



Universidade do Porto
Faculdade de Engenharia
FEUP

Processamento criptográfico
em microcontroladores de 8 bits
Aplicações à família 51

Nuno Manuel Cabral Nunes
ee99043@fe.up.pt

Departamento de Engenharia Electrotécnica e de Computadores
Porto, 22 de Julho de 2002

Processamento criptográfico em microcontroladores de 8 bits

Aplicações à família 51

Trabalho desenvolvido no âmbito da disciplina de *Projecto Final de Curso* da Licenciatura em Engenharia Electrotécnica e de Computadores, efectuado sob a orientação do Prof. Doutor João Paulo Sousa, Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto.

Conteúdo

1	Introdução	6
1.1	Motivação	7
1.2	Método de Trabalho	7
1.2.1	Calendarização	7
1.2.2	Reuniões	8
1.3	Guia de Leitura	8
2	Contexto	9
2.1	Algoritmos de bloco de chave simétrica	9
2.2	Perspectiva histórica das técnicas criptográficas de chave simétrica	9
2.3	O porquê da AES	10
2.4	O processo de selecção do standard AES	10
3	O Algoritmo AES	12
3.1	Introdução	12
3.2	Definições	12
3.2.1	Glossário de termos e acrónimos	12
3.3	Notações e convenções	14
3.3.1	Entradas e saídas	14
3.3.2	Bytes	15
3.3.3	Vector de bytes	16
3.3.4	O <i>estado</i>	16
3.3.5	O <i>estado</i> como um vector de colunas	17
3.4	Introdução matemática	17
3.4.1	Adição	17
3.4.2	Multiplicação	18
3.4.3	Multiplicação por x	19
3.4.4	Polinomiais com coeficientes no domínio finito (2^8)	19
3.5	Especificação do algoritmo	21
3.5.1	A cifra	21
3.5.2	Transformação <i>SubBytes()</i>	22
3.5.3	Transformação <i>ShiftRows()</i>	23
3.5.4	Transformação <i>MixColumns</i>	24
3.5.5	Transformação <i>AddRoundKey()</i>	25
3.6	Expansão da Chave	26
3.7	Cifra Inversa	27

3.7.1	Transformação <i>InvShiftRows()</i>	28
3.7.2	Transformação <i>InvSubBytes()</i>	28
3.7.3	Transformação <i>InvMixColumns()</i>	30
3.7.4	Inversa da transformação <i>AddRoundKey()</i>	30
3.7.5	Equivalente da Cifra Inversa	30
4	Modos de Funcionamento	33
4.1	Electronic Codebook	34
4.2	Cipher Block Chaining	34
4.3	Cipher Feedback	35
4.4	Output Feedback Mode	37
4.5	Counter	38
5	Implementação	40
5.1	Aspectos a ter em conta	40
5.1.1	Ataques de <i>temporização</i>	41
5.1.2	Paralelismo	41
5.1.3	Implementação da cifra inversa	41
5.2	A Multiplicação	42
5.2.1	No domínio finito 2^8	42
5.2.2	Através de <i>shift</i>	43
5.3	Pormenores da implementação no microcontrolador em questão	44
5.3.1	Especificação da cifra	44
5.3.2	A rotina <i>cifra</i>	45
5.3.3	GetRoundKey	46
5.3.4	Considerações quanto ao resto das rotinas	47
5.4	Validação da implementação efectuada	47
5.5	Comparação com outras implementações	47
6	Ensaaios	51
6.1	Descrição da aplicação	51
6.2	Descrição do hardware	52
6.2.1	Placa utilizada	52
6.2.2	Pormenores de hardware	52
6.3	Pormenores do software	52
6.4	Ensaaios efectuados	53
7	Conclusão	54
A	CVS	55
A.1	Introdução	55
A.2	O que é ?	55
A.3	Instalar e configurar <i>CVS</i>	56
A.3.1	Instalação inicial	56
A.3.2	Definição de variáveis de ambiente	57
A.3.3	Configuração	57
A.3.4	cvs na Web	59
A.4	Para o utilizador	59
A.4.1	O que fazer primeiro	60

A.4.2	Como importar código fonte no repositório	60
A.4.3	Como verificar a fonte através de SSH	60
A.4.4	Como verificar a fonte anonimamente através do <i>pserver</i> .	60
A.5	Referências	61

Lista de Tabelas

3.1	Combinações que respeitam o standard	21
5.1	Multiplicação através da rotação	44
5.2	Tamanho do Código Implementado	48
5.3	Resultados da implementação usada para o concurso AES	49

Lista de Figuras

3.1	Entrada e saída do vector estado	16
3.2	Pseudo-código da cifra	22
3.3	A transformação <i>SubBytes()</i>	23
3.4	Valores de substituição para o byte <i>xy</i> (no formato hexadecimal)	24
3.5	A transformação <i>ShiftRows()</i>	24
3.6	A transformação <i>MixColumns()</i>	25
3.7	Transformação <i>AddRoundKey()</i>	26
3.8	Pseudo-código para a expansão da chave	27
3.9	Pseudo-código para a cifra inversa	28
3.10	A transformação <i>InvShiftRows()</i>	29
3.11	<i>S-Box</i> Inversa	29
3.12	Pseudo-código para a cifra inversa equivalente	32
4.1	O modo <i>Electronic Codebook</i>	35
4.2	O modo <i>Cipher Block Chaining</i>	36
4.3	O modo <i>Cipher Feedback</i>	37
4.4	O modo <i>Output Feedback</i>	38
4.5	O modo <i>Counter</i>	39
5.1	A importância da operação de multiplicação na cifra	42
5.2	Comparação do resultado da cifra com os valores de teste	47
5.3	Tempo de execução do Código Implementado	48
6.1	Esquema da apresentação efectuada	51

Capítulo 1

Introdução

Neste relatório discute-se a implementação do *Advanced Encryption Standard* (AES) num microcontrolador de 8 bits.

AES é baseado na cifra de bloco *Rijndael* e foi designado como o sucessor do *Data Encryption Standard* (DES), que tem sido implementado em diversos módulos por todo o mundo desde 1977.

