expressão:

$$k_{0,0} = 0xD6 \text{ XOR } \text{Sbox}[k_{0,13}] \text{ XOR } 0x01 \quad (4.5)$$

O valor de $k_{0,13}$ é desconhecido, não sendo possível ainda determinar $k_{0,0}$. Porém, com as informações disponíveis na figura 4.1, pode-se calcular também o valor daquele *byte*. Sabe-se, de acordo com a expansão de chave, que $k_{0,13}$ é também utilizado no cálculo do *byte* $k_{1,13}$ através da expressão:

$$k_{1,13} = k_{0,13} \text{ XOR } k_{1,9} \quad (4.6)$$

Vale salientar que a expressão 4.6 é diferente de 4.3 devido à coluna na qual o *byte* a ser calculado está. Para as colunas da chave expandida com o número de índice múltiplo do número de colunas da chave original (no caso do AES 128, aquela coluna sempre é a primeira coluna de cada subchave, devido ao tamanho de chave ser igual ao tamanho de

bloco na rodada) utiliza-se as operações tal como em 4.3. Já para as colunas restantes, apenas um XOR entre *bytes* de colunas anteriores é realizado, como visto em 4.6.

Isolando $k_{0,13}$ em 4.6 e substituindo os valores dos *bytes* $k_{1,9}$ e $k_{1,13}$ obtém-se:

$$k_{0,13} = k_{1,13} \text{ XOR } k_{1,9}$$

$$k_{0,13} = 0xAB \text{ XOR } 0xA6$$

$$k_{0,13} = 0x0D$$

Após obter o valor de $k_{0,13}$ é possível descobrir o valor de $k_{0,0}$ na expressão em 4.5:

$$k_{0,0} = 0xD6 \text{ XOR Sbox}[0x0D] \text{ XOR } 0x01$$

Uma consulta à caixa de substituição do AES, mostrada na figura 2.5, informa que o resultado, para o valor de entrada 0x0D, é 0xD7.

$$k_{0,0} = 0xD6 \text{ XOR } 0xD7 \text{ XOR } 0x01$$

$$k_{0,0} = 0x01\text{XOR } 0x01$$

$$k_{0,0} = 0x00$$

O valor do *byte* $k_{0,0}$, de acordo com os valores na figura 4.1 de três dos *bytes* de $k_1$, é 0x00.

Apesar do simples exemplo, a relação entre *bytes* de subchaves no AES é um assunto ainda bastante explorado por criptoanalistas, visto que a maior parte do algoritmo de expansão de chave se comporta de forma linear. Outras relações, mais complexas, podem ser utilizadas para realizar ataques ao AES, como, por exemplo, demonstradas em [BOUILLAGUET, C., 2010], porém fogem ao escopo deste trabalho.

### 4.2.3 Equivalência entre transformações de estado

Introduzido na seção 2.6 e exemplificado na figura 2.13, o AES dispõe de um modo de decifragem equivalente ao modo de cifragem, isto é, é possível implementar a decifragem sem alterar a ordem em que as transformações são aplicadas na cifragem. Esta propriedade foi projetada pelos autores do AES, e visa obter uma implementação mais eficiente para o algoritmo.

Em [DAEMEN, J., 2002-b], as propriedades algébricas, utilizadas para realizar a troca de ordem das transformações, são apresentadas em maiores detalhes. Resumidamente, esta consiste em duas inversões: deslocamento de linhas inverso e substituição de *bytes* inversa; e mistura de colunas inversa e adição de chave de rodada.

A transformação de deslocamento de linhas inversa opera alterando a posição dos *bytes* na matriz estado, enquanto a substituição de *bytes* inversa opera somente nos seus valores, não importando sua ordem na matriz. Deste modo, a ordem das duas operações é indiferente.

Devido ao fato de ser uma transformação linear, a mistura de colunas inversa pode ser aplicada à subchave a ser utilizada pela transformação de adição de chave de rodada. Assim, pode-se adicionar esta subchave à matriz estado após a operação de mistura de colunas inversa, sem que o resultado seja alterado. A nova subchave gerada pode ser descrita da seguinte forma, para uma rodada *i*:

$$u_i = MC^{-1}(k_i) \tag{4.7}$$

Por ser uma subchave modificada, $u_i$ não mantém nenhuma relação direta com os *bytes* das outras subchaves, como $k_{i-1}$ e $k_{i+1}$, não sendo possível aplicar as propriedades descritas na seção anterior.

Apesar disso, a expressão 4.7 denota uma transformação linear, através de uma matriz, como apresentada na seção 2.3.4, de modo que cada coluna *c* de $u_i$ recebe o resultado da multiplicação da matriz da mistura de colunas inversa pela coluna *c* de $k_i$.

Generalizando, tem-se um par de vetores (*a, b*) de quatro *bytes*, tal que:

$$a = MC\ (b) \tag{4.8}$$

isto é, a entrada e saída da operação de mistura de colunas (esta e sua inversa são lineares, portanto preservam a mesma propriedade) aplicadas a uma coluna. Expandindo a expressão 4.8, obtém-se um sistema linear contendo oito variáveis e quatro equações, cada uma relacionando uma variável do vetor *a* com as quatro variáveis do vetor *b*. Deste modo, quando o valor de quatro variáveis é conhecido, é possível determinar o valor das outras quatro variáveis resolvendo o sistema linear gerado por 4.8.

## 4.3 Ataque ao AES de uma rodada

O AES reduzido a uma rodada consiste na sequência de operações de adição de chave da rodada inicial, substituição de *bytes*, deslocamento de linhas, mistura de colunas e adição de chave da primeira rodada. A versão do AES a ser utilizada é a de tamanho de chave de 128 *bits*, porém o ataque pode ser estendido para as versões de 192 e 256 *bits*, dado que toda a chave de entrada é utilizada durante a primeira rodada. Como mostrado na figura 4.2, o processo de cifragem pode ser visualizado através de cada transformação aplicada sobre a matriz estado, representada por um bloco de 16 *byes.*

Os dados necessários para realizar o ataque consistem em, no mínimo, dois pares de textos planos conhecidos, e seus respectivos textos cifrados. O objetivo do ataque é recuperar a chave secreta utilizada para cifrar os pares de textos planos, ou seja, recuperar os 16 *bytes* de $k_0$, já que toda a chave de entrada é copiada para $k_0$, enquanto $k_1$ é resultado da expansão de chave.

Na figura 4.2 abaixo, o texto plano é denotado pela letra *P* (inicial do termo em inglês *plaintext*), e o texto cifrado pela letra *C* (inicial do termo em inglês *ciphertext*), e ambos os textos são representados pelos quadrados na cor cinza, denotando *bytes* conhecidos pelo atacante, enquanto os *bytes* em branco são desconhecidos.

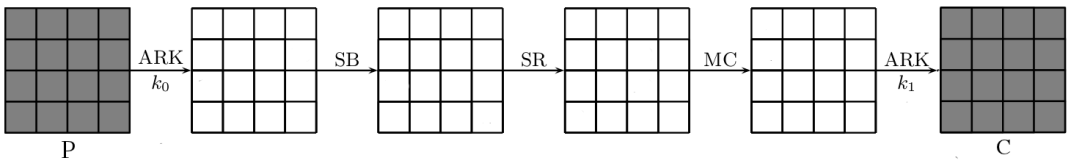


Figura 4.2: Cifragem do AES reduzido a uma rodada.

Para exemplificar o processo, serão utilizados dois pares de textos conhecidos, $P_1$ e $P_2$, e, respectivamente, seus textos cifrados, $C_1$ e $C_2$. Os valores dos *bytes* são representados no formato hexadecimal, porém a notação definida anteriormente (“0x”) será omitida para facilitar a visualização. Nas expressões que seguem, a posição de cada *byte* segue a ordem definida na figura 2.1, ou seja, o primeiro *byte* corresponde ao *byte*

da linha 0 e coluna 0, o segundo *byte* corresponde ao *byte* da linha 1 coluna 0, e assim sucessivamente.

$$P_1 = 41\ 51\ 6d\ 4a\ 6f\ 32\ 53\ 53\ 73\ 64\ 49\ 6f\ 4a\ 34\ 31\ 63$$

$$C_1 = 34\ 78\ a9\ 43\ 26\ 0c\ 96\ 44\ 26\ 60\ 60\ 68\ 28\ 0b\ 59\ 39$$

$$P_2 = 31\ 4f\ 31\ 41\ 6c\ 31\ 6e\ 55\ 34\ 30\ 4e\ 38\ 50\ 6a\ 33\ 73$$

$$C_2 = 60\ 60\ ad\ 8c\ 65\ 8b\ 1f\ 58\ 40\ 02\ 8b\ 91\ b5\ b3\ c7\ d7$$

O primeiro passo é obter as diferenças entre os pares de textos planos e entre os pares de textos cifrados, ou seja, as diferenças na entrada da cifragem e na saída. Uma operação deve ser escolhida para gerar a diferença, e, para o AES, utiliza-se o XOR. Realiza-se então o XOR *byte* a *byte* dos pares de texto, e denota-se a diferença por $\Delta P = P_1$ XOR $P_2$ e $\Delta C = C_1$ XOR $C_2$.

$$\Delta P = P_1 \text{ XOR } P_2 = 70\ 1e\ 5c\ 0b\ 03\ 03\ 3d\ 06\ 47\ 54\ 07\ 57\ 1a\ 5e\ 02\ 10$$

$$\Delta C = C_1 \text{ XOR } C_2 = 54\ 18\ 04\ cf\ 43\ 87\ 89\ 1c\ 66\ 62\ eb\ f9\ 9d\ b8\ 9e\ ee$$

Ao obter as diferenças através da operação de XOR, no AES, elimina-se assim o valor dos *bytes* das subchaves, neste caso $k_0$ e $k_1$, isto porque a transformação de adição de chave de rodada também utiliza a operação de XOR.

Em outras palavras, ambos os textos cifrados $C_1$ e $C_2$ contém o valor da subchave $k_1$ em seus *bytes*, visto que a adição de chave de rodada foi a última realizada antes da saída do texto cifrado. Ao operar ambos textos com XOR, o resultado $\Delta C$ contém a diferença tanto dos textos cifrados, quanto dos textos antes da adição da subchave $k_1$. O mesmo se aplica para os textos planos, porém com relação à subchave $k_0$, que é aplicada na primeira transformação realizada sobre estes textos. Tem-se, então, em $\Delta P$ a diferença dos textos planos, assim como a diferença entre os textos após a adição da subchave $k_0$.

Até o momento, utilizando os diferenciais dos textos, conseguiu-se eliminar o efeito de duas transformações da operação de cifragem, ARK da rodada inicial para os textos planos e ARK da primeira rodada para os textos cifrados. Para aquele par, a próxima operação é a de substituição de *bytes*. Por ser não-linear, não é possível inferir, por enquanto, o resultado desta transformação na aplicação das diferenças dos textos planos. Quanto ao par de textos cifrados, a transformação anterior é a de mistura de colunas (MC), e, por ser linear, pode-se aplicar sua inversa em $\Delta C$ para obter as diferenças antes desta transformação ocorrer:

$$MC^{-1}(\Delta C) = 44\ 82\ 01\ 40\ 3a\ 2e\ 52\ 17\ ca\ 97\ 6b\ 20\ 10\ 7c\ b9\ 80 \qquad (4.9)$$

Da mesma forma, o efeito da operação de deslocamento de linhas (SR) pode ser revertido do resultado de 4.9, aplicando a operação inversa:

$$SR^{-1}(MC^{-1}(\Delta C)) = 44\ 7c\ 6b\ 17\ 3a\ 82\ b9\ 20\ ca\ 2e\ 01\ 80\ 10\ 97\ 52\ 40 \qquad (4.10)$$

Tem-se em 4.10 a diferença dos *bytes* de $\Delta C$ logo após a operação de substituição de *bytes* (SB). Aliada a $\Delta P$, que consiste nas diferenças logo antes da aplicação de SB, pode-se utilizar da análise diferencial da *S-box,* descrita em 4.2.1, para inferir o valor dos *bytes* antes e após SB através das diferenças de entrada e saída da caixa de substituição.

Para o *byte* na primeira posição, por exemplo, tem-se o valor 0x70, de $\Delta P$, como a diferença na entrada, e 0x44, em 4.10, como a diferença na saída da *S-box*. Substituindo em 4.1 e 4.2, espera-se encontrar *x* e *y* tal que:

$$x \text{ XOR } y = \alpha = 0x70$$

$$\text{Sbox}(x) \text{ XOR } \text{Sbox}(y) = \beta = 0x44$$

Consultando a tabela DDT da *S-box*, para as diferenças $\alpha$ e $\beta$ acima, o par (0x31, 0x41) é encontrado como solução para as expressões acima. Como não é possível definir a ordem dos valores neste par resultante, cada valor do par (*x, y*) sugere um valor possível para o *byte* de mesma posição na subchave $k_0$.

Exemplificando, sabe-se que:

$$0x31 \text{ XOR } 0x41 = 0x70$$

porém não é possível afirmar qual valor é proveniente de qual texto plano, $P_1$ ou $P_2$. Para tanto, deve-se propagar os valores encontrados, seja o par (*x*, *y*) ou o par (*Sbox(x)*, *Sbox(y)*), em direção aos *bytes* conhecidos, isto é, ou o texto plano ou o texto cifrado. Dado que a distância até a operação de ARK mais próxima, em quantidade de transformações a partir da substituição de *bytes*, é menor do lado do texto plano, estabelece-se uma relação entre os *bytes* do texto plano e a entrada da caixa de substituição, com o intuito de encontrar os *bytes* de $k_0$, que pode ser definida como abaixo:

$$(x, y) = ARK_{k0}(P) \tag{4.11}$$

Escolhe-se então um dos textos planos para realizar as análises. A escolha em si é indiferente, apenas deve ser a mesma para toda análise feita sobre a *S-box*. No exemplo, os *bytes* de $P_1$ serão utilizados. Assim, a expressão 4.11 pode ser escrita desta forma:

$$(x, y) = k_{0,0} \text{ XOR } P_{1,0}$$

Passando o termo desconhecido $k_{0,0}$ para o lado esquerdo da equação, e expandindo-a em duas equações, visto que tanto *x* quanto *y* podem satisfazer, tem-se:

$$k_{0,0} = x \text{ XOR } P_{1,0} \text{ ou } k_{0,0} = y \text{ XOR } P_{1,0}$$

Substituindo $P_{1,0}$, *x* e *y*:

$$k_{0,0} = 0x31 \text{ XOR } 0x41 \text{ ou } k_{0,0} = 0x41 \text{ XOR } 0x41$$

$$k_{0,0} = 0x70 \text{ ou } k_{0,0} = 0x00 \tag{4.12}$$

De acordo com 4.12, o *byte* $k_{0,0}$ pode ter valor 0x70 ou 0x00.

Deve-se, então, repetir este processo de análise para o restante das diferenças de entrada e saída da *S-box*. A análise resultará, em média, em dois valores possíveis para cada *byte* de $k_0$. Combinando todas possibilidades, tem-se $2^{16}$ possíveis chaves, que podem ser testadas exaustivamente, encriptando um dos textos planos e verificando se a saída da cifragem equivale ao respectivo texto cifrado.

A tabela abaixo resume o processo de análise do exemplo acima para os quatro primeiros *bytes* de $k_0$.

| | Byte 00 | Byte 01 | Byte 02 | Byte 03 |
|---|---|---|---|---|
| Par de diferenças ($\alpha$, $\beta$) | (0x70, 0x44) | (0x1e, 0x7c) | (0x5c, 0x6b) | (0x0b, 0x17) |
| Par da tabela DDT (x, y) | (0x31, 0x41) | (0x4e, 0x50) | (0x33, 0x6f) | (0x42, 0x49) |
| *Byte* do texto plano $P_{1,i}$ | 0x41 | 0x51 | 0x6d | 0x4a |
| Possíveis valores para $k_{0,1}$ | 0x70 ou 0x00 | 0x1f ou 0x01 | 0x5e ou 0x02 | 0x08 ou 0x03 |

Tabela 4.1: Análise dos quatro primeiros *bytes* de $k_0$.