O microcontrolador de 8 bits utilizado foi um *clone* do 8051 da família i51 da Intel. Graças às propriedades da cifra (orientada ao byte), é possível a sua implementação num microcontrolador deste tipo. Embora esta implementação seja completamente funcional, dado o nível académico em que se insere, não foi dada especial importância à velocidade de execução, mas sim à sua compreensibilidade.

Programadores de assembler bradariam aos céus em agonia quando vissem uma implementação de funções de rotação feitas algebricamente, passo a passo, ao contrário de uma troca de índices de apontadores. Nesta implementação *educacional* optei por explicar *o que deve fazer* em vez de *fazer o mais rápido possível*. No entanto, não menosprezei alguns aspectos de implementação que são explicados em capítulo próprio.

Ficam assim abertas as portas, para outras implementações orientadas a um mais baixo nível possivelmente em detrimento da sua legibilidade, aspecto esse que optei por melhorar nesta implementação.

Neste relatório são também dados antecedentes essenciais para a compreensão do algoritmo utilizado, nomeadamente nas técnicas, aplicações e modos de funcionamento da cifra, no entanto, este relatório, dado o seu âmbito, não se debruça profundamente no *porquê* de ser este o novo standard (e não outro qualquer), i.e., não se debruça sob o comportamento do standard ao nível da resistência a uma criptoanálise linear e diferencial (Para os interessados neste assunto específico, são incluídos na bibliografia os documentos que suportam a proposta para o standard[18], ou recomenda-se um bom livro de criptografia[21]).

Os aspectos matemáticos essenciais à correcta compreensão do algoritmo são apresentados a um nível pragmático utilizando apenas os antecedentes matemáticos estritamente necessários, sendo por isso, acessível a um leque mais alargado de leitores.

1.1 Motivação

Desde à muito tempo que organizações governamentais usam métodos criptográficos para manter os seus dados fora dos olhares curiosos quer do público em geral, quer de possíveis inimigos de guerra. Foram feitos esforços quer para aperfeiçoar os seus métodos, quer para tentar quebrar os métodos dos inimigos. Durante alguns anos, os métodos criptográficos foram desenvolvidos exclusivamente para fins militares.

Hoje em dia a criptografia é utilizada no nosso dia a dia, quando consultamos o saldo bancário numa vulgar caixa automática, ou quando acedemos ao nosso site preferido de compras online.

De tempo em tempo, são desenvolvidos ou aperfeiçoados algoritmos que garantam uma maior segurança, quer devido a uma falha no algoritmo original, ou mesmo devido à evolução tecnológica que inviabiliza a segurança do algoritmo até então considerado seguro.

Desde a década de 70, o standard utilizado para várias aplicações e recomendado pelo *NIST*¹ como um standard para a utilização em dados federais, foi a cifra *DES*².

Graças aos avanços tecnológicos, considerou-se essa cifra insegura para algumas aplicações, tendo mesmo sido recomendada a utilização do *triplo DES* como medida de aumento de segurança.

Foi então necessário o desenvolvimento de um novo standard (AES) que oferecesse uma maior segurança, sendo escolhido, depois de um longo e exaustivo processo de selecção, o algoritmo *Rijndael*.

Actualmente, o DES continua a ser usado em produtos descontinuados, dado que ainda oferece uma segurança relativa, sendo a grande maioria dos novos produtos baseados neste novo standard.

Este projecto compreendeu a implementação da cifra AES num microcontrolador de 8 bits, que poderá vir a ser usada em diversas aplicações.

1.2 Método de Trabalho

1.2.1 Calendarização

Numa primeira fase, foi necessária a recolha de informação acerca deste novo standard. Utilizei principalmente a internet, dado que devido a ser um assunto relativamente novo, ainda não existem livros (o único livro conhecido até ao momento foi editado em 25 de Fevereiro de 2002) e a internet revelou-se assim essencial para esta recolha.

Numa fase seguinte procedi ao estudo da cifra. Foi necessário perceber o funcionamento completo da cifra incluindo as componentes matemáticas, e todas as transformações que ela contém.

Posteriormente passei à implementação no microcontrolador escolhido, utilizando a placa de demonstração *CORE-51*. Foi a implementação da cifra que ocupou a maior parte do tempo que tive disponível para a execução deste projecto.

¹National Institute of Standards and Technology

²Data Encryption Standard

Simultaneamente fui recolhendo e analisando mais informação acerca de parâmetros de implementação e suas implicações.

Para uma aplicação prática depois de alguma indecisão acerca da criação de um gerador de números aleatórios baseado na cifra, optei por estabelecer uma comunicação cifrada num canal serial em que é demonstrada a cifra em funcionamento.

1.2.2 Reuniões

Para uma orientação do projecto, houve reuniões semanais com o meu orientador, Prof. Dr. João Paulo Sousa.

Estas reuniões revelaram-se essenciais para o estado actual do projecto, graças a uma discussão aberta entre os caminhos a tomar, assim como a definição de objectivos intermédios e suas exigências temporais.

Estas reuniões levaram a uma eficiência acrescida na organização do esforço dispendido.

1.3 Guia de Leitura

No capítulo 2 é dado ênfase ao contexto que levou à criação deste novo standard, referindo para isso a perspectiva histórica das técnicas criptográficas assim como o processo de selecção que levou à sua criação.

O capítulo 3 explica o funcionamento e todos os pormenores da cifra AES. Contém as definições dos termos utilizados neste relatório e poderá ser omitido para os leitores já familiarizados com estes termos. As notações e convenções usadas no algoritmo são essenciais para a compreensão da cifra. A introdução matemática explica as operações que são especificadas no algoritmo. Neste capítulo encontra-se a descrição pormenorizada das várias operações que constituem a cifra e a cifra inversa.

No capítulo 4 são referenciados os modos de funcionamento da cifra que fornecem serviços de confidencialidade. O capítulo 5 são explicados os aspectos e pormenores a ter em conta na implementação. O capítulo 6 tem em conta os ensaios que foram efectuados descrevendo assim a aplicação final e alguns pormenores de hardware e software.