Para este exemplo, a chave correta encontrada é:

K = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

## 4.4 Ataque ao AES de duas rodadas

Utilizando também a versão de 128 *bits* do AES, este ataque consiste em encontrar a chave utilizada na cifragem de textos planos/cifrados conhecidos através do AES reduzido a duas rodadas completas, isto é, a adição de chave da rodada inicial e duas rodadas compostas das operações de substituição de *bytes*, deslocamento de linhas, mistura de colunas e adição de chave.

Em [BOUILLAGUET, C., 2010], duas versões são propostas para este ataque: uma utilizando dois e outra utilizando três pares de textos conhecidos. Apesar de muito similares quanto ao método de criptoanálise, o conhecimento de três pares de texto acaba por diminuir substancialmente a complexidade de tempo do ataque, e por isso esta versão será utilizada neste trabalho.

Em resumo, o ataque consiste em duas fases que visam explorar as diferenças na entrada e saída de uma caixa de substituição durante o processo de cifragem. Para que as diferenças sejam conhecidas, será necessário supor seis *bytes* da chave $k_0$, de modo a propagar e acompanhar as transformações das diferenças geradas pelos textos planos. Além disso, relações entre as três subchaves, $k_0$, $k_1$ e $k_2$, serão exploradas, e a equivalência na ordem das operações de mistura de colunas e adição de chave de rodada será útil para obter *bytes* adicionais na última subchave.

Para exemplificar o processo, serão utilizados os pares de textos planos e seus respectivos textos cifrados abaixo:

$P_1$ = 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

$C_1$ = 49 15 59 8f 55 e5 d7 a0 da ca 94 fa 1f 0a 63 f7

$P_2$ = ff ee dd cc bb aa 99 88 77 66 55 44 33 22 11 00

$C_2$ = fb 6e fd a8 2c d1 05 60 60 fd 96 80 f1 18 49 cb

$P_3$ = 77 88 99 aa bb cc dd ee ff 00 11 22 33 44 55 66

$C_3$ = a6 26 8d d1 f6 3b 35 1c 31 93 0b 95 dd 05 9b 8d

### 4.4.1 Primeira fase

A primeira fase do ataque pode ser visualmente expressada na figura a seguir, e será explicada em detalhes no texto que segue.

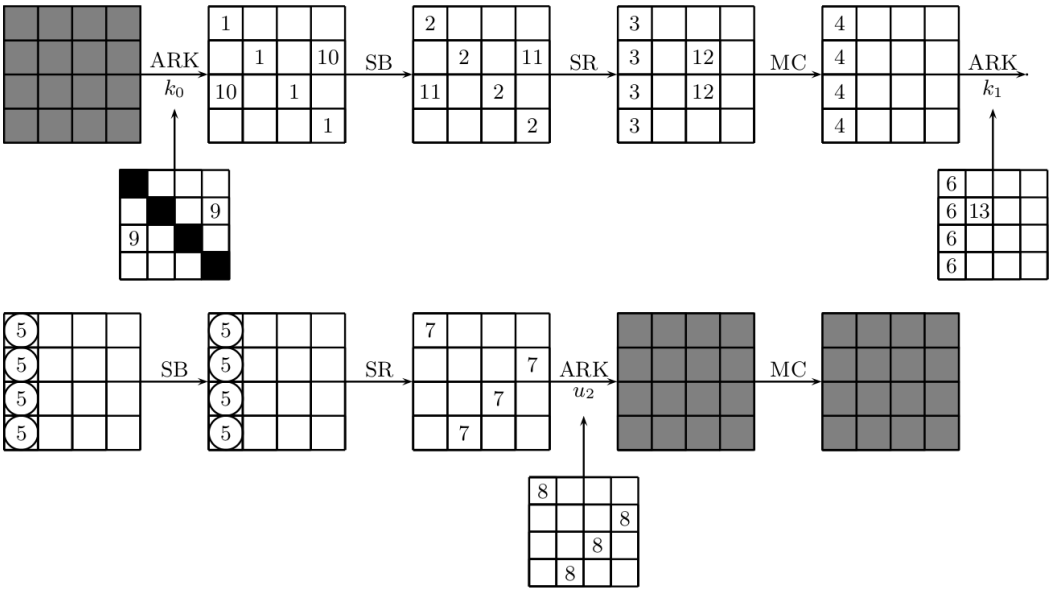


Figura 4.3: Primeira fase do ataque ao AES reduzido a duas rodadas. (BOUILLAGUET, C., 2010).

Na figura acima, o texto plano, cifrado e a matriz estado em cada transformação são representados por blocos de 16 *bytes* contendo 4 linhas e 4 colunas. Os *bytes* em cinza denotam *bytes* conhecidos pelo atacante, enquanto os *bytes* na cor branca são desconhecidos. Os *bytes* em preto serão supostos pelo atacante.

O primeiro passo da primeira fase do ataque consiste em supor quatro *bytes* de $k_0$. Deste modo, a complexidade de tempo para execução do ataque já fica limitada inferiormente por $2^{32}$ iterações, visto que para os quatro *bytes* supostos existem $2^{32}$ possibilidades de combinações. O intuito deste passo é propagar quatro dos *bytes* dos textos planos para o início da segunda rodada, isto é, logo antes da transformação de substituição de *bytes* daquela rodada.

Como mostra a figura 4.3, demarcados em cor preta, os *bytes* a serem supostos são os localizados nas posições 0, 5, 10 e 15 de $k_0$, ou seja, em termos matriciais, a diagonal principal. Visando o entendimento e continuidade do ataque, os *bytes* aqui supostos para $k_0$ estarão corretos de acordo com a chave secreta. Porém, para um ataque sem o conhecimento da mesma, é necessário testar todas combinações possíveis dos *bytes* supostos. Tem-se então em $k_0$ os seguintes valores:

$k_0$ = 00 ?? ?? ?? ?? 05 ?? ?? ?? ?? 0a ?? ?? ?? ?? 0f

onde “??” denota valores desconhecidos pelo atacante até o momento.

Com estes quatro *bytes* de $k_0$ escolhidos, pode-se aplicar a adição dos mesmos sobre os três textos planos, no que seria a transformação de adição de chave da rodada inicial, porém somente com estes quatro *bytes*.

$$ARK_{k0}(P_1) = 00\ ??\ ??\ ??\ ??\ 50\ ??\ ??\ ??\ ??\ a0\ ??\ ??\ ??\ ??\ f0$$

$$ARK_{k0}(P_2) = ff\ ??\ ??\ ??\ ??\ af\ ??\ ??\ ??\ ??\ af\ ??\ ??\ ??\ ??\ 0f$$

$$ARK_{k0}(P_3) = 77\ ??\ ??\ ??\ ??\ c9\ ??\ ??\ ??\ ??\ 1b\ ??\ ??\ ??\ ??\ 69$$

Sobre os *bytes* resultantes, aplica-se a transformação de substituição de *bytes* e deslocamento de linhas, respectivamente:

$$SB(ARK_{k0}(P_1)) = 63\ ??\ ??\ ??\ ??\ 53\ ??\ ??\ ??\ ??\ e0\ ??\ ??\ ??\ ??\ 8c$$

$$SB(ARK_{k0}(P_2)) = 16\ ??\ ??\ ??\ ??\ 79\ ??\ ??\ ??\ ??\ cf\ ??\ ??\ ??\ ??\ 76$$

$$SB(ARK_{k0}(P_3)) = f5\ ??\ ??\ ??\ ??\ dd\ ??\ ??\ ??\ ??\ af\ ??\ ??\ ??\ ??\ f9$$

$$SR(SB(ARK_{k0}(P_1))) = 63\ 53\ e0\ 8c\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??$$

$$SR(SB(ARK_{k0}(P_2))) = 16\ 79\ cf\ 76\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??$$

$$SR(SB(ARK_{k0}(P_3))) = f5\ dd\ af\ f9\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??$$

A próxima transformação é a mistura de colunas. Aqui fica claro o porquê da escolha dos *bytes* de $k_0$ pertencerem à diagonal principal: devido ao deslocamento de linhas, tem-se a primeira coluna de *bytes* com valores conhecidos, sendo possível aplicar a operação de mistura de colunas sobre somente esta coluna, sem que os *bytes* de valor desconhecido pelo atacante alterem os *bytes* conhecidos.

$$z1_{1,Col(0)} = MC(SR(SB(ARK_{k0}(P_1)))_{Col(0)}) = 5f\ 72\ 64\ 15$$

$$z2_{1,Col(0)} = MC(SR(SB(ARK_{k0}(P_2)))_{Col(0)}) = 1e\ d8\ 70\ 60$$

$$z3_{1,Col(0)} = MC(SR(SB(ARK_{k0}(P_3)))_{Col(0)}) = db\ 47\ 7d\ 9f$$

Os quatro passos exemplificados acima, assim como os *bytes* encontrados, são denotados na figura 4.3 pelos números 1, 2, 3 e 4.

Em $z1_{1,Col(0)}$, $z2_{1,Col(0)}$ e $z3_{1,Col(0)}$ tem-se os *bytes* da primeira coluna da matriz estado logo antes da operação de adição de chave da primeira rodada. De maneira semelhante à realizada no ataque ao AES reduzido a uma rodada, obtém-se a diferença entre os *bytes* conhecidos, através da operação de XOR. Esta diferença será utilizada para explorar a caixa de substituição de *bytes* da segunda rodada.

$$\Delta z12_{1,Col(0)} = z1_{1,Col(0)}\ XOR\ z2_{1,Col(0)} = 41\ aa\ 14\ 75$$

$$\Delta z13_{1,Col(0)} = z1_{1,Col(0)}\ XOR\ z3_{1,Col(0)} = 84\ 35\ 19\ 8a$$

Acima, em $\Delta z12_{1,Col(0)}$ e $\Delta z13_{1,Col(0)}$, tem-se então as diferenças para a entrada da segunda *S-box* do processo de cifragem.

O próximo passo consiste em encontrar as diferenças na saída desta mesma caixa. Diferentemente do ataque a uma rodada, é optado por primeiro aplicar a operação inversa de mistura de colunas, ao invés de obter as diferenças entre os textos cifrados. Deste modo, previne-se que esta operação, aliada ao deslocamento de linhas, impeça a propagação dos *bytes* conhecidos pelo atacante na primeira rodada até o fim da segunda rodada.

Como demonstrado na subseção 4.2.3, é possível considerar, no processo de cifragem, a aplicação de chave de rodada antes da mistura de colunas, desde que o valor da subchave a ser utilizada seja modificado de acordo com esta operação. Deste modo, enquanto na rodada inicial e na primeira rodada tem-se as subchaves $k_0$ e $k_1$ encontradas através da expansão de chave, a adição de chave da segunda rodada deverá utilizar uma chave modificada, denotada por $u_2$, de modo a manter a equivalência das operações.

Abaixo, é realizada então a transformação inversa de mistura de colunas dos textos cifrados:

$$MC^{-1}(C_1) = 07\ 65\ 18\ f0\ b5\ 2b\ a2\ fb\ 7a\ 15\ c8\ d9\ 3b\ e4\ 5e\ 00 \quad (4.13)$$

$$MC^{-1}(C_2) = 48\ fc\ d5\ a1\ 5d\ a4\ 5f\ 3e\ 0b\ 7b\ f0\ 0b\ f0\ c5\ 9e\ c0 \quad (4.14)$$

$$MC^{-1}(C_3) = 77\ fc\ ff\ a8\ d3\ 3e\ c3\ ca\ 41\ 67\ 03\ 19\ 91\ d0\ 50\ df \quad (4.15)$$

e suas diferenças são então calculadas através do XOR, eliminando assim os valores da operação ARK da segunda rodada:

$\Delta C_{1;2} = MC^{-1}(C_1)$ XOR $MC^{-1}(C_2) = 4f\ 99\ cd\ 51\ e8\ 8f\ fd\ c5\ 71\ 6e\ 38\ d2\ cb\ 21\ c0\ c0$

$\Delta C_{1;3} = MC^{-1}(C_1)$ XOR $MC^{-1}(C_3) = 70\ 99\ e7\ 58\ 66\ 15\ 61\ 31\ 3b\ 72\ cb\ c0\ aa\ 34\ 0e\ 0f$

Realizando o deslocamento de linhas inverso sobre as diferenças $\Delta C_{1;2}$ e $\Delta C_{1;3}$:

$$\Delta y12_2 = SR^{-1}(\Delta C_{1;2}) = 4f\ 21\ 38\ c5\ e8\ 99\ c0\ d2\ 71\ 8f\ cd\ c0\ cb\ 6e\ fd\ 51 \quad (4.16)$$

$$\Delta y13_2 = SR^{-1}(\Delta C_{1;3}) = 70\ 34\ cb\ 31\ 66\ 99\ 0e\ c0\ 3b\ 15\ e7\ df\ aa\ 72\ 61\ 58 \quad (4.17)$$

$$\Delta y12_{2,Col(0)} = SR^{-1}(\Delta C_{1;2})_{Col(0)} = 4f\ 21\ 38\ c5$$

$$\Delta y13_{2,Col(0)} = SR^{-1}(\Delta C_{1;3})_{Col(0)} = 70\ 34\ cb\ 31$$

tem-se as diferenças dos 16 *bytes* na saída da *S-box* da segunda rodada. Juntamente com $\Delta z12_{1,Col(0)}$ e $\Delta z13_{1,Col(0)}$, pode-se consultar a tabela DDT da *S-box* visando encontrar o par $(x, y)$ que satisfaz as diferenças de entrada e saída da *S-box* da segunda rodada para os *bytes* da primeira coluna. A tabela abaixo mostra os resultados encontrados para os *bytes* de $\Delta z12_{1,Col(0)}$ e $\Delta y12_{2,Col(0)}$:

| | Byte 00 | Byte 01 | Byte 02 | Byte 03 |
|---|---|---|---|---|
| Par de diferenças (α, β) | (0x41, 0x4f) | (0xaa, 0x21) | (0x14, 0x38) | (0x75, 0xc5) |
| Par da tabela DDT (x, y) | (0xc8, 0x89) | (0xd8, 0x72) | (0x10, 0x04) | (0xe8, 0x9d) |

Tabela 4.2: Análise da primeira coluna da *S-box* da segunda rodada dos pares 1 e 2

Obtém-se assim dois possíveis valores na entrada da caixa de substituição da segunda rodada, para cada *byte* da primeira coluna. No caso do ataque somente utilizar dois pares de textos conhecidos, o próximo passo seria testar todas combinações para estes quatro *bytes*, em relação à diferença de entrada $\Delta x_{1;2,Col(0)}$, para encontrar os *bytes* equivalentes de $k_1$, tal como realizado no ataque a uma rodada. Porém, tendo disponível três pares de textos conhecidos, pode-se utilizar do outro conjunto de diferenças de entrada e saída para esta *S-box*, $\Delta z13_{1,Col(0)}$ e $\Delta y13_{2,Col(0)}$.

| | Byte 00 | Byte 01 | Byte 02 | Byte 03 |
|---|---|---|---|---|
| Par de diferenças (α, β) | (0x84, 0x70) | (0x35, 0x34) | (0x19, 0xcb) | (0x8a, 0x31) |
| Par da tabela DDT (x, y) | (0x89, 0x0d) | (0xed, 0xd8) | (0x10, 0x09) | (0xe8, 0x62) |

Tabela 4.3: Análise da primeira coluna da *S-box* da segunda rodada dos pares 1 e 3