O capítulo 7 é, como era esperada, a conclusão deste relatório onde estão referidas a conclusão aos objectivos propostos.

Como apêndices, incluo o apêndice A que descreve a utilização do CVS e as suas vantagens e um pequeno guia de instalação e configuração do servidor propriamente dito e do módulo para interface via web.

O apêndice ??, contém o código final efectuado.

A bibliografia mostra os documentos que consultei e que são recomendados ao longo do relatório, para a realização deste projecto.

Capítulo 2

Contexto

Neste capítulo é explicado o contexto em que o standard *AES* se insere assim como as motivações que levaram ao seu aparecimento. É também indicado o processo e os critérios utilizados na sua selecção.

2.1 Algoritmos de bloco de chave simétrica

Existem basicamente dois tipos de criptografia, a criptografia simétrica ou de chave privada e a criptografia assimétrica ou de chave pública.

Os algoritmos de chave simétrica, também chamados de algoritmos convencionais, são a forma mais antiga de criptografia, nos quais a chave para a cifra pode ser calculada a partir da chave da cifra inversa e vice-versa. Na maioria dos algoritmos de chave simétrica, a chave chega a ser igual, como é o caso do algoritmo AES.

Nestes algoritmos é necessário que no início de qualquer comunicação (se for essa a aplicação), o emissor e o receptor tenham uma chave secreta para que possam comunicar. A segurança da comunicação, depende assim, do secretismo dessa chave. Caso a chave seja conhecida por mais alguém, esse alguém pode conhecer quer o conteúdo de comunicações futuras, quer mesmo das comunicações já efectuadas.

Alguns algoritmos operam bit a bit no texto *normal* (*stream cipher*). Outros operam em blocos de bits do texto *normal* e são chamados de cifra de bloco *block cipher*. A cifra AES é uma cifra que actua em blocos de 128 bits.

2.2 Perspectiva histórica das técnicas criptográficas de chave simétrica

Uma cifra de chave simétrica é, basicamente um método de ocultação de informação, em que a chave para a cifragem (conversão de texto simples, para texto cifrado) é a mesma para a cifra inversa (conversão de texto cifrado em texto simples).

As cifras de chave simétrica são historicamente as mais antigas, havendo relatos da sua utilização no império romano.

Desde 1977 que o *DES*¹ é o algoritmo de encriptação mais utilizado. Neste instante ainda é o standard usado pelas instituições federais, no entanto, são conhecidas implementações que possibilitam a decifragem numa questão de horas.

2.3 O porquê da AES

Graças ao crescimento exponencial do poder de computação, este algoritmo já não é considerado seguro. Isto foi provado por acções da E.F.F. (Electronic Frontier Foundation), que ganhou consecutivamente os concursos de quebrar a cifra DES pelos Laboratórios RSA. Foi construída uma máquina especial para quebrar a cifra com um custo abaixo de \$250000 que em 18 de Julho de 1998 conseguiu quebrar a cifra DES em menos de 3 dias.

O NIST reconheceu a insuficiência de protecção que o DES oferecia, e em 1999 recomendou a utilização do DES triplo. Curiosamente, 6 anos antes, o NIST, acerca da possibilidade de através do teste de todas as chaves possíveis, escreveu: *... the feasibility of deriving a particular key in this way is extremely unlikely in typical threat environments.*

Existem variantes da cifra DES (o DES triplo, que consiste na aplicação repetida por três vezes da cifra DES) que ainda são relativamente seguras, mas cuja performance é superada por outras cifras mais rápidas e eficazes.

Surge assim a necessidade de criação de um novo standard de seja mais seguro que o *DES*.

2.4 O processo de selecção do standard AES

O processo de selecção de um novo standard começou a 2 de Janeiro de 1997 através do NIST, com uma *workshop* pública para o estabelecimento de critérios mínimos e quais os critérios para a avaliação das possíveis propostas. A 12 de Setembro de 1997, iniciou-se formalmente a procura e desenvolvimento de candidatos para o novo standard. Os algoritmos candidatos tinham de responder aos seguintes critérios :

- O algoritmo tinha de implementar cifragem de chave simétrica.
- O algoritmo tinha de ser uma cifra de bloco.
- O algoritmo tinha de implementar chaves e blocos com tamanhos com as combinações de 128-128, 192-128 e 256-128 bits.

Foram então aceites as várias propostas que foram avaliadas segundo os critérios:

Segurança - A *qualidade* do algoritmo em relação a outras propostas. As propostas foram analisadas por *criptoanalistas* que se pronunciaram quanto à segurança do algoritmo em si.

¹Data Encryption Standard

Licença - O algoritmo AES devia ser de livre uso em todos os lugares, para uso não exclusivo e livre de jónias de utilização.

Eficiência computacional e suas exigências - A velocidade de execução do algoritmo em relação ao custo do hardware em que é executado.

Flexibilidade - Além dos tamanhos de chave e de bloco recomendados, os algoritmos foram analisados tendo em conta uma possível expansão futura, através do aumento dos tamanhos da chave aumentando assim, a sua segurança.

Possibilidade de implementação em hardware e software - Foi analisada a possibilidade de execução por software, mas sem menosprezar as implementações em hardware.

Simplicidade - Os algoritmos foram avaliados também pela sua simplicidade.

Em Agosto de 1998, o NIST anunciou 15 algoritmos candidatos na primeira conferência para a selecção do algoritmo AES.

Estes algoritmos foram analisados minuciosamente e os resultados foram apresentados e discutidos em Março de 1999 numa segunda conferência.

Até 15 de Abril de 1999, foram recebidos e discutidos os comentários públicos a estes algoritmos sendo apenas seleccionados cinco.

Estes cinco algoritmos foram continuamente analisados quer pelo NIST quer pelo público em geral.

Finalmente a 2 de Outubro de 2000, o NIST anunciou o algoritmo *Rijndael* como o vencedor da competição. Os criadores deste algoritmo foram os criptógrafos *Joan Daemen* e *Vincent Rijndael*, ambos prestigiados peritos na comunidade internacional de criptografia.

Capítulo 3

O Algoritmo AES

3.1 Introdução

Os três critérios no projecto da cifra AES foram:

- A resistência contra todos os ataques conhecidos
- A velocidade e a optimização do código em diversas plataformas
- A simplicidade de *design*.

Ao contrário da maioria das cifras que normalmente fazem a transposição sem qualquer modificação de alguns bits do *estado* intermédio para outra posição, no algoritmo AES, cada *rotação* é composta por três transformações uniformes¹, distintas e invertíveis, chamadas de *layers*.

Cada *layer* tem uma função específica:

Mistura linear Garante uma boa difusão ao longo de múltiplas *rotações*.

Mistura não linear A aplicação paralela de *S-box*² com propriedades não lineares óptimas (no pior dos casos).

Adição de chave A operação *EXOR* da chave de rotação com o *estado* intermédio.

Neste capítulo descrevem-se as notações usadas no resto do documento, uma leve introdução matemática ao algoritmo, a especificação do algoritmo em si e a descrição das operações de expansão da chave e a operação de cifra inversa.

3.2 Definições

3.2.1 Glossário de termos e acrónimos

- Multiplicação no domínio finito

¹Cada bit do estado é tratado da mesma maneira

²Ver Sec. 3.2.1

\oplus O m. q. *XOR*

\otimes Multiplicação de dois polinômios (cada um com grau inferior a quatro) com módulo $x^4 + 1$.

K Chave da cifra.

Nb Número de colunas (palavras de 32 bits) que o *estado* contém. Para o AES é igual a quatro.

Nk Número de palavras de 32 bits que a cifra contém. Para o AES é igual a quatro, seis ou oito.

Nr Número de rotações, que é uma função de *Nk* e *Nb* (que são constantes). Para o AES é igual a dez, doze ou catorze.

AddRoundKey() Transformação na cifra e na cifra inversa em que uma chave de rotação é adicionada ao *estado* através de uma operação XOR (\oplus). O tamanho de uma chave de rotação é igual ao tamanho do *estado* (i.e., para *Nb* = 4, o tamanho da chave de rotação é 128 bits (16 bytes))

InvMixColumns() Transformação na cifra inversa que é a inversa da **MixColumns()**.

InvShiftRows() Transformação na cifra inversa que é a inversa da **ShiftRows()**.

InvSubBytes() Transformação na cifra inversa que é a inversa da **SubBytes()**.

MixColumns() Transformação na cifra que tem como entrada todas as colunas do *estado* e mistura os seus dados (independentemente das colunas) de forma a produzir novas colunas.

RotWord() Função usada na rotina de expansão da chave que tem como entrada uma palavra de quatro bytes e executa uma permutação cíclica.

ShiftRows() Transformação na cifra que processa o *estado* através de uma rotação cíclica das suas três últimas linhas com diferentes desvios.

SubBytes() Transformação na cifra que processa o *estado* através de uma tabela de substituição não linear (*S-box*). Esta tabela opera em cada byte do *estado* independentemente.

SubWord() Função usada na rotina de expansão da chave que tem como entrada uma palavra de quatro bytes e aplica a *S-box* a cada um dos quatro bytes de forma a produzir uma palavra de saída.

XOR Operação *OU* exclusiva

afinidade, transformação de Transformação que consiste na multiplicação por uma matriz, seguida pela adição de um vector.

bit Um dígito binário consistindo unicamente nos valores 0 ou 1.

bloco Sequência de bits que é composta pela entrada, saída, estado e a chave de rotação. O tamanho da sequência é o número de bits que contém. Blocos são convenientemente interpretados como vectores de bytes.

- byte** Um grupo de oito bits que é tratado como uma entidade separada ou como um vector de 8 bits individuais.
- Chave (da cifra)** Chave secreta que é usada na rotina de expansão da chave para gerar as chaves de rotação (usadas a cada rotação). Pode ser visualizada como um vector rectangular de bytes, tendo quatro linhas e Nk colunas.
- Chave de rotação** Valores derivados da chave da cifra através da utilização da rotina de expansão da chave. É aplicada ao vector *estado* na cifra e na cifra inversa.
- Cifra inversa** Séries de transformações que convertem texto cifrado *ciphertext* em *texto simples* usando a chave da cifra.
- Cifra** Séries de transformações que convertem *texto simples* em texto cifrado usando a chave da cifra.
- Ciphertext (*texto cifrado*)** Corresponde aos valores resultantes da cifra, ou que são a entrada na operação de cifra inversa.
- estado** Resultado intermédio da cifra que pode ser visualizado como um vector rectangular de bytes, tendo quatro linhas e Nb colunas.
- Expansão da chave** Rotina usada para gerar uma série de chaves de rotação através da chave da cifra.
- palavra** Um grupo de 32 bits que é tratado como uma entidade simples ou como um vector de 4 bytes.
- S-box** Tabela de substituição não linear usada em várias substituições de bytes, e na rotina de expansão de cifra de forma a executar uma substituição unívoca entre bytes.
- texto simples** Corresponde à entrada na operação de cifra, ou à saída na operação de cifra inversa.
- Vector** Uma colecção enumerada de entidades idênticas (por exemplo, um vector de bytes).

3.3 Notações e convenções

3.3.1 Entradas e saídas

A entrada e a saída do algoritmo AES consiste em sequências de 128 bits. Estas sequências são também referidas como *blocos* e o número de bits que estes contêm é chamado de *tamanho*.

A chave de cifra para o algoritmo AES é uma sequência de 128, 192 ou 256 bits. Os bits nestas sequências são numerados a partir do zero e acabam no tamanho da sequência subtraído de uma unidade.

O número i como sufixo de um bit é uma referência ao seu índice, e estará nas gamas $0 \leq i < 128$, $0 \leq i < 192$ ou $0 \leq i < 256$ dependendo do tamanho do bloco e do tamanho da chave.

3.3.2 Bytes

A unidade básica para o processamento do algoritmo AES é o *byte* (uma sequência de oito bits tratada com uma entidade única).