Os pares (*x*, *y*) encontrados nas tabelas 4.2 e 4.3 sugerem, cada um, um valor possível para *x* e um para *y*. Porém, dado que ambas diferenças foram geradas através de textos planos e cifrados utilizando a mesma chave secreta em comum, mais especificamente a mesma $k_1$, ambos pares (*x*, *y*) devem ter um valor em comum. A intersecção destes pares é mostrada na tabela abaixo.

| | Byte 00 | Byte 01 | Byte 02 | Byte 03 |
|---|---|---|---|---|
| (x, y) da tabela 4.2 | (0xc8, 0x89) | (0xd8, 0x72) | (0x10, 0x04) | (0xe8, 0x9d) |
| (x, y) da tabela 4.3 | (0x89, 0x0d) | (0xed, 0xd8) | (0x10, 0x09) | (0xe8, 0x62) |
| Valor em comum | 0x89 | 0xd8 | 0x10 | 0xe8 |

Tabela 4.4: Intersecção dos valores para a primeira coluna da *S-box*

Tem-se então, na linha "Valor em comum" da tabela acima, o valor de entrada na caixa de substituição da segunda rodada, denotado aqui por $x1_{2,Col(0)}$, quando o texto plano utilizado na cifragem é $P_1$ (devido às diferenças serem relacionadas a este, em relação a $P_2$ e $P_3$). Assim, sabe-se que:

$$x1_{2,Col(0)} = ARK_{k1,Col(0)}(z1_{1,Col(0)})$$

isto é, $x1_{2,Col(0)}$ é a primeira coluna da matriz estado no início da segunda rodada. $x1_{2,Col(0)}$ é calculada operando com XOR a primeira coluna de $z1_{1,Col(0)}$, isto é, a saída da operação de mistura de colunas da primeira rodada, com a primeira coluna de $k_1$. Sabendo os valores de $x1_{2,Col(0)}$ e $z1_{1,Col(0)}$, pode-se então calcular os quatro primeiros *bytes* de $k_1$.

$$x1_{2,Col(0)} = k_{1,Col(0)} \text{ XOR } z1_{1,Col(0)}$$

$$k_{1,Col(0)} = x1_{2,Col(0)} \text{ XOR } z1_{1,Col(0)} \quad (4.18)$$

Expandindo a expressão 4.18 acima em quatro, uma para cada *byte*, tem-se:

$$k_{1,0} = x1_{2,0} \text{ XOR } z1_{1,0}$$

$$k_{1,1} = x1_{2,1} \text{ XOR } z1_{1,1}$$

$$k_{1,2} = x1_{2,2} \text{ XOR } z1_{1,2}$$

$$k_{1,3} = x1_{2,3} \text{ XOR } z1_{1,3}$$

Substituindo com os valores conhecidos, encontra-se os *bytes* de $k_1$:

$$k_{1,0} = 0x89 \text{ XOR } 0x5f = 0xd6$$

$$k_{1,1} = 0xd8 \text{ XOR } 0x72 = 0xaa$$

$$k_{1,2} = 0x10 \text{ XOR } 0x64 = 0x74$$

$$k_{1,3} = 0xe8 \text{ XOR } 0x15 = 0xfd$$

$$k_{1,Col(0)} = \text{ d6 aa 74 fd ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??}$$

Na figura 4.3, para este passo, as diferenças exploradas na *S-box* estão denotadas pelo número 5, enquanto os *bytes* encontrados em $k_1$ são marcados com o número 6.

De posse destes novos *bytes* de $k_1$, torna-se possível propagar os *bytes* conhecidos do texto plano $P_1$ até o momento, $x1_{2,Col(0)}$, até a adição de chave da segunda rodada, isto é, com a subchave $u_2$, ainda desconhecida. Para isto, aplica-se sobre a coluna as operações de substituição de *bytes*:

$$y1_{2,Col(0)} = SB(x1_{2,Col(0)}) = a7\ 61\ ca\ 9b$$

e, em seguida, o deslocamento de linhas. Por estarem em uma coluna, os quatro *bytes* conhecidos acima serão distribuídos para as quatro colunas da matriz estado, sobre os *bytes* de posição 0, 7. 10 e 13:

$$w1_2 = SR(y1_{2,Col(0)}) = a7\ ??\ ??\ ??\ ??\ ??\ ??\ 9b\ ??\ ??\ ca\ ??\ ??\ 61\ ??\ ??$$

Os *bytes* de $w1_2$ contém os valores da matriz estado antes da adição de chave da segunda rodada. Após esta transformação, sabe-se que estes *bytes* tem valores conhecidos, definidos em 4.13, pela inversão da mistura de colunas sobre o texto cifrado $C_1$.

$$MC^{-1}(C_1) = ARK_{u2}(w1_2)$$

Assim, pode-se encontrar os *bytes* 0, 7, 10 e 13 da subchave $u_2$ utilizada para o ARK da segunda rodada de acordo com a expressão abaixo:

$$MC^{-1}(C_1) = u_2 \text{ XOR } w1_2$$

$$u_2 = MC^{-1}(C_1) \text{ XOR } w1_2$$

$$u_2 = a0\ ??\ ??\ ??\ ??\ ??\ ??\ 60\ ??\ ??\ 02\ ??\ ??\ 85\ ??\ ??$$

Visualmente, na figura 4.3, os *bytes* encontrados em $u_2$ são representados pelo número 8.

Neste ponto do ataque, já se tem conhecimento de quatro *bytes* de cada uma das chaves de rodada, $k_0$, $k_1$ e $u_2$. Deve-se lembrar que todos *bytes* são suposições, dado que os quatro *bytes* de $k_0$ foram escolhidos pelo atacante, e devem iterar sobre todas combinações possíveis.

Em comum, tem-se nas três subchaves o *byte* de posição 0 conhecido. Assim, pode-se propagar o *byte* 0 dos textos planos $P_2$ e $P_3$ por todo o processo de cifragem, e verificar na saída se seu valor está de acordo com o *byte* 0 do respectivo texto cifrado. Caso a suposição dos quatro *bytes* de $k_0$ feita no início do ataque estiver incorreta, os *bytes* propagados terão valor diferente do esperado.

Abaixo, as expressões mostram a propagação, a cada transformação, do último valor conhecido para os textos $P_2$ e $P_3$, logo antes da adição de chave da primeira rodada, até após a adição de chave da segunda rodada.

$$x2_{2,0} = ARK_{k1,0}(z2_{1,0}) = k_{1,0} \text{ XOR } z2_{1,0} = 0xd6 \text{ XOR } 0x1e = 0xc8$$

$$x3_{2,0} = ARK_{k1,0}(z3_{1,0}) = k_{1,0} \text{ XOR } z3_{1,0} = 0xd6 \text{ XOR } 0xdb = 0x0d$$

$$y2_{2,0} = SB(x2_{2,0}) = SB(0xc8) = 0xe8$$

$$y3_{2,0} = SB(x3_{2,0}) = SB(0x0d) = 0xd7$$

Por pertencer à primeira linha, o *byte* na posição 0 não é deslocado na operação de deslocamento de linhas.

$$w2_{2,0} = y2_{2,0} = 0xe8$$

$$w3_{2,0} = y3_{2,0} = 0xd7$$

Após o ARK da segunda rodada, o *byte* 0 é comparado com o respectivo *byte* do texto cifrado aplicado à mistura de colunas inversa. Se ambas expressões forem verdadeiras, então pode-se continuar o ataque. Caso contrário, outra escolha para $k_0$ deve ser feita.

$$ARK_{u2,0}(w2_{2,0}) = MC^{-1}(C_2)_0$$

$$ARK_{u2,0}(w3_{2,0}) = MC^{-1}(C_3)_0$$

Substituindo com os valores conhecidos:

$$w2_{2,0} \text{ XOR } u_{2,0} = 0xe8 \text{ XOR } 0xa0 = 0x48$$

$$w3_{2,0} \text{ XOR } u_{2,0} = 0xd7 \text{ XOR } 0xa0 = 0x77$$

e, sabendo que o *byte* 0 de $MC^{-1}(C_2)$ tem valor 0x48 de acordo com 4.14, e o *byte* 0 de $MC^{-1}(C_3)$ tem valor 0x77 de acordo com 4.15, a suposição da chave $k_0$ se mostra correta até este passo do ataque.

O último passo da primeira fase do ataque consiste em obter três *bytes* adicionais nas subchaves $k_0$ e $k_1$ através das relações entre estas, de modo semelhante ao exemplificado na subseção 4.2.2. Estes *byte*s podem ser identificados na figura 4.3 denotados pelos números 9 e 13.

O *byte* da posição 2 de $k_0$ é calculado a partir da seguinte expressão:

$$k_{1,2} = \text{SubWord}(\text{RotWord}(k_{0,Col(3)}))_2 \text{ XOR } \text{Rcon}[1]_2 \text{ XOR } k_{0,2}$$

onde o termo $SubWord(RotWord(k_{0,Col(3)}))_2$ é simplesmente a seleção do *byte* na posição 15 de $k_0$ aplicado na caixa de substituição de *bytes*. Isolando $k_{0,2}$ e substituindo, tem-se:

$$k_{0,2} = \text{SubWord}(\text{RotWord}(k_{0,Col(3)}))_2 \text{ XOR } k_{1,2}$$

$$k_{0,2} = 0x76 \text{ XOR } 0x74$$

$$k_{0,2} = 0x02$$

O *byte* da posição 13 de $k_0$ é calculado a partir da seguinte expressão:

$$k_{1,0} = \text{SubWord}(\text{RotWord}(k_{0,Col(3)}))_0 \text{ XOR } \text{Rcon}[1]_0 \text{ XOR } k_{0,0}$$

onde o termo $SubWord(RotWord(k_{0,Col(3)}))_0$ contém o valor procurado. Em suma, $RotWord(k_{0,Col(3)})_0$ seleciona o primeiro *byte* da terceira coluna de $k_0$ rotacionada, que representa o *byte* na posição 13. Isolando e passando a operação de substituição de *bytes* para o lado direito da equação, tem-se:

$$k_{1,0} = \text{SubBytes}(k_{0,13}) \text{ XOR } \text{Rcon}[1]_0 \text{ XOR } k_{0,0}$$

$$\text{SubBytes}(k_{0,13}) = k_{1,0} \text{ XOR } \text{Rcon}[1]_0 \text{ XOR } k_{0,0}$$

$$k_{0,13} = \text{InvSubBytes}(k_{1,0} \text{ XOR } \text{Rcon}[1]_0 \text{ XOR } k_{0,0})$$

onde *InvSubBytes* é o mesmo que a operação $SB^{-1}$, isto é, substituição de *bytes* inversa.

Substituindo os valores conhecidos:

$$k_{0,13} = \text{InvSubBytes}(0xd6 \text{ XOR } 0x01 \text{ XOR } 0x00)$$

$$k_{0,13} = \text{InvSubBytes}(0xd7)$$

$$k_{0,13} = 0x0d$$

O terceiro e último *byte* a ser recuperado nesta fase do ataque é o *byte* da posição 5 de $k_1$, e pode ser expressado diretamente pela expansão de chave deste mesmo *byte*:

$$k_{1,5} = k_{0,5} \text{ XOR } k_{1,1}$$

$$k_{1,5} = 0x05 \text{ XOR } 0xaa$$

$$k_{1,5} = 0xaf$$

Assim, ao fim da primeira fase do ataque, tem-se conhecimento de 6 *bytes* de $k_0$, 5 *bytes* de $k_1$ e 4 *bytes* de $u_2$.

### 4.4.2 Segunda fase

Para a segunda fase, dois *bytes* de $k_0$ são supostos pelo atacante, necessitando que este teste todas combinações possíveis para os novos dois *bytes*. De modo semelhante à primeira fase, utiliza-se os novos *bytes* supostos, juntamente com os dois *bytes* de $k_0$ calculados através da relação entre subchaves na fase anterior, para propagar novos valores dos três textos planos até o fim da primeira rodada. Então é calculada a diferença entre estes *bytes* para, novamente, explorar a *S-box* da segunda rodada, obtendo novos *bytes* de $k_1$ e $u_2$.

A figura 4.4 abaixo ilustra a segunda fase do ataque:

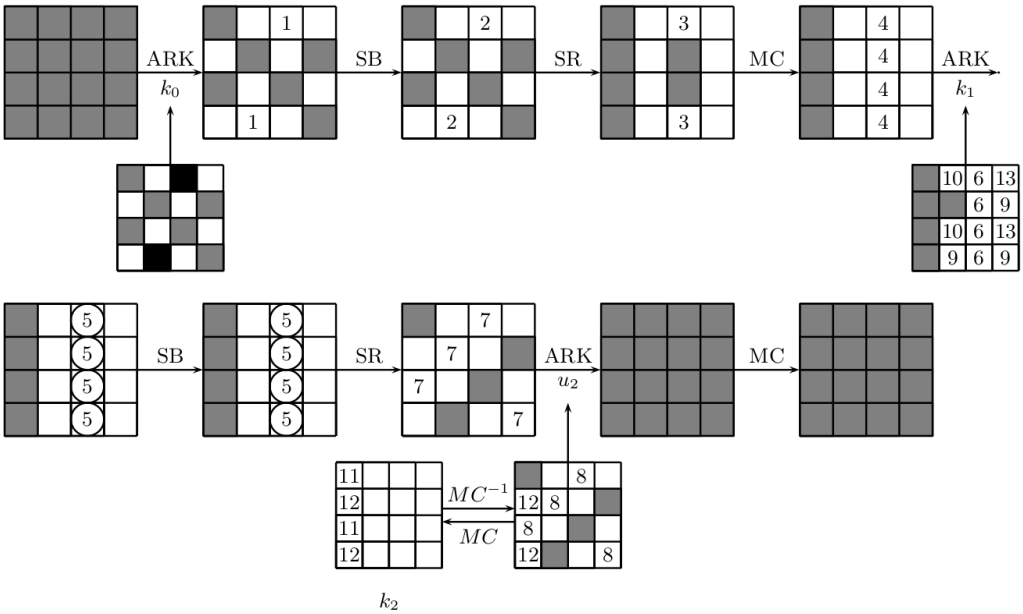


Figura 4.4: Segunda fase do ataque ao AES reduzido a duas rodadas. (BOUILLAGUET, C., 2010)

Os dois *bytes* de $k_0$ a serem supostos pelo atacante ocupam as posições 7 e 8 de $k_0$, e, para este exemplo, como observado anteriormente, a suposição estará correta de acordo com a chave secreta. Os valores para $k_{0,7}$ e $k_{0,8}$ serão, respectivamente, 0x07 e 0x08.