As sequências de entrada, saída e chave de cifra (descritas na Sec. 3.3.1) são processadas como vectores de bytes que são transformados através da divisão destas sequências em grupos de oito bits contínuos para formar vectores de bytes.

Para uma entrada, saída ou chave de cifra denotada como a , os bytes no vector resultante serão referenciados numa das formas a_n ou $a[n]$, onde n estará numa das seguintes gamas:

- Tamanho da chave = 128 bits; $0 \leq n < 16$
- Tamanho da chave = 192 bits; $0 \leq n < 24$
- Tamanho da chave = 256 bits; $0 \leq n < 32$
- Tamanho do bloco = 128 bits; $0 \leq n < 16$

Todos os bytes no algoritmo serão apresentados como a concatenação dos seus valores de bit individuais (0 ou 1), entre chavetas, na ordem $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Estes bytes são interpretados como elementos do domínio finito usando a representação polinomial:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^7 b_i x^i \quad (3.1)$$

Por exemplo, $\{01100011\}$ identifica o elemento de domínio finito específico $x^6 + x^5 + x + 1$.

É também conveniente a notação de valores de bytes usando a notação hexadecimal em que cada grupo de quatro bits é denotado por um simples character.

Padrão de bits	Caracter	Padrão de bits	Caracter
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Assim o elemento $\{01100011\}$ pode ser representado como $\{63\}$, onde o character que denota o grupo de quatro bits com os bits mais significativos é notado novamente à esquerda.

Algumas operações envolvem um bit adicional (b_8) à esquerda do byte. Onde tal bit extra seja apresentado, será notado como $\{01\}$, precedendo imediatamente o byte; por exemplo, uma sequência de nove bits será apresentada como $\{01\}\{1b\}$.

3.3.3 Vectores de bytes

Os vectores de bytes serão representados na forma:

$$a_0 \ a_1 \ a_2 \ \dots \ a_{15}$$

A ordem dos bytes e dos bits entre os bytes é derivada da sequência dos 128 bits de entrada :

$$in_0 \ in_1 \ in_2 \ \dots \ in_{126} \ in_{127}$$

da seguinte maneira:

$$\begin{aligned} a_0 &= \{in_0, in_1, \dots, in_7\}; \\ a_1 &= \{in_8, in_9, \dots, in_{15}\}; \\ &\vdots \\ a_{15} &= \{in_{120}, in_{121}, \dots, in_{127}\}. \end{aligned}$$

Este padrão pode ser extendido a sequências maiores (i.e., para chaves de 192 e 256 bits), então, e no caso geral,

$$a_n = \{in_{8n}, in_{8n+1}, \dots, in_{8n+7}\} \quad (3.2)$$

3.3.4 O estado

Internamente, as operações do algoritmo AES são executadas num vector bidimensional de bytes, chamado *estado*.

O *estado* consiste em quatro linhas de bytes, cada uma contendo Nb bytes, onde Nb é o tamanho do bloco dividido por 32.

No vector *estado*, denotado pelo símbolo s , cada byte individual tem dois índices, com o seu número de linha r na gama $0 \leq r < 4$ e o seu número de coluna c na gama $0 \leq c < Nb$. Isto permite que um byte individual do vector *estado* seja referido como $S_{r,c}$ ou $S[r, c]$.

Para o algoritmo AES, $Nb = 4$, i.e., $0 \leq c < 4$.

No início da cifra e da cifra inversa, a entrada (o vector de bytes $in_0, in_1, \dots, in_{15}$) é copiada para o vector *estado*, conforme ilustrado na Fig. 3.1.

As operações de cifra ou de cifra inversa são aplicadas neste vector *estado*, a partir do qual é copiada a saída (o vector de bytes $out_0, out_1, \dots, out_{15}$).

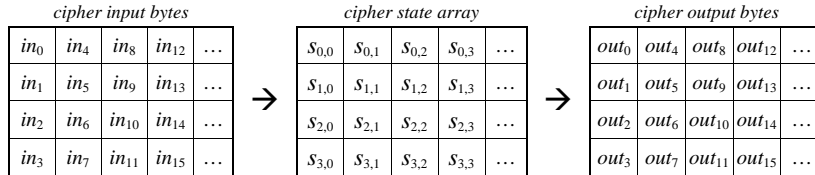


Figura 3.1: Entrada e saída do vector estado

Assim, no início da cifra ou da cifra inversa, o vector de entrada, in , é copiado para o vector de *estado* de acordo com :

$$s[r, c] = in[r + 4c] \quad \text{para } 0 \leq r < 4 \text{ e } 0 \leq c < Nb \quad (3.3)$$

e no fim da cifra ou da cifra inversa, o vector *estado* é copiado para o vector de saída, *out* :

$$out[r + 4c] = s[r, c] \quad \text{para } 0 \leq r < 4 \text{ e } 0 \leq c < Nb \quad (3.4)$$

3.3.5 O *estado* como um vector de colunas

Os quatro bytes de cada coluna do vector *estado* formam palavras de 32 bits. O número da linha r fornece um índice para os quatro bytes dentro de cada palavra.

O *estado* pode assim ser interpretado como um vector unidimensional de palavras de 32 bits (colunas), $w_0 \dots w_3$, onde o número da coluna c fornece um índice para este vector.

Assim, por exemplo na Fig. 3.1, o *estado* pode ser considerado como um vector de quatro palavras, da seguinte maneira:

$$\begin{aligned} w_0 &= s_{0,0} \ s_{1,0} \ s_{2,0} \ s_{3,0} \\ w_1 &= s_{0,1} \ s_{1,1} \ s_{2,1} \ s_{3,1} \\ w_2 &= s_{0,2} \ s_{1,2} \ s_{2,2} \ s_{3,2} \\ w_3 &= s_{0,3} \ s_{1,3} \ s_{2,3} \ s_{3,3} \end{aligned}$$

3.4 Introdução matemática

Todos os bytes no algoritmo AES são interpretados como elementos no domínio finito usando a notação introduzida na Sec. 3.3.2.