Os *bytes* conhecidos e supostos de $k_0$ são mostrados a seguir:

$$k_0 = 00\ ??\ 02\ ??\ ??\ 05\ ??\ 07\ 08\ ??\ 0a\ ??\ ??\ 0d\ ??\ 0f$$

A seguir, os passos 1, 2, 3 e 4 denotados na figura 4.3 são calculados, respectivamente, para os três textos planos com a nova chave $k_0$:

$$ARK_{k0}(P_1) = 00\ ??\ 20\ ??\ ??\ 50\ ??\ 70\ 80\ ??\ a0\ ??\ ??\ d0\ ??\ f0$$

$$ARK_{k0}(P_2) = ff\ ??\ df\ ??\ ??\ af\ ??\ 8f\ 7f\ ??\ af\ ??\ ??\ 2f\ ??\ 0f$$

$$ARK_{k0}(P_3) = 77\ ??\ 9b\ ??\ ??\ c9\ ??\ e9\ f7\ ??\ 1b\ ??\ ??\ 49\ ??\ 69$$

$$SB(ARK_{k0}(P_1)) = 63\ ??\ b7\ ??\ ??\ 53\ ??\ 51\ cd\ ??\ e0\ ??\ ??\ 70\ ??\ 8c$$

$$SB(ARK_{k0}(P_2)) = 16\ ??\ 9e\ ??\ ??\ 79\ ??\ 73\ d2\ ??\ cf\ ??\ ??\ 15\ ??\ 76$$

$$SB(ARK_{k0}(P_3)) = f5\ ??\ 14\ ??\ ??\ dd\ ??\ 1e\ 68\ ??\ af\ ??\ ??\ 3b\ ??\ f9$$

$$SR(SB(ARK_{k0}(P_1))) = 63\ 53\ e0\ 8c\ ??\ ??\ ??\ ??\ cd\ 70\ b7\ 51\ ??\ ??\ ??\ ??$$

$$SR(SB(ARK_{k0}(P_2))) = 16\ 79\ cf\ 76\ ??\ ??\ ??\ ??\ d2\ 15\ 9e\ 73\ ??\ ??\ ??\ ??$$

$$SR(SB(ARK_{k0}(P_3))) = f5\ dd\ af\ f9\ ??\ ??\ ??\ ??\ 68\ 3b\ 14\ 1e\ ??\ ??\ ??\ ??$$

$$z1_{1,Col(2)} = MC(SR(SB(ARK_{k0}(P_1)))_{Col(2)}) = f7\ be\ 3b\ 29$$

$$z2_{1,Col(2)} = MC(SR(SB(ARK_{k0}(P_2)))_{Col(2)}) = 6d\ 32\ 75\ 00$$

$$z3_{1,Col(2)} = MC(SR(SB(ARK_{k0}(P_3)))_{Col(2)}) = 97\ 3c\ 59\ ab$$

Com a terceira coluna da matriz estado contendo *bytes* conhecidos e propagados até antes da adição de chave da primeira rodada, calcula-se a diferença, para obter assim a diferença de entrada da *S-box* da segunda rodada.

$$\Delta z12_{1,Col(3)} = z1_{1,Col(3)}\ XOR\ z2_{1,Col(3)} = 9a\ 8c\ 4e\ 29$$

$$\Delta z13_{1,Col(3)} = z1_{1,Col(3)}\ XOR\ z3_{1,Col(3)} = 60\ 82\ 62\ 82$$

Com as diferenças dos textos cifrados logo após a *S-box* da segunda rodada, obtida em 4.16 e 4.17, pode-se utilizar novamente da análise da *S-box* através das diferenças nos *bytes* entrada e saída, porém desta vez sobre a terceira coluna. A tabela 4.5 a seguir resume a análise.

| | Byte 08 | Byte 09 | Byte 10 | Byte 11 |
|---|---|---|---|---|
| Par de diferenças (α, β) para $P_1$ e $P_2$ | (0x94, 0x71) | (0x8c, 0x8f) | (0x4e, 0xcd) | (0x29, 0xc0) |
| Par da tabela DDT (x, y) para $P_1$ e $P_2$ | (0xb7, 0x2d) | (0x94, 0x18) | (0x43, 0x0d) | (0xf1, 0xd8) |
| Par de diferenças (α, β) para $P_1$ e $P_3$ | (0x60, 0x3b) | (0x82, 0x15) | (0x62, 0xe7) | (0x82, 0xdf) |
| Par da tabela DDT (x, y) para $P_1$ e $P_3$ | (0x4d, 0x2d) | (0x9a, 0x18) | (0x43, 0x21) | (0xd8, 0x5a) |
| Valor em comum | 0x2d | 0x18 | 0x43 | 0xd8 |

Tabela 4.5: Análise completa da terceira coluna da *S-box* da segunda rodada.

Tal como na primeira fase, com os valores de entrada da *S-box*, pode-se encontrar os *bytes* respectivos para $k_1$, a partir da terceira coluna de $P_1$. A expressão é semelhante à 4.18, porém sobre a terceira coluna. O termo $x1_{2,Col(3)}$ representa os quatro *bytes* encontrados para $P_1$ na tabela 4.5, isto é, os *bytes* 8, 9, 10 e 11.

$$k_{1,Col(3)} = x1_{2,Col(3)} \text{ XOR } z1_{1,Col(3)}$$

Expandindo e substituindo, na expressão acima, com os valores conhecidos, encontra-se os *bytes* de $k_1$:

$$k_{1,8} = 0x2d \text{ XOR } 0xf7 = 0xda$$

$$k_{1,9} = 0x18 \text{ XOR } 0xbe = 0xa6$$

$$k_{1,10} = 0x43 \text{ XOR } 0x3b = 0x78$$

$$k_{1,11} = 0xd8 \text{ XOR } 0x29 = 0xf1$$

Até o momento, $k_1$ contém os seguintes *bytes* conhecidos:

$$k_1 = \text{ d6 aa 74 fd ?? af ?? ?? da a6 78 f1 ?? ?? ?? ??}$$

Utilizando os novos *bytes* de $k_1$, pode-se propagar os valores no início da segunda rodada referentes à terceira coluna de $P_1$ até a adição de chave da segunda rodada, para obter então novos valores da subchave $u_2$. A transformação de substituição de *bytes* é aplicada:

$$y1_{2,Col(3)} = SB(x1_{2,Col(3)}) = \text{d8 ad 1a 61}$$

e, após, o deslocamento de linhas. Aqui, os *bytes* previamente conhecidos na primeira fase (0, 7, 10 e 13) são mostrados juntamente com os novos *bytes* encontrados.

$$w1_2 = SR(y1_{2,Col(3)}) = \text{a7 ?? 1a ?? ?? ad ?? 9b d8 ?? ca ?? ?? 61 ?? 61}$$

Novamente, utilizando-se do texto cifrado operado com a mistura de colunas inversa, e os novos valores de $w1_2$, pode-se calcular os *bytes* 2, 5, 8 e 15 de $u_2$, obtendo nesta subchave os valores abaixo:

$$u_2 = a0\ ??\ 02\ ??\ ??\ 86\ ??\ 60\ a2\ ??\ 02\ ??\ ??\ 85\ ??\ 61$$

O próximo passo consiste em aplicar novas relações entre os *bytes* conhecidos das subchaves, os quais são denotados pelos números 9 e 10 em $k_1$ e 11 em $k_2$ na figura 4.4.

Para $k_1$, as expressões relacionam os novos *bytes* de $k_0$ e $k_1$. Os cinco novos *bytes* são calculados como segue:

$$k_{1,7} = k_{1,3}\ \text{XOR}\ k_{0,7} = 0xfd\ \text{XOR}\ 0x07 = 0xfa$$

$$k_{1,15} = k_{1,11}\ \text{XOR}\ k_{0,15} = 0xf1\ \text{XOR}\ 0x0f = 0xfe$$

$$k_{1,13} = k_{1,9}\ \text{XOR}\ k_{0,13} = 0xa6\ \text{XOR}\ 0x0d = 0xab$$

$$k_{1,4} = k_{1,8}\ \text{XOR}\ k_{0,8} = 0xda\ \text{XOR}\ 0x08 = 0xd2$$

$$k_{1,6} = k_{1,10}\ \text{XOR}\ k_{0,10} = 0x78\ \text{XOR}\ 0x0a = 0x72$$

Quanto à $k_2$, não se tem, até este passo do ataque, qualquer valor conhecido, devido à utilização da chave modificada para a segunda rodada, $u_2$. Porém, através de $k_0$ e $k_1$, é possível calcular dois *bytes* de $k_2$, nas posições 0 e 2, tal como é feito pela expansão de chave, como mostram as expressões abaixo.

$$k_{2,0} = \text{SubWord}(\text{RotWord}(k_{1,\text{Col}(3)}))_0\ \text{XOR}\ \text{Rcon}[2]_0\ \text{XOR}\ k_{1,0}$$

$$k_{2,2} = \text{SubWord}(\text{RotWord}(k_{1,\text{Col}(3)}))_0\ \text{XOR}\ \text{Rcon}[2]_2\ \text{XOR}\ k_{1,2}$$

Substituindo e calculando as aplicações das funções sobre os *bytes* indicados, tem-se:

$$k_{2,0} = 0x62\ \text{XOR}\ 0x02\ \text{XOR}\ 0xd6 = 0xb6$$

$$k_{2,2} = 0xbb\ \text{XOR}\ 0x00\ \text{XOR}\ 0x74 = 0xcf$$

A penúltima parte deste ataque explora a relação entre as subchaves $k_2$ e $u_2$. Como explicado na subseção 4.2.3, as operações de adição de rodada e mistura de colunas podem ser aplicadas em ordem inversa na cifragem, desde que a chave seja modificada de acordo.

A relação entre $k_2$ e $u_2$ é expressa por:

$$u_2 = MC^{-1}(k_2)$$

ou, invertendo a operação:

$$k_2 = MC\ (u_2)$$

Por ser uma multiplicação matricial, a mistura de colunas pode ser escrita em quatro expressões de um sistema linear de oito variáveis:

$$k_{2,0} = 0x02*u_{2,0}\ \text{XOR}\ 0x03*u_{2,1}\ \text{XOR}\ u_{2,2}\ \text{XOR}\ u_{2,3}$$

$$k_{2,1} = u_{2,0}\ \text{XOR}\ 0x02*u_{2,1}\ \text{XOR}\ 0x03*u_{2,2}\ \text{XOR}\ u_{2,3}$$

$$k_{2,2} = u_{2,0}\ \text{XOR}\ u_{2,1}\ \text{XOR}\ 0x02*u_{2,2}\ \text{XOR}\ 0x03*u_{2,3}$$

$$k_{2,3} = 0x03*u_{2,0}\ \text{XOR}\ 0x03*u_{2,1}\ \text{XOR}\ u_{2,2}\ \text{XOR}\ 0x02*u_{2,3}$$

Sabe-se o valor de quatro das oito variáveis deste sistema, sendo elas os *bytes* $k_{2,0}$, $k_{2,2}$, $u_{2,0}$ e $u_{2,2}$. Deste modo, resolvendo o sistema linear, pode-se encontrar o valor dos *bytes* restantes $k_{2,1}$, $k_{2,3}$, $u_{2,1}$ e $u_{2,3}$.

Para o exemplo, os *bytes* encontrados para $k_{2,1}$, $k_{2,3}$, $u_{2,1}$ e $u_{2,3}$ são, respectivamente, 0x92, 0x0b, 0xdb e 0x99.

Os dados conhecidos, até o momento, das três subchaves *k* são:

$$k_0 = 00\ ??\ 02\ ??\ ??\ 05\ ??\ 07\ 08\ ??\ 0a\ ??\ ??\ 0d\ ??\ 0f$$

$$k_1 = d6\ aa\ 74\ fd\ d2\ af\ 72\ fa\ da\ a6\ 78\ f1\ ??\ ab\ ??\ fe$$

$$k_2 = b6\ 92\ cf\ 0b\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??\ ??$$

Pode-se notar que restam apenas dois *bytes* de $k_1$ para que esta subchave seja completamente determinada. Uma vez que alguma subchave for determinada por completo, é possível reverter todo processo de expansão de chave, até obter a chave secreta utilizada para a cifragem.

Utilizando estes dois novos *bytes* de $k_2$, através da relação entre subchaves, é possível então achar $k_{1,12}$ e $k_{1,14}$. Revertendo as expressões que calculam $k_{2,1}$ e $k_{2,3}$, e isolando as variáveis das quais se deseja encontrar os valores, tem-se:

$$k_{1,12} = \text{InvSubBytes}(k_{2,3}\ \text{XOR}\ k_{1,3})$$

$$k_{1,14} = \text{InvSubBytes}(k_{2,1}\ \text{XOR}\ k_{1,1})$$

e substituindo com os valores conhecidos:

$$k_{1,12} = \text{InvSubBytes}(0x0b\ \text{XOR}\ 0xfb) = 0xd6$$

$$k_{1,14} = \text{InvSubBytes}(0x92\ \text{XOR}\ 0xaa) = 0x76$$

Com toda subchave $k_1$ determinada, o último passo do ataque consiste em encontrar o restante de *bytes* ainda não determinados em $k_0$, também através de relações entre subchaves. Para este exemplo, o valor de $k_0$ encontrado é:

$$k_0 = 00\ 01\ 02\ 03\ 04\ 05\ 06\ 07\ 08\ 09\ 0a\ 0b\ 0c\ 0d\ 0e\ 0f$$

Tem-se assim uma possível chave secreta encontrada. O atacante deve então cifrar um dos textos planos, por exemplo $P_1$, e verificar se a saída desta cifragem é igual ao respectivo texto cifrado, neste caso $C_1$. Em caso positivo, a chave foi encontrada. Em caso negativo, deve-se recomeçar o ataque, desde o primeiro passo, porém tentando uma combinação diferente para os *bytes* supostos pelo atacante.

Segundo [BOUILLAGUET, C., 2010], este ataque tem uma complexidade de tempo de execução de aproximadamente $2^{32}$ operações de cifragem.

# 5 SIMULADOR YAAES

Com o objetivo de demonstrar interativamente o processo de cifragem do AES e uma criptoanálise diferencial aplicada a este algoritmo de criptografia, dois simuladores, um para cada fim, foram implementados. A base de ambos foi denominada de *Yet Another AES* (YAAES). O simulador de cifragem leva o nome de YAAES *Inspector*, enquanto o simulador que implementa o ataque à versão do AES reduzida a duas rodadas é denominado de YAAES *Attacker*. No capítulo que segue, uma breve descrição da implementação é apresentada, seguido da apresentação de cada um dos simuladores.

## 5.1 Especificações e visão geral da implementação

Os simuladores, assim como a classe que implementa a cifragem e decifragem do AES, foram desenvolvidos em ambiente Linux, utilizando a linguagem C++. Para os simuladores, a interface foi desenvolvida utilizando o *framework* Qt4 [QT4], permitindo assim que os arquivos fontes sejam compilados em múltiplas plataformas, tal como Windows e OS X, além do Linux.

Para o núcleo dos simuladores é utilizada uma classe, também em C++, que implementa o algoritmo AES, nomeada de *FastRijndael*. O nome foi escolhido por ser o nome dado pelos seus autores, antes da padronização pelo NIST. A partir da segunda versão desta classe feita para este trabalho, as operações foram otimizadas, utilizando, por exemplo, cópias de trechos de memória ao invés de atribuição simples, e pré-calculando operações custosas, como multiplicações. Por este motivo, a classe leva a palavra *Fast* no nome.

Internamente, os dados são armazenados em vetores unidimensionais do tipo *unsigned char*. Este tipo foi escolhido por ter o tamanho exato necessário para armazenar *bytes*, isto é, 256 valores, de 0 a 255, visto que a linguagem C++ não possui um tipo específico para *bytes*.

Entre as principais funcionalidades da classe, estão a cifragem e decifragem de textos de 16 *bytes*, podendo-se utilizar chaves nos três tamanhos permitidos para o AES, 128, 192 e 256 *bits*. Por padrão, a cifragem e decifragem são realizadas no modo ECB, porém existe o suporte para os modos CBC, CFB e OFB, de acordo como especifica o NIST, permitindo assim que dados com tamanho maior que 16 *bytes* sejam utilizados. Também, caso o modo de operação seja ECB ou CBC, a classe implementa uma função de *padding*.

A seguir, na figura 5.1, pode-se visualizar parte das chamadas de funções que a classe disponibiliza.

```
class FastRijndael {
        public:

                enum Mode { ECB = 1, CBC = 2, CFB = 3, OFB = 4, CTR = 5 };
                enum KeySize { K128 = 128, K192 = 192, K256 = 256 };
                enum BlockSize { B128 = 128 };

                //CONSTRUCTORS
                FastRijndael(KeySize ks = K128, BlockSize bs = B128, Mode mode = ECB);

                //DESTRUCTOR
                virtual ~FastRijndael();

                void cleanUp();

                //KEY
                void makeKey(unsigned char* key);

                //CIPHER
                void encrypt(unsigned char* block);
                void encrypt(unsigned char* block, int &length);
                void decrypt(unsigned char* block);
                void decrypt(unsigned char* block, int &length);

                void encryptOneRound(unsigned char* block);
                void decryptOneRound(unsigned char* block);

                void encryptTwoRounds(unsigned char* block);
                void decryptTwoRounds(unsigned char* block);

                void encryptThreeRounds(unsigned char* block);
                void decryptThreeRounds(unsigned char* block);

                //IV
                void getIV(unsigned char* iv);
                void setIV(unsigned char* iv);
```

Figura 5.1: Parte de chamadas das funções da classe FastRijndael.