Elementos no domínio finito podem ser adicionados ou multiplicados, mas estas operações são diferentes daquelas usadas por números.

As subsecções seguintes introduzem os conceitos matemáticos necessários para a Sec. 3.5 onde é dada a especificação do algoritmo.

3.4.1 Adição

A adição de dois elementos no domínio finito é conseguida através da *adição* dos coeficientes dos polinómios dos dois elementos.

A adição é executada através da operação *XOR* (notada através de \oplus) - i.e. o módulo 2 - $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 1 = 1$ e $0 \oplus 0 = 0$.

Assim a subtração de polinómios é igual à adição de polinómios.

Alternativamente, a adição de elementos no domínio finito pode ser descrita como uma adição de módulo 2, dos bits correspondentes no byte.

Para dois bytes $\{a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0\}$ e $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$, a soma é $\{c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0\}$, onde $c_i = a_i \oplus b_i$.

Por exemplo, as equações seguintes são equivalentes:

$$\begin{aligned} (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) &= x^7 + x^6 + x^4 + x^2 && \text{notação polinomial} \\ \{01010111\} \oplus \{10000011\} &= \{11010100\} && \text{notação binária} \\ \{57\} \oplus \{83\} &= \{d4\} && \text{notação hexadecimal} \end{aligned}$$

3.4.2 Multiplicação

Na representação polinomial, a multiplicação em 2^8 (denominada por \bullet) corresponde à multiplicação dos módulos do polinómio com um polinómio irreduzível de grau 8.

Um polinómio é irreduzível se os seus divisores forem apenas o elemento unitário e ele próprio.

Para o algoritmo AES o polinómio irreduzível é :

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (3.5)$$

ou $\{01\}\{1b\}$ na notação hexadecimal.

Por exemplo, $\{57\} \bullet \{83\} = \{C1\}$ dado que :

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

e

$$\begin{array}{ccc} x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 & & \\ \text{módulo} & & \\ (x^8 + x^4 + x^3 + x + 1) & = & x^7 + x^6 + 1 \end{array}$$

A redução modular através de $m(x)$ garante que o resultado será um polinómio binário de grau menor que oito, e pode então ser representado por um byte.

Ao contrário da adição, não existem operações ao nível do byte que correspondam a esta multiplicação.

A multiplicação conforme definida é associativa, e o elemento $\{01\}$ é a identidade multiplicativa.

Para qualquer polinómio binário diferente de zero $b(x)$ de grau inferior a oito, a inversa da multiplicação de $b(x)$, denominada por $b^{-1}(x)$ pode ser encontrada da seguinte forma:

O algoritmo extendido de *Euclides* é usado para o cálculo de polinómios $a(x)$ e $c(x)$ tais que

$$b(x)a(x) + m(x)c(x) = 1 \quad (3.6)$$

Assim, $a(x) \bullet b(x) \bmod m(x) = 1$, logo

$$b^{-1}(x) = a(x) \bmod m(x) \quad (3.7)$$

Mais ainda, para qualquer $a(x)$, $b(x)$ e $c(x)$ no domínio finito,

$$a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x)$$

Assim, com a adição definida como *XOR* e a multiplicação definida como foi agora apresentada, os 256 valores possíveis para o byte têm uma estrutura no domínio finito 2^8 .

3.4.3 Multiplicação por x

A multiplicação do polinómio definido na Eq. 3.1 com o polinómio x resulta em:

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

O resultado $x \bullet b(x)$ é obtido através da redução do módulo $m(x)$, definido na Eq. 3.5.

Se $b_7 = 0$, o resultado já se encontra na forma reduzida.

Se $b_7 = 1$, a redução é efectuada através da subtracção (i.e, a utilização de *XOR*) do polinomial $m(x)$.

A multiplicação por x (i.e., $\{00000010\}$ ou $\{02\}$) pode ser implementada ao nível do byte através de uma rotação para a esquerda e uma operação ao nível de bit *XOR* com $\{1B\}$ condicionada.

Esta operação nos bytes é denominada de `xtime()`.

A multiplicação por potências superiores a x pode ser implementada através da aplicação repetida de `xtime()`.

Com a adição de resultados intermédios, pode ser conseguida a multiplicação por qualquer constante.

Por exemplo, $\{57\} \bullet \{13\} = \{FE\}$, dado que :

$$\begin{aligned} \{57\} \bullet \{02\} &= \text{xtime}(\{57\}) = \{AE\} \\ \{57\} \bullet \{04\} &= \text{xtime}(\{AE\}) = \{47\} \\ \{57\} \bullet \{08\} &= \text{xtime}(\{47\}) = \{8E\} \\ \{57\} \bullet \{10\} &= \text{xtime}(\{8E\}) = \{07\} \end{aligned}$$

logo,

$$\begin{aligned} \{57\} \bullet \{13\} &= \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\}) \\ &= \{57\} \oplus \{AE\} \oplus \{07\} \\ &= \{FE\} \end{aligned}$$

3.4.4 Polinomiais com coeficientes no domínio finito (2^8)

Polinomiais de quatro termos com coeficientes que são elementos do domínio finito podem ser definidos como:

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

que serão apresentados numa palavra de notação $[a_0, a_1, a_2, a_3]$.

De notar que os polinómios nesta secção comportam-se ligeiramente diferente dos polinómios usados na definição de elementos do domínio finito, embora ambos utilizem a mesma variável x .

Os coeficientes nesta secção são eles mesmos elementos do domínio finito, i.e., bytes, em vez de bits; também a multiplicação de polinómios de quatro termos usa um polinómio de redução diferente, definido em baixo.

A distinção deve ser clara mediante o contexto. Para ilustrar as operações de adição e multiplicação, defina-se

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

como um segundo polinómio.

A adição é executada através da adição dos elementos do domínio finito como potências de x . Esta adição corresponde à operação *XOR* entre os bytes correspondentes a cada palavra.