O algoritmo de cifragem e decifragem da classe, juntamente com os quatro modos de operação implementados, foram validados através dos dados disponibilizados pelo NIST no teste de nome *Known Answer Test Vectors*. Basicamente, consiste em arquivos de textos que contém vetores de entrada para o AES, para todas combinações entre tamnho de chave e modos de operação, devem ser verificados com a respectiva saída esperada.

## 5.2 YAAES *Inspector*

O simulador YAAES *Inspector* foi desenvolvido com o objetivo de demonstrar interativamente o processo de cifragem do AES de 128 *bits*, mostrando, a cada transformação aplicada, os valores da matriz estado, desde a entrada do texto plano até a saída do texto cifrado.

Para a visualização dos dados, foi optado por mostrá-los através do formato matricial, isto é, um bloco de 16 *bytes* é organizado em uma matriz bidimensional de quatro linhas e quatro colunas. Todos os valores dos *bytes* são representados com seu valor em base hexadecimal.

Abaixo, na figura 5.2, pode-se visualizar a tela principal do programa, a qual é dividida em duas partes distintas: os dados de entrada (texto plano e chave) e saída (texto cifrado) na parte superior, e três estados distintos da matriz estado, e as transformações envolvidas, durante o processo de cifragem na parte inferior.

Além disso, na parte inferior, encontram-se quatro botões que permitem avançar ou retroceder a aplicação das transformações sobre a matriz estado, bem como o número de

rodada em que esta matriz se encontra. Para a rodada inicial, o número correspondente é o número zero.

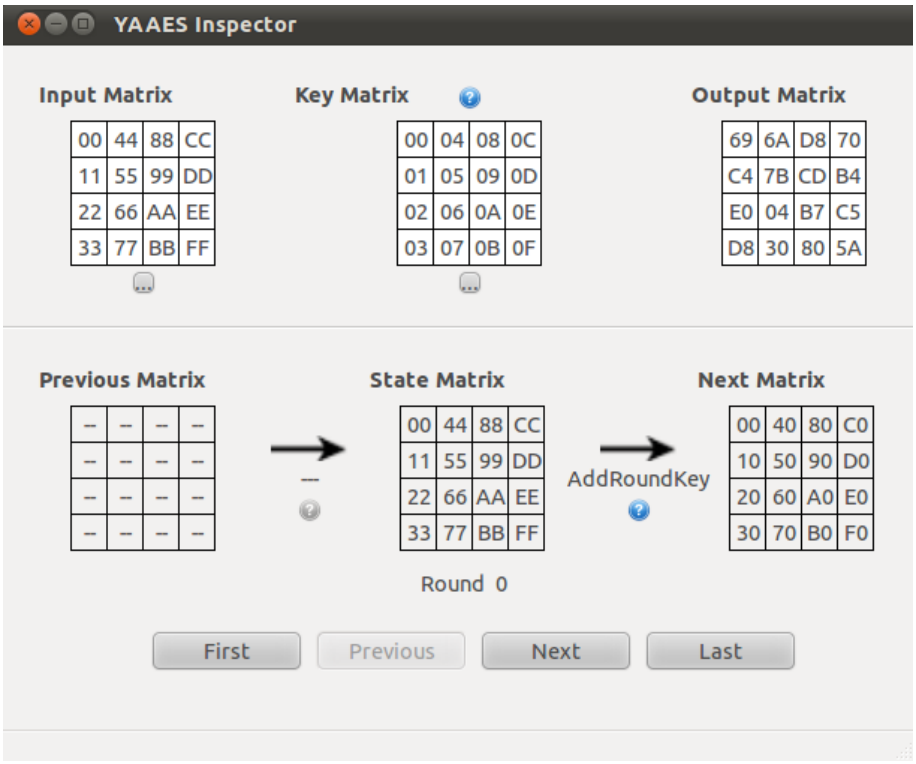


Figura 5.2: Tela principal do YAAES *Inspector*.

Através do menu da aplicação, ou clicando nos botões localizados abaixo de *Input Matrix* ou *Key Matrix* na tela principal, o usuário pode alterar os valores de entrada, isto é, o texto plano e a chave que serão utilizados. Uma janela *pop-up*, tal como a mostrada na figura 5.3, é aberta, possibilitando que o usuário entre os dados desejados em três modos distintos: *bytes* no formato matricial com valores em hexadecimal, um vetor de *bytes* em uma caixa de texto também com valores hexadecimais, ou através de texto com caracteres visíveis.

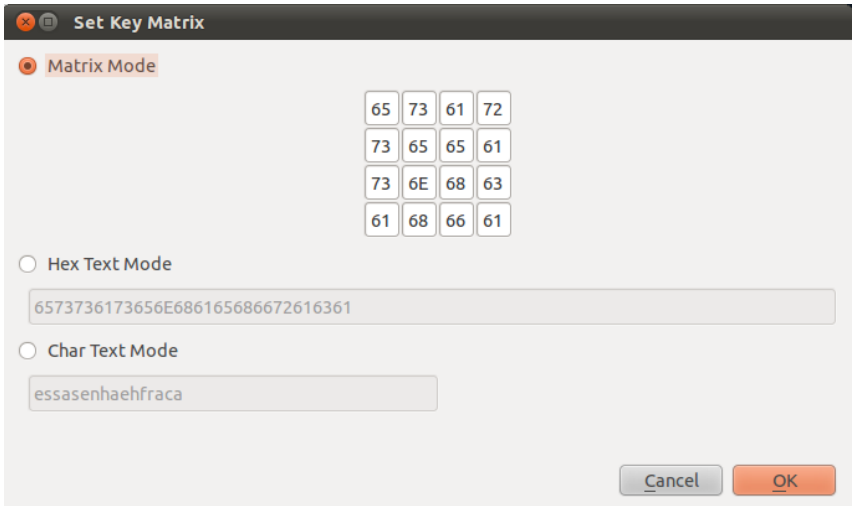


Figura 5.3: Entrada de dados do texto plano e chave.

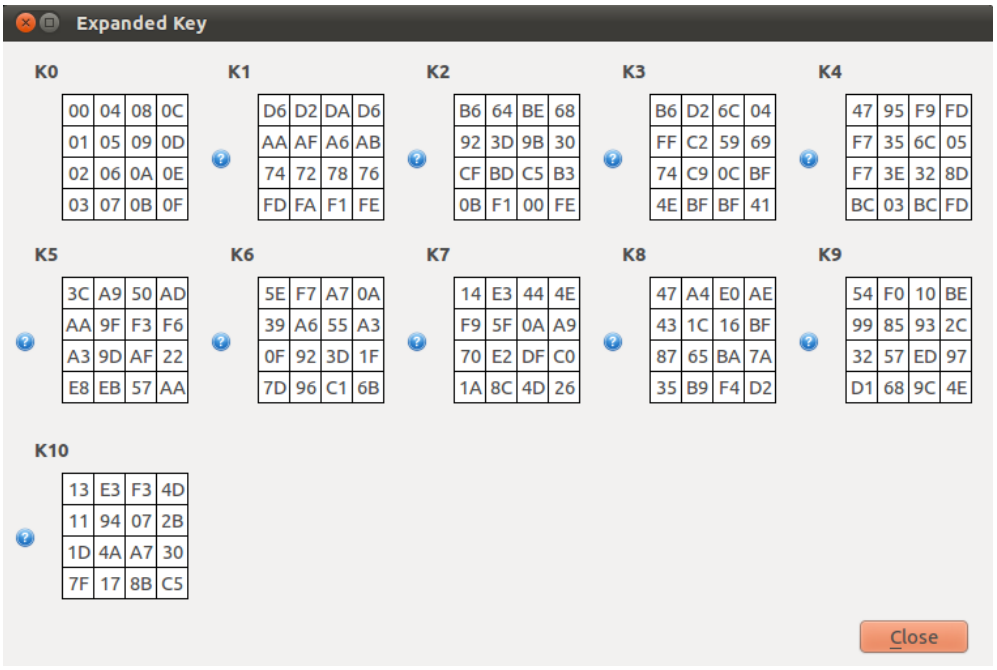


Figura 5.4: Expansão de chave.

Após entrar com os 16 *bytes* da chave, o usuário pode examinar o processo de expansão de chaves realizado no AES para a chave desejada. Para isto, deve-se clicar no botão de cor azul com um ponto de interrogação logo acima da *Key Matrix* na tela principal. Uma nova janela, então, é aberta, onde é possível visualizar as 11 subchaves geradas a partir da chave. A figura 5.4 mostra a janela de expansão de chave.

Entre cada subchave gerada, um botão clicável abre uma nova janela que permite verificar como a subchave $k_i$ foi calculada a partir da subchave anterior $k_{i-1}$. Tal janela mostra ambas subchaves, e, na subchave sendo gerada, quatro botões do tipo *radio* permitem que o usuário escolha a coluna a qual deseja ver o cálculo realizado, assim como os *bytes* utilizados para tal.

A figura 5.5 mostra a composição do cálculo para a primeira coluna de uma subchave. A figura 5.6 mostra para a segunda coluna, tal como também é realizado para a terceira e quarta coluna.

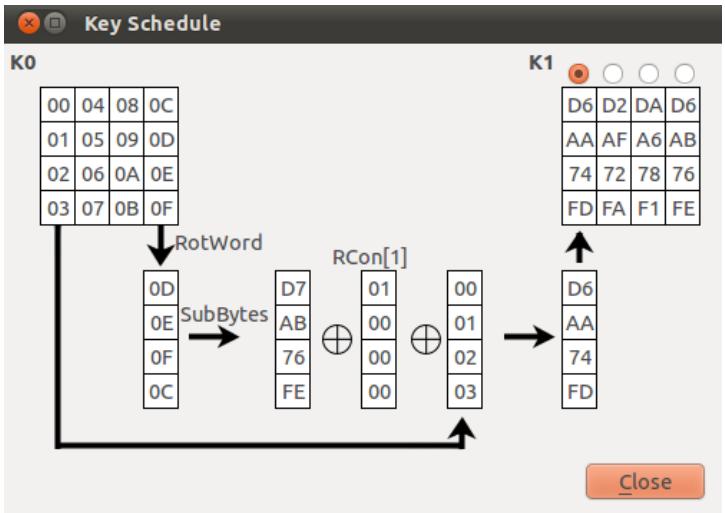


Figura 5.5: Cálculo da primeira coluna de uma subchave do AES 128.

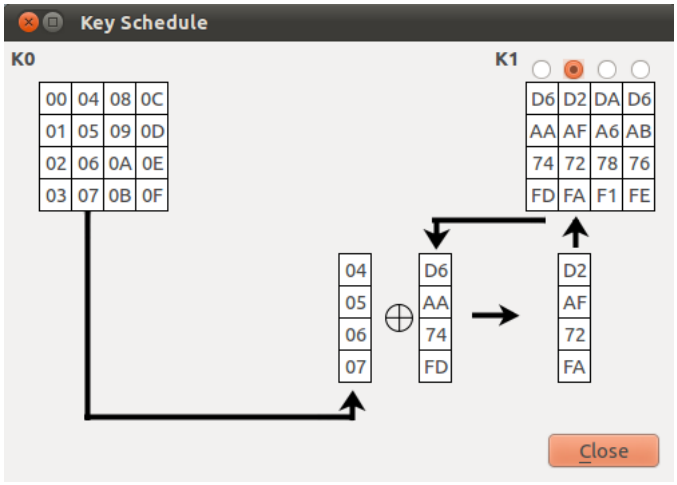


Figura 5.6: Cálculo da segunda coluna de uma subchave do AES 128.

Entre as três matrizes visualizadas na parte inferior da tela principal encontram-se as respectivas transformações que ocorrem entre um estado e outro. Ao pressionar o botão *Next*, ou o botão *Previous*, pode-se acompanhar o processo de cifragem avançar, ou retroceder, no que se refere à aplicação das transformações.

Ao lado esquerdo da matriz estado (*State matrix*) a última operação realizada sobre esta matriz é mostrada, e, ao lado direito, a próxima transformação a ser aplicada é mostrada. Em ambos os lados, quando a matriz estado não for igual ao texto plano ou ao texto cifrado, pode-se clicar no botão abaixo do nome da transformação que está sendo aplicada. Uma nova janela será aberta, demonstrando de maneira visual como a operação em questão ocorre, aplicada sobre os valores das matrizes naquele momento.

A operação de substituição de *bytes* é simplesmente representada através da aplicação dos valores da matriz em uma caixa *S-box*, de modo que os valores desta caixa são omitidos. A figura 5.7 a seguir exemplifica a operação.

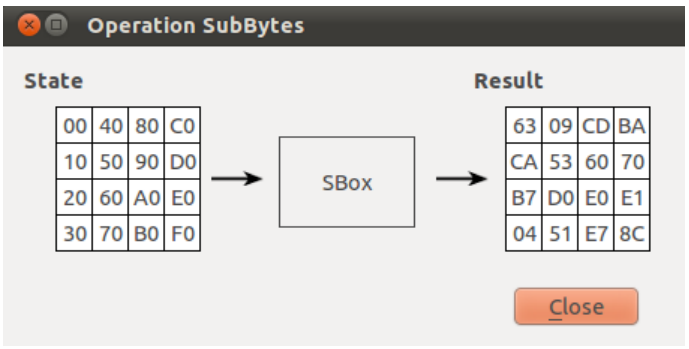


Figura 5.7: Operação de substituição de *bytes*.

As três outras operações realizadas pelo AES são apresentadas de uma forma que permite ao usuário entender como está acontecendo tal transformação, podendo comparar os valores de entrada e saída da operação.

A operação de deslocamento de linhas, na figura 5.8, demonstra através de setas de que forma os *bytes* das linhas são deslocados sobre a matriz.

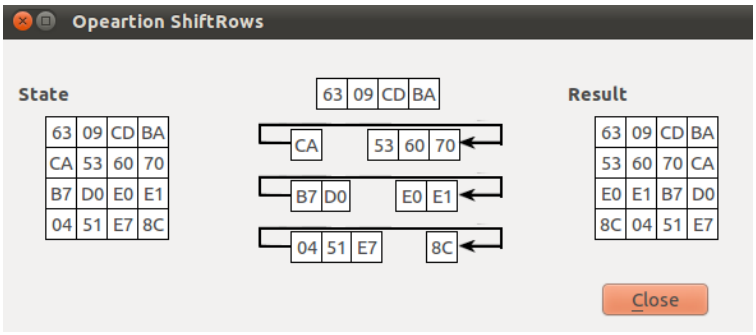


Figura 5.8: Operação de deslocamento de *bytes*.

As operações de mistura de coluna e adição de chave de rodada aparecem nas figuras 5.9 e 5.10. Sobre a primeira, pode-se ver cada coluna da matriz estado ser multiplicada pela matriz contendo os valores definidos para a operação de mistura de colunas. Deve-se notar que a multiplicação matricial apresentada não está na ordem correta, isto é, a ordem das matrizes nos operandos da multiplicação está invertida, visando manter à esquerda os valores de entrada da operação, a operação em si ao centro, e à direita o seu resultado.