Assim, usando as as Eq. 3.4.4 e 3.4.4

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

A multiplicação é conseguida em duas etapas. Na primeira etapa, o produto do polinómio $c(x) = a(x) \bullet b(x)$ é expandido algebricamente, tal como potências dando:

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

onde

$$\begin{aligned} c_0 &= a_0 \bullet b_0 & c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 & c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 & c_6 &= a_3 \bullet b_3 \\ c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \end{aligned}$$

O resultado $c(x)$, não representa uma palavra de quatro bytes.

Assim, uma segunda etapa é necessária para reduzir $c(x)$ módulo com um polinomial de grau quatro; o resultado pode assim ser reduzido a um polinomial de grau menor que quatro. Para o algoritmo AES, isto é conseguido com o polinómio $x^4 + 1$, de forma a que :

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4}$$

O produto modular entre $a(x)$ e $b(x)$, denominado por $a(x) \oplus b(x)$ é dado pelo polinomial de quatro termos $d(x)$, definido como :

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

com:

$$\begin{aligned} d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\ d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\ d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\ d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \end{aligned}$$

quando $a(x)$ é um polinómio fixo, a operação definida na Eq. 3.4.4 pode ser escrita na forma matricial como:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (3.8)$$

Dado que $x^4 + x$ não é um polinómio irredutível no domínio 2^8 , a multiplicação por um polinómio de quatro termos não é necessariamente invertível. No entanto, o algoritmo AES especifica um polinómio fixo de quatro termos que tem uma inversa (Ver Sec. 3.5.4 e 3.7.3).

$$\begin{aligned} a(x) &= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \\ a^{-1}(x) &= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \end{aligned}$$

Um outro polinómio, usado no algoritmo AES (ver a função `RotWord()` na Sec. 3.6), que também tem $a_0 = a_1 = a_2 = \{00\}$ e $a_3 = \{01\}$ é o polinómio x^3 . A inspecção da Eq. 3.8 mostra que o seu efeito é formar a palavra de saída através de uma rotação dos bytes na palavra de entrada, i.e., $[b_0, b_1, b_2, b_3]$ é transformado em $[b_1, b_2, b_3, b_0]$.

3.5 Especificação do algoritmo

No algoritmo AES o tamanho dos blocos de entrada, de saída e do *estado* é de 128 bits. Isto é representado por $Nb = 4$, que reflecte o número de palavras de 32 bits (número de colunas) do *estado*.

O tamanho da chave de cifra, K , é de 128, 192 ou 256 bits. O tamanho da chave é representado por $Nk = 4, 6$ ou 8 , que também reflecte o número de palavras de 32 bits (número de colunas) na chave de cifra.

O número de rotações a serem efectuadas durante a execução do algoritmo é dependente do tamanho da chave. O número de rotações é representado por Nr , onde $Nr = 10$ quando $Nk = 4$, $Nr = 12$ quando $Nk = 6$ e $Nr = 14$ quando $Nk = 8$.

As únicas combinações que estão conforme o standard são mostradas na Tab. 3.1.

Tanto para a cifra como para a cifra inversa, o algoritmo AES usa uma função

	Tamanho da Chave Nk palavras	Tamanho dos blocos Nb palavras	Número de Rotações Nr
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabela 3.1: Combinações que respeitam o standard

de rotação ³ que é composta de quatro operações diferentes orientadas ao byte.

1. Substituição de bytes através de uma tabela de substituição (*S-Box*).
2. Deslocamento de linhas no vector *estado* com diferentes desvios.
3. Mistura de dados em cada coluna do vector *estado*.
4. Soma de uma chave de rotação ao *estado*.

3.5.1 A cifra

No início da cifra, a entrada é copiada para o vector *estado* usando as convenções descritas na Sec. 3.3.4.

Depois de uma adição inicial da chave de rotação, o vector *estado* é transformado através da implementação duma função de rotação 10, 12 ou 14 vezes (dependendo do tamanho da chave), com a última rotação sendo ligeiramente

³round function

diferente das primeiras $Nr - 1$ rotações.

O *estado* final é então copiado para a saída como descrito na Sec. 3.3.4.

A função de rotação é parametrizada usando uma chave de escalonamento⁴ que consiste num vector unidimensional de palavras de 4 bytes, derivada da rotina de expansão de chave descrita na Sec. 3.7.

A cifra é descrita pelo pseudo-código dado pela Fig. 7.

As transformações individuais no *estado* (`SubBytes()`, `SfiftRows()`, `MixColumns()`, `AddRoundKey()`) são descritas nas subsecções seguintes.

Na Fig. 7, o vector `w[]` contém a chave de escalonamento, que é descrita na subsecção 3.7.

Como é mostrado na Fig. 7, todas as rotações Nr são idênticas, com a excepção para a última rotação, que não inclui a transformação `MixColumns()`.

Figura 3.2: Pseudo-código da cifra

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[0, Nb-1]) // See Sec. 5.1.4
  for round = 1 step 1 to Nr-1
    SubBytes(state) // See Sec. 5.1.1
    ShiftRows(state) // See Sec. 5.1.2
    MixColumns(state) // See Sec. 5.1.3
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  out = state
end
```

3.5.2 Transformação *SubBytes()*

Esta transformação é uma substituição não linear de bytes que opera independentemente de cada byte do *estado* usando uma tabela de substituição *S-box*.

Esta *S-box* (Fig. ??), que é invertível, é construída através da composição de duas transformações:

1. Calcula a inversão multiplicativa no campo finito $GF(2^8)$, descrita na Sec. 3.4.2; o elemento $\{0, 0\}$ é mapeado nele mesmo.

⁴key schedule

2. Aplica a transformação de afinidade seguinte (em $GF(2)$):

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (3.9)$$

para $0 \leq i < 8$, onde b_i é o índice i do bit no byte, e c_i que é o índice i do byte com valor $\{63\}$ ou $\{01100011\}$. Assim, um número primo numa variável (por exemplo, b') indica que a variável deve ser actualizada com o valor na direita.

Na forma matricial, a transformação de afinidade de um elemento da S -box pode ser expressa como:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (3.10)$$

A Fig. 3.3 ilustra o efeito da transformação **SubBytes()** no Estado.

A S -box usada na transformação *SubBytes()* é apresentada no formato hex-

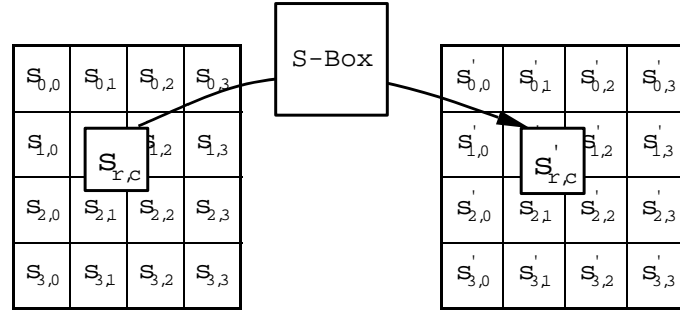


Figura 3.3: A transformação *SubBytes()*

adecimal na Fig. 3.4. Por exemplo, se $s_{1,1} = \{53\}$ então o valor de substituição ($s'_{1,1}$) teria o valor $\{ed\}$.

3.5.3 Transformação *ShiftRows()*

Na transformação *ShiftRows()*, os bytes nas últimas três linhas do *estado* são ciclicamente rodadas com diferentes desvios.

A primeira linha, $r = 0$ não é rodada. A transformação procede da seguinte maneira:

$$S'_{r,c} = S_{r,(c+shift(r,Nb)) \bmod Nb} \quad \text{para } 0 < r < 4 \text{ e } 0 \leq c \leq Nb$$

onde o valor a ser rodado $shift(r, Nb)$ depende no número linha, r , como de seguida (lembrar que $Nb = 4$);

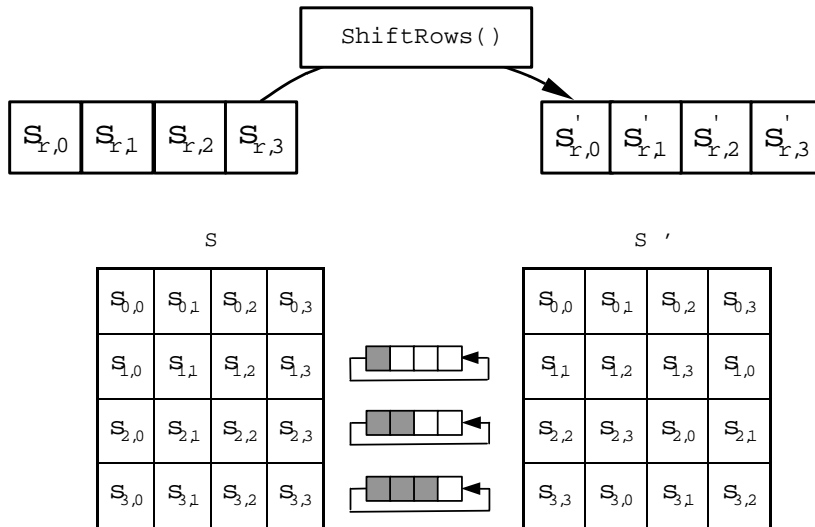
$$shift(1, 4) = 1; shift(2, 4) = 2; shift(3, 4) = 3 \quad (3.11)$$

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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 3.4: Valores de substituição para o byte xy (no formato hexadecimal)

Isto tem o efeito de mover os bytes para um valor inferior na linha (i.e., baixos valores de c numa dada linha), enquanto que os bytes *mais baixos* passam para o topo da linha (i.e, altos valores de c numa dada linha).

A Fig. 3.5 ilustra a transformação $ShiftRows()$.

Figura 3.5: A transformação $ShiftRows()$

3.5.4 Transformação *MixColumns*

A transformação $MixColumns()$ opera no *estado* coluna a coluna, tratando cada coluna como um polinómio de quatro termos, conforme descrito na Sec. 3.4.4.

As colunas são consideradas polinómios em $GF(2^8)$ e são multiplicadas por

um módulo $x^4 + 1$ com um polinómio constante $a(x)$ dado por:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Conforme descrito na Sec. 3.4.4, pode ser visto como uma multiplicação de matrizes. Sendo

$$\begin{aligned} s'(x) &= a(x) \otimes s(x) \\ \begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad \text{para } 0 \leq c \leq Nb \end{aligned}$$

Como resultado desta multiplicação, os quatro bytes numa coluna são substituídos por:

$$\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \oplus s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \end{aligned}$$

A Fig. 3.6 ilustra a transformação *MixColumns()*.

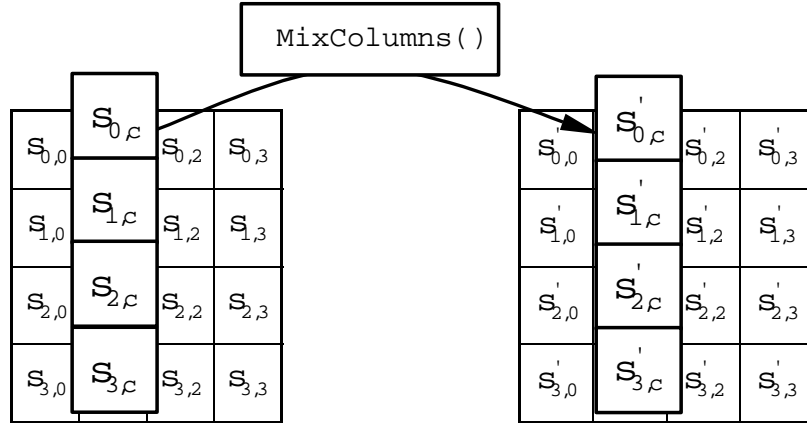


Figura 3.6: A transformação *MixColumns()*

3.5.5 Transformação *AddRoundKey()*

Na transformação *AddRoundKey()*, uma chave de rotação⁵ é adicionada ao *estado* através de uma operação XOR. Cada chave de rotação consiste em Nb palavras da chave de escalonamento (Descrito na Sec. 