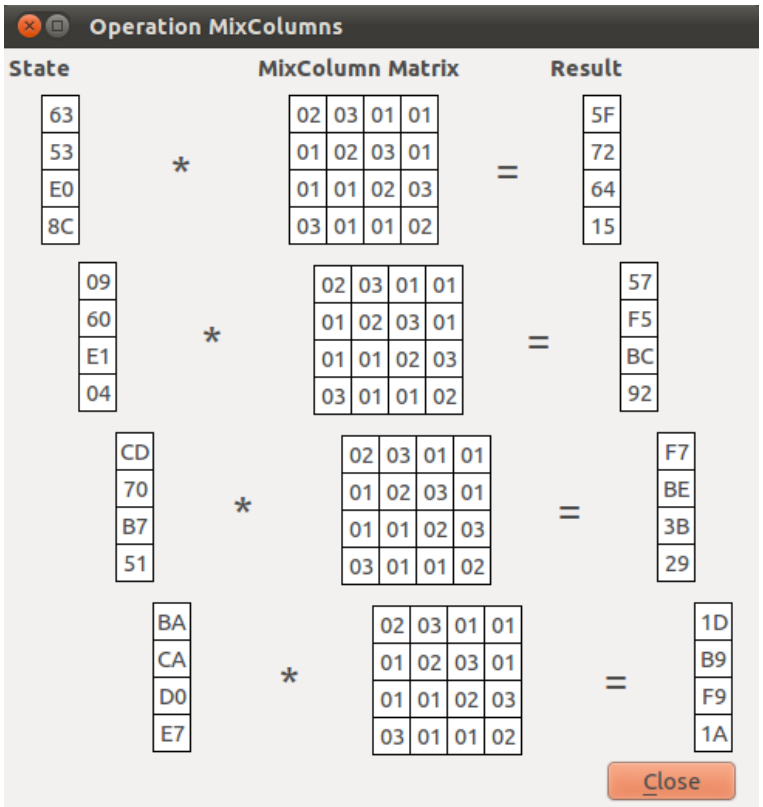


Figura 5.9: Operação de mistura de colunas.

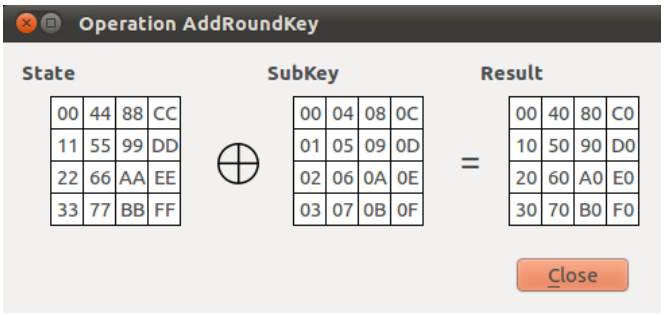


Figura 5.10: Operação de adição de chave de rodada.

## 5.3 YAAES *Attacker*

Visando apresentar visual e interativamente o ataque utilizando criptoanálise diferencial ao AES reduzido a duas rodadas, tal como demonstrado na seção 4.4, foi desenvolvido o simulador YAAES *Attacker*. Resumidamente, o simulador consiste na entrada e saída de dados do ataque, assim como as duas fases do ataque, cada uma dividida em quatro passos distintos.

Para a entrada de dados, o usuário deve informar, na aba *Input*, os três pares de textos planos/cifrados a serem utilizados, a partir dos quais quer se descobrir a chave. Por ter uma complexidade de tempo de execução relativamente alta, o usuário pode limitar o espaço de tentativas para os *bytes* de $k_0$ supostos no ataque através das caixas de texto na seção *Search Range for Key* $k_0$, localizadas na mesma aba. Caso o usuário deseje, pode escolher qual será a próxima sugestão para os valores destes *bytes* de $k_0$ durante a simulação através das caixas de texto em *Next Suggestion for* $k_0$.

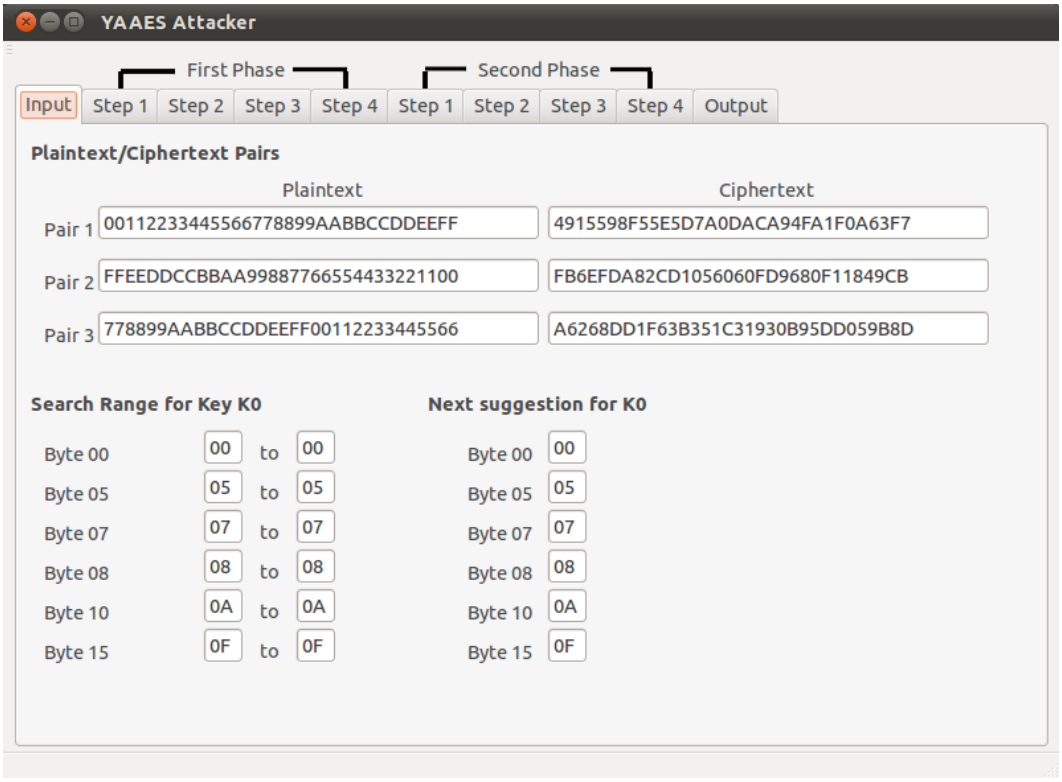


Figura 5.11: Entrada de dados para o YAAES *Attacker*.

A figura 5.11 mostra a aba *Input*. Tal como esta, todas outras figuras desta seção demonstram a execução do programa utilizando os mesmos dados que foram utilizados para exemplificar o ataque na seção 4.4.

Após entrar com os dados, o usuário pode começar a realização do passo-a-passo do ataque, alternando as abas da aplicação, de acordo com a ordem sugerida para o ataque.

O primeiro passo da primeira fase, na aba *Step 1*, acompanha a propagação dos *bytes* dos planos textos, cada um indicado pelo par de texto plano/cifrado que pertence, utilizando quatro dos *bytes* supostos para $k_0$. Pressionando os botões das transformações, a matriz em seguida é alimentada com a saída da operação aplicada à matriz anterior. Nas matrizes que ainda não contém dados conhecidos, os caracteres "??" são apresentados, indicando que o *byte* em questão será descoberto com a continuidade do ataque.

De modo similar, o segundo passo, demonstrado na aba *Step 2*, utiliza os dados dos textos cifrados, aplicando a estes a mistura de colunas inversa, através do botão *InvMC*, calcula a diferença desta saída entre os textos dos pares 1 e 2 e de 1 e 3 através do XOR, e, sobre as diferenças resultantes, aplica o deslocamento de linhas inverso.

As figuras 5.12 e 5.13 abaixo mostram ambos passos.

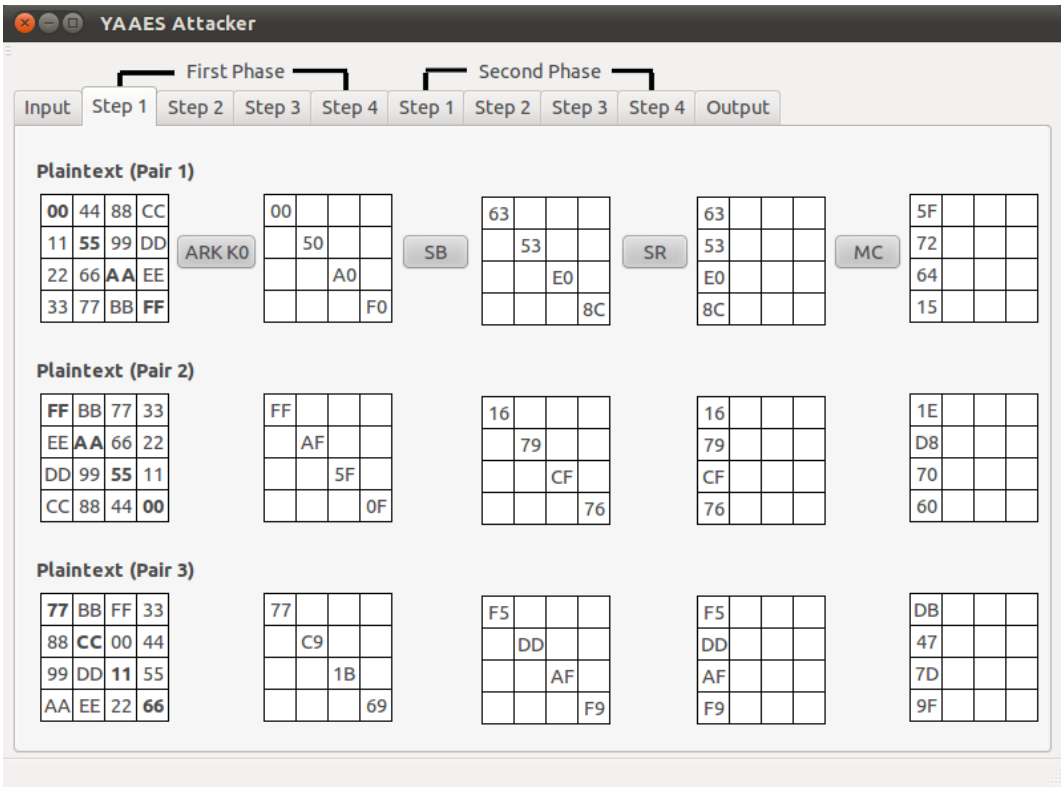


Figura 5.12: Primeiro passo da primeira fase do ataque.

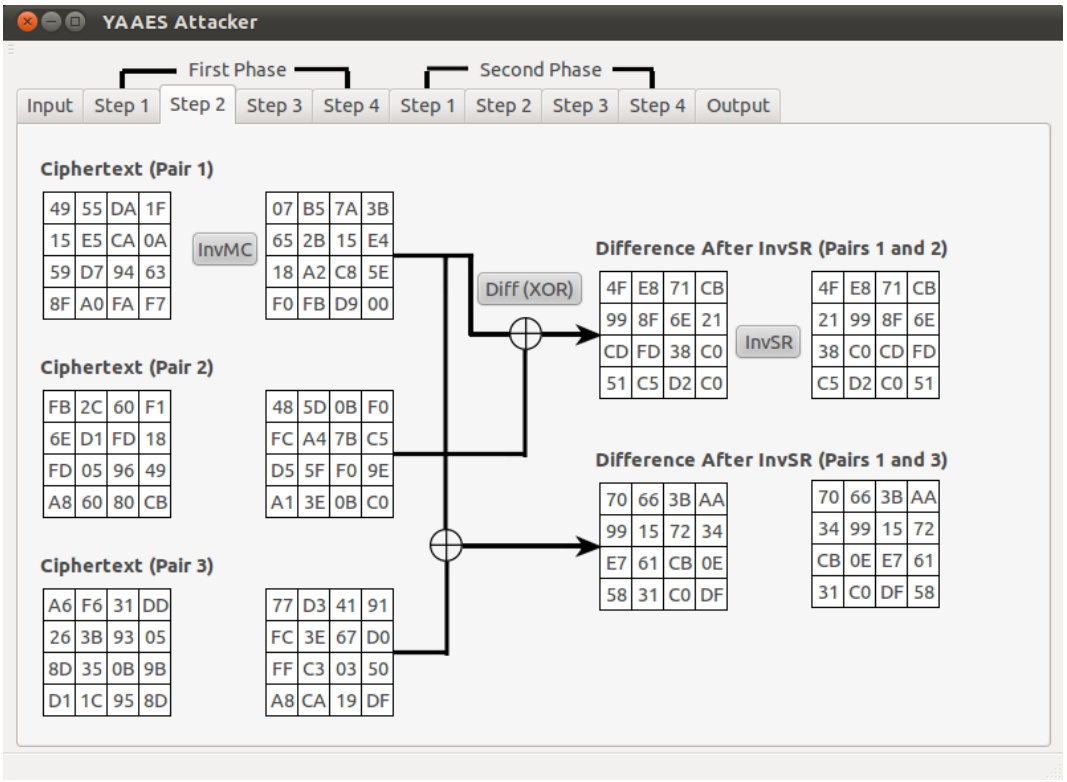


Figura 5.13: Segundo passo da primeira fase do ataque.

No terceiro passo, as colunas contendo as diferenças na entrada e saída da caixa de substituição da segunda são apresentadas. Na área central da aba *Step 3*, o usuário pode digitar os valores de *Alpha* e *Beta*, a partir das colunas de diferença entre os pares, e pressionar o botão *Lookup* para verificar qual o resultado da consulta à tabela DDT da *S-box*, sugerindo dois valores possíveis para o par (*x*, *y*). Realizando a procura para as diferenças entre os pares 1 e 2 e os pares 1 e 3, um valor em comum será encontrado, valor este que, quando inserido nas caixas de texto de *Results for Pair 1*, sugere um valor para um *byte* de $k_1$.

Caso deseje-se, também é possível, pressionando o botão *Resolve* ao lado das caixas de texto, completá-las automaticamente com o valor em comum encontrado pelo algoritmo.

O passo é visualizado na figura 5.14 abaixo.

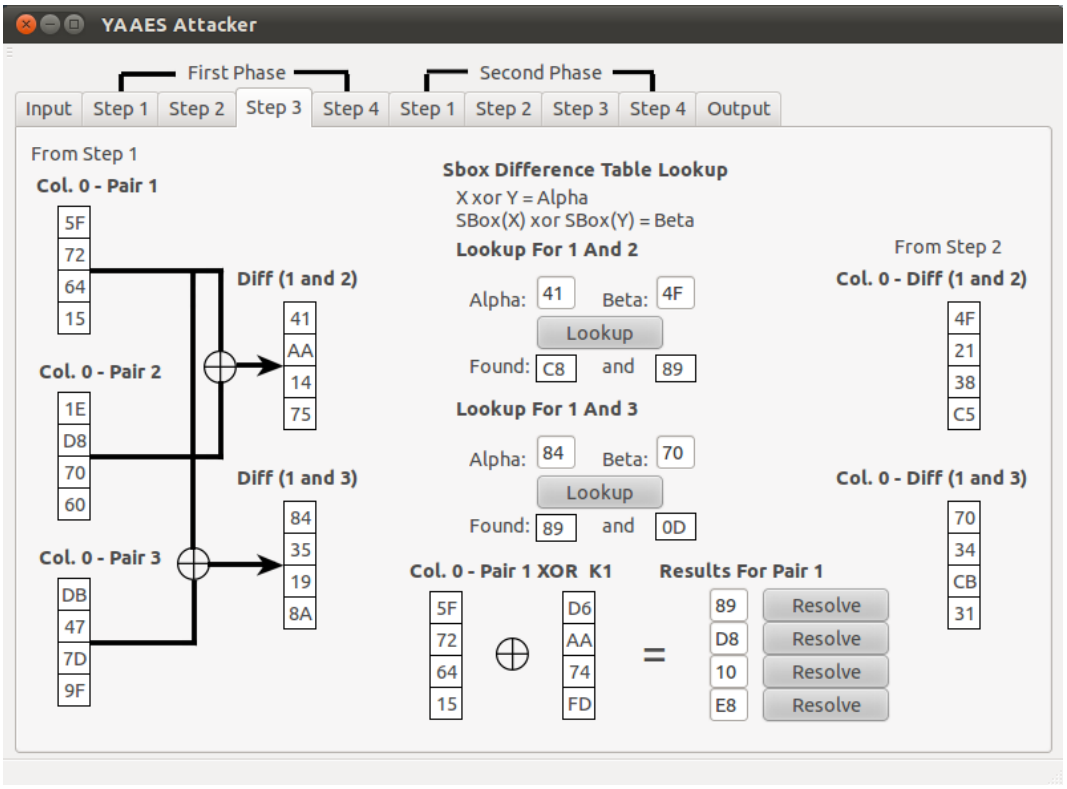


Figura 5.14: Terceiro passo da primeira fase do ataque.

Com os quatro *bytes* obtidos para $k_1$, pode-se então continuar o ataque na aba *Step 4* da primeira fase, onde a coluna referente ao texto plano do par 1 é propagada através das operações da segunda rodada de cifragem, até antes da adição da subchave $u_2$. Juntamente com os dados do texto cifrado, aplicado à mistura de colunas inversa e obtidos no passo 2 desta fase, o usuário pode calcular quatro novos *bytes* para a subchave $u_2$.

Ainda nesta fase, é possível fazer uma verificação de consistência ao propagar o *byte* da posição 0 dos textos planos dos pares 2 e 3 através de todo processo de cifragem, e comparar este valor com o encontrado também no passo 2 dos respectivos pares. Caso estes *bytes* forem diferentes, a sugestão para $k_0$ feita no início do ataque está incorreta.

Ao fim desta fase também pode-se visualizar os três *bytes* adicionais encontrados em $k_0$ e $k_1$ através das relações entre estas subchaves.

Na figura 5.15 a seguir pode-se visualizar este quarto e último passo da primeira fase.

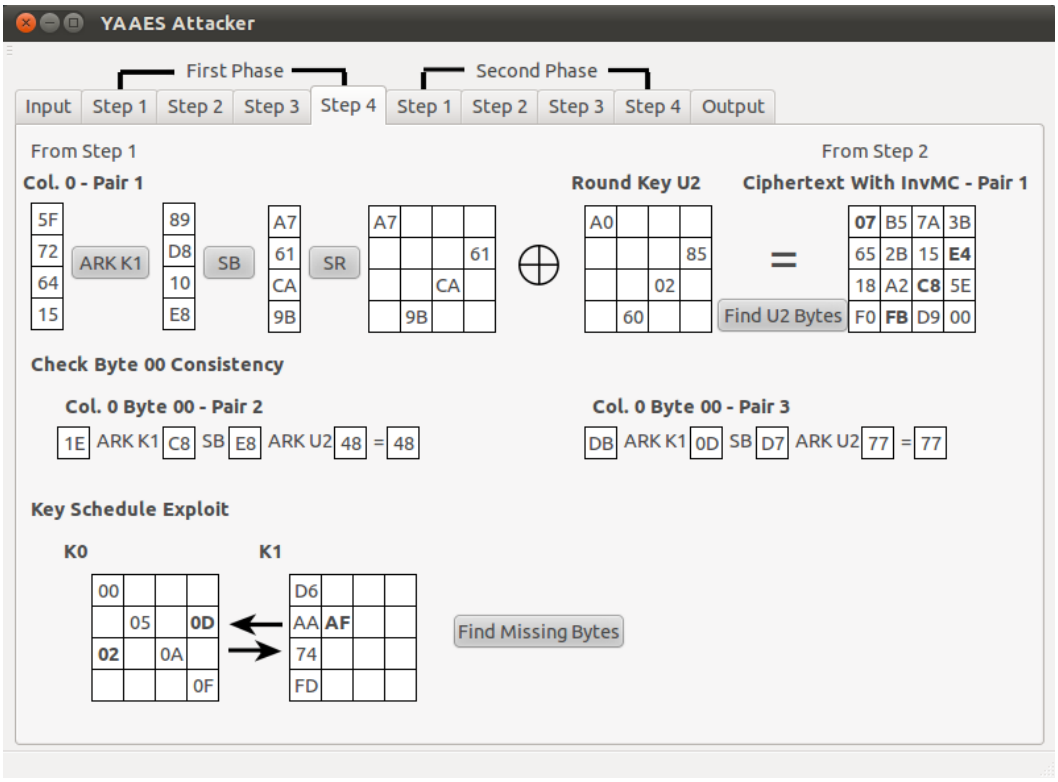


Figura 5.15: Quarto passo da primeira fase do ataque.

Durante todo o processo do ataque, o usuário pode verificar quais *bytes* das três subchaves k e da subchave $u_2$ já são conhecidos através de uma janela *pop-up*. A figura 5.16 abaixo mostra um exemplo da janela após o fim do ataque, porém pode-se visualizar o conteúdo das subchaves a qualquer momento. Através do menu da aplicação, deve-se clicar no atalho *Subkeys* para que a janela seja visualizada.

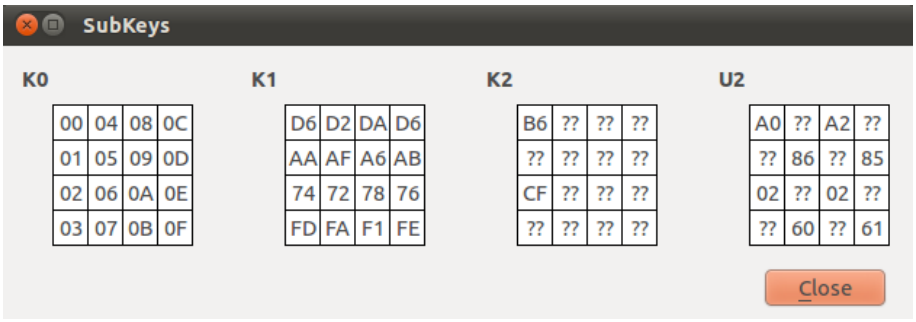


Figura 5.16: *Bytes* conhecidos das subchaves.

A segunda fase do ataque inicia, na aplicação, através da aba *Step 1* do conjunto de abas *Second Phase*. De maneira similar ao primeiro passo da primeira fase, os *bytes* dos planos textos são propagados através da primeira rodada, dado que obteve-se dois novos *bytes* de $k_0$ na fase anterior, e mais dois *bytes* estão sendo supostos pelo ataque. A figura 5.17 mostra este passo.

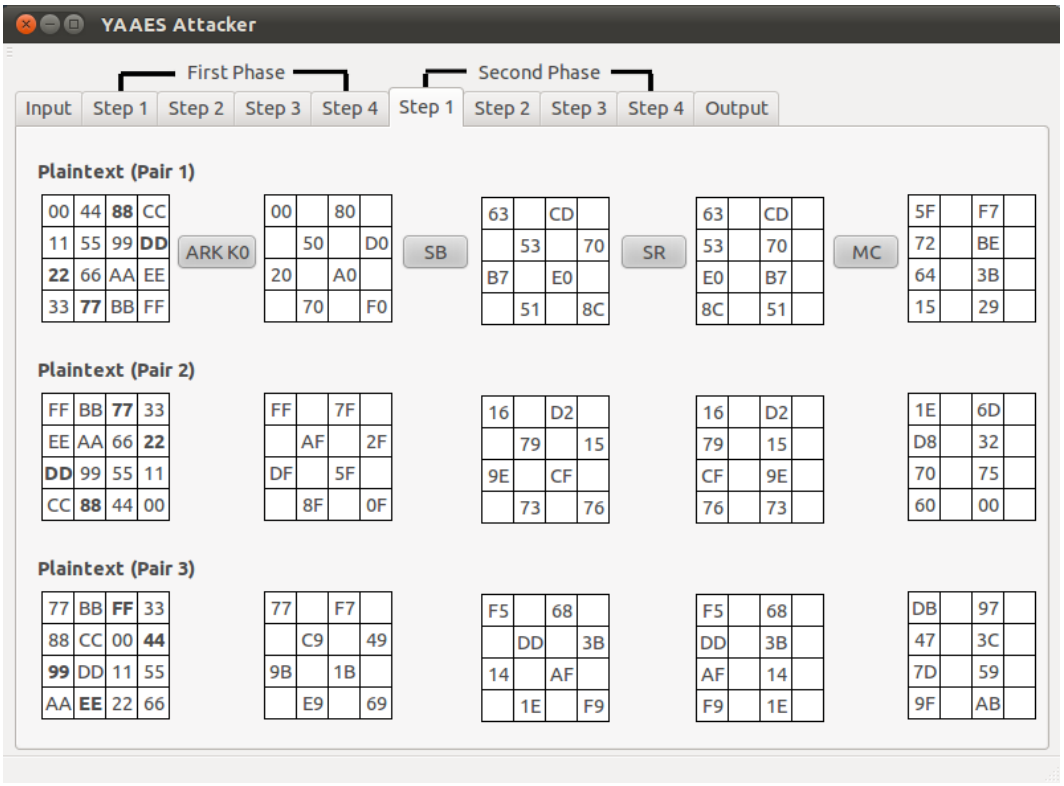


Figura 5.17: Primeiro passo da segunda fase do ataque.

Novamente, a *S-box* da segunda rodada é explorada através das diferenças de entrada e saída, porém desta vez utilizando a terceira coluna de *bytes* das diferenças. Da mesma maneira que o passo 3 da primeira fase, o usuário pode consultar e preencher o valor correto para *Results for Pair 1* manualmente, ou através do botão ao lado. Este é o segundo passo da segunda fase, que pode ser visto na figura 5.18 a seguir.

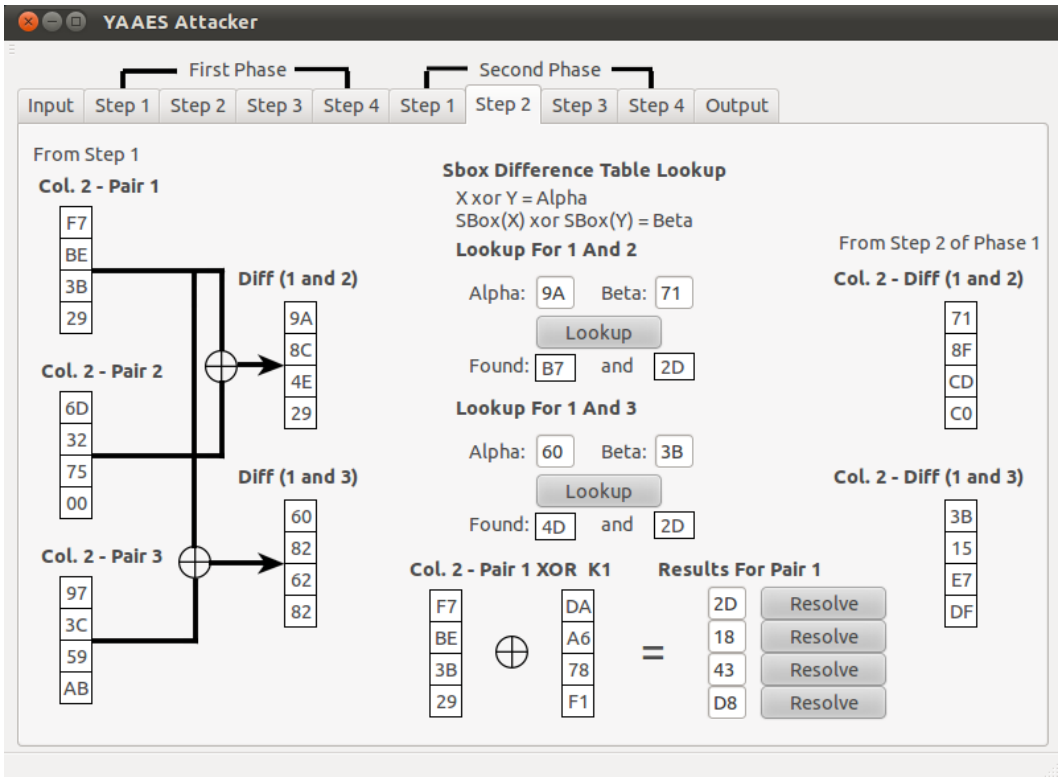


Figura 5.18: Segundo passo da segunda fase do ataque.

De tal modo como realizado no último passo da primeira fase, pode-se, no terceiro passo da segunda fase, propagar os valores do texto plano do par 1 até a adição da chave $u_2$, obtendo assim novos *bytes* desta subchave.

Ainda neste passo, sete novos *bytes*, sendo cinco de $k_1$ e dois de $k_2$, podem ser calculados através da relação entre estas subchaves, de acordo com a figura 5.19.

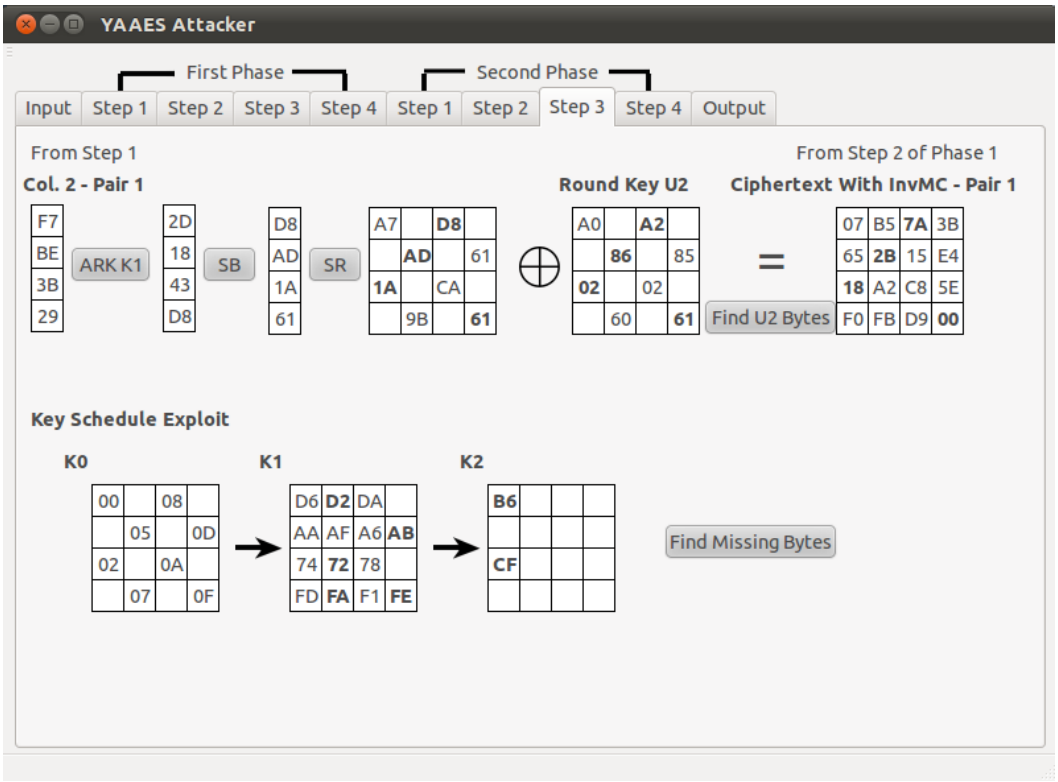


Figura 5.19: Terceiro passo da segunda fase do ataque.

No último passo da segunda fase, dado o conhecimento de quatro *bytes* da relação entre as subchaves $k_2$ e $u_2$ (onde a relação é a transformação de mistura de colunas), pode-se então construir um sistema linear contendo quatro equações e oito variáveis, sendo que quatro são conhecidas. Assim, o sistema pode ser resolvido e sua solução, isto é, novos *bytes* de $k_2$, permitem que, também através de relações entre subchaves, toda a subchave $k_1$ seja conhecida pelo usuário. Consequentemente, toda subchave $k_0$ passa a ser conhecida, visto que aquela é gerada a partir dos valores desta. A figura 5.20 demonstra este quarto e último passo da segunda fase.

Na última aba da aplicação, de nome *Output*, o usuário pode visualizar a chave $k_0$ encontrada no último passo, sempre lembrando que esta chave foi encontrada dadas as sugestões feitas no início do ataque. Ao pressionar o botão *Encrypt*, como mostra a figura 5.21, o texto plano do par 1 é criptografado utilizando a chave $k_0$ encontrada. Caso a saída desta cifragem seja igual ao texto cifrado do par 1, então a chave encontrada é a chave correta.

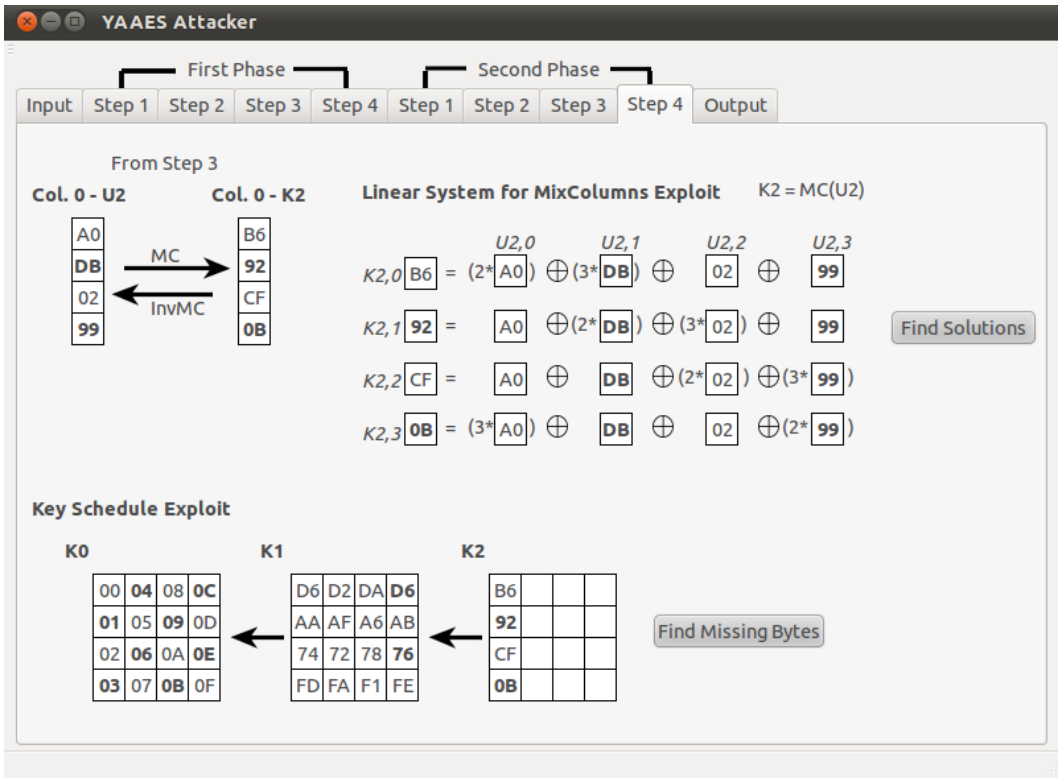


Figura 5.20: Quarto passo da segunda fase do ataque.

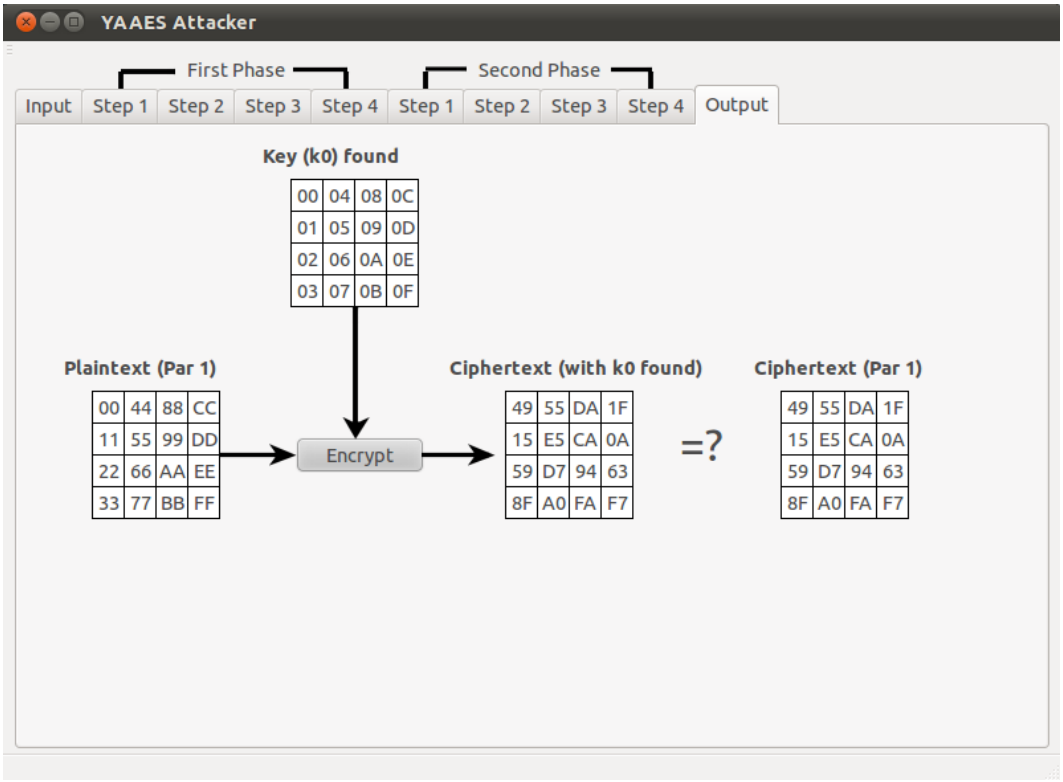


Figura 5.21: Verificação de dados do ataque.

# 6 CONCLUSÃO

Através deste trabalho foi possível demonstrar o funcionamento do algoritmo AES, tal como transformações de rodada, expansão de chave, modos de operação e outras importantes propriedades projetadas pelos seus autores para garantir a confiabilidade da criptografia realizada por este algoritmo. Com a implementação do simulador YAAES *Inspector*, pode-se visualizar, de forma didática, todo o processo de cifragem do AES de 128 *bits*.

Também abordou-se rapidamente técnicas de criptoanálise que possibilitam realizar ataques ao AES, normalmente sobre versões reduzidas do algoritmo. No que se refere à versão completa, isto é, com todas rodadas, para qualquer tamanho de chave, pôde-se entender que este algoritmo ainda é considerado seguro, dado que nenhuma criptoanálise ou ataque publicados até o momento são impraticáveis, não ameaçando assim a confiabilidade do AES.

Apesar disso, a criptoanálise do AES, por ser um padrão recente quando comparado ao DES, ainda está em expansão. Deste modo, a compreensão de métodos de ataque já desenvolvidos se torna muito importante para um criptoanalista, sabendo que tais ataques podem ser estendidos e melhorados, seja com o avanço tecnológico, seja com a descoberta de novas fraquezas no algoritmo.

Assim, este trabalho teve como segundo objetivo a demonstração de dois ataques, práticos e de complexidade de tempo factível, ao AES reduzido, utilizando para isto conceitos da criptoanálise diferencial. Para tanto, além da exemplificação completa do ataque a uma rodada e a duas rodadas, foi desenvolvido o simulador YAAES *Attacker* como ferramenta de apoio para auxiliar na visualização e entendimento do processo do ataque a duas rodadas do AES.

Concluindo, como sugestão para trabalhos futuros, tem-se a possibilidade de, através de novas descobertas nos métodos de criptoanálise do AES como a diferencial, criar novos métodos didáticos mais eficazes no que se refere ao número de rodadas e complexidade, para a criptoanálise do algoritmo AES.

# REFERÊNCIAS

[BIHAM, E., 2006] BIHAM, E.; DUNKELMAN, O.; KELLER, N. Related-key Impossible Differential Attacks on 8-Round AES-192. **Topics In Cryptology – CT RSA 2006.** Springer: [S.l.], p. 21-33, 2006.

[BIRYUKOV, A., 2009] BIRYUKOV, A.; KHOVRATOVICH, D. Related-key Cryptanalysis of the Full AES-192 and AES-256. **Advances in Cryptology: ASIACRYPT '09.** Berlin: Springer-Verlag, p. 1-18, Jun. 2009.

[BOGDANOV, A., 2011] BOGDANOV, A.; KHOVRATOVICH, D.; RECHBERGER, C. Biclique Cryptanalysis of the Full AES. **Advances in Cryptology: ASIACRYPT '11.** Berlin: Springer-Verlag, p. 344-371, Ago. 2011.

[BOUILLAGUET, C., 2010] BOUILLAGUET, C. et al. Low Data Complexity Attacks on AES. **IACR Cryptology ePrint Archive**, [S.l], p. 633-666, dez, 2010. Disponível em: <http://eprint.iacr.org/2010/633>. Acesso em: Jun. 2012.

[CHEON, J. H., 2002] CHEON, J. H., et al. Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. **ICISC '01,** London: Springer-Verlag, p. 39-49, 2002.

[COURTOIS, N., 2002] COURTOIS, N.; PIEPRZYK, J. Cryptanalysis of Block Ciphers with Overdefined System of Equations. **Advances in Cryptology: ASIACRYPT '02**, London: Springer-Verlag, p. 267-287, 2002.

[DAEMEN, J., 1997] DAEMEN, J.; KNUDSEN, L.; RIJMEN, V. The Block Cipher SQUARE. **Fast Software Encryption 1997**, [S.l.:s.n.], p. 149-165, 1997.

[DAEMEN, J., 1999] DAEMEN, J.; RIJMEN, V. **AES Proposal: Rijndael.** [S.l], v. 2, Set. 1999. Disponível em: <http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf>. Acesso em: Jun. 2012.

[DAEMEN, J., 2002-a] DAEMEN, J.; RIJMEN, V. Security of a Wide Trail Design. **Progress in Cryptology: INDOCRYPT '02.** London: Springer-Verlag, p. 1-11, Dez. 2002.

[DAEMEN, J., 2002-b] DAEMEN, J.; RIJMEN, V. **The Design of Rijndael: AES – The Advanced Encryption Standard**. New York: Springer-Verlag, 2002.

[DAEMEN, J., 2002-c] DAEMEN, J.; RIJMEN, V.; BARRETO, P. S. L. M. Rijndael: beyond the AES. **Mikulášská kryptobesídka 2002 - Third Czech and Slovak Cryptography Workshop.** Prague, Czech Republic, Dez. 2002.

[FIPS PUB 197, 2001] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Federal Information Processing Standards Publication 197**: Announcing the ADVANCED ENCRYPTION STANDARD (AES). [S.l.:s.n.], Nov. 2001. Disponível em: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. Acesso em: Jun. 2012.

[FERGUSON, N., 2010] FERGUSON, N.; SCHNEIER, B.; KOHNO, T. **Cryptography Engineering: Design Principles and Practical Applications.** Indianapolis: Wiley Publishing Inc., p. 43-70, 2010.

[FERGUSON, N., 2001] FERGUSON, N.; SCHROEPPEL, R.; WHITING, D. A simple algebraic representation of Rijndael. **Selected Areas In Cryptography 2001**. London: Springer-Verlag, p.103-111, 2001.

[FULLER, J., 2002] FULLER, J.; MILLAN, W. Linear Redundancy in S-Boxes. **IACR Cryptology ePrint Archive**, [S.l.], p. 111-128, Ago. 2002. Disponível em: <http://eprint.iacr.org/2002/111>. Acesso em: Jun. 2012.

[HEYS, H. M., 2002] HEYS, H. M. A Tutorial on Linear and Differential Cryptanalysis. **Cryptologia**, Bristol, v.26, n.3, Jul. 2002.

[KELIHER, L., 2005] KELIHER, L. Refined Analysis of Bounds Related to Linear and Differential Cryptanalysis for the AES. **Fourth Conference on the Advanced Encryption Standard,** [S.l]: Springer-Verlag, v.3373, p. 42-57, 2005.

[MATSUI, M., 1994] MATSUI, M. Linear Cryptanalysis Method for DES Cipher. **Advances in Cryptology: EUROCRYPT '93**, New York: Springer-Verlag, p. 386-397, 1994.

[MENDEL, F., 2009], MENDEL, F., et al. Improved Cryptanalysis of the Reduced Grostl Compression Function, ECHO Permutation and AES Block Cipher. **Selected Areas in Cryptography,** Berlin: Springer-Verlag, p. 16-35, 2009.

[MENEZES, A. J., 1997] MENEZES, A. J.; OORSCHOT, P. C. van; VANSTONE, S. A. **Handbook of Applied Cryptography.** Boca Raton, Florida: CRC Press, p. 223-271, 1997.

[QT4] *Framework* QT 4. Disponível em: <http://qt.nokia.com>. Acesso em: Jun. 2012.

[SCHNEIER, B., 1999] SCHNEIER, B. et al. **Performance Comparison of the AES Submissions.** [S.l.:s.n.], Fev. 1999. Disponível em: <http://www.schneier.com/paper-aes-performance.pdf>. Acesso em: Jun. 2012.

[SP 800-38A, 2001] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation**. [S.l.:s.n.], Dez. 2001. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>. Acesso em: Jun. 2012.

Wikipedia. Advanced Encryption Standard. Disponível em <http://en.wikipedia.org/wiki/Advanced_Encryption_Standard>. Acesso em: Jun. 2012.

Wikipedia. Advanced Encryption Standard Process. Disponível em <http://en.wikipedia.org/wiki/Advanced_Encryption_Standard_process>. Acesso em: Jun. 2012.

Wikipedia. Differential Cryptanalysis. Disponível em <http://en.wikipedia.org/wiki/Differential_cryptanalysis>. Acesso em: Jun. 2012.

# GLOSSÁRIO

O objetivo deste glossário é prover definições informais para termos comumente utilizados neste trabalho.

*Bloco:* consiste em uma sequência de *bytes* que denotam toda, ou parte, da entrada e saída de um processo de cifragem ou decifragem.

*Caixa de substituição:* tabela de valores que, dado valor de entrada, substitui-o de forma não-linear por um valor diferente.

*Chave:* sequência de valores, normalmente secreta, que, quando utilizada por um algoritmo de criptografia, encripta dados, gerando uma saída única que somente pode ser revertida à sua forma original por quem tiver o conhecimento da mesma.

*Chave expandida:* sequência de valores calculados a partir da chave de um algoritmo de criptografia, visando aumentar a influência da chave sobre a saída criptografada.

*Estado:* conjunto de valores que denotam o estado interno, em dado momento, do processo de criptografia.

*Matriz:* formato de representação dos valores de entrada, saída e estado de um algoritmo de criptografia, limitado a um número de linhas e colunas pré-definidas.

*Rodada:* conjunto de transformações repetidas durante o processo de criptografia.

*Subchave:* parte da chave expandida que será utilizada por uma das rodadas de um algoritmo criptográfico.

*Texto plano:* dados de entrada de um processo de cifragem, ou saída da decifragem.

*Texto cifrado:* dados de saída de um processo de cifragem, ou entrada da cifragem.

*Transformação:* denota uma operação ou cálculo que deve ser realizado sobre uma sequência de valores, tal como a matriz estado ou a chave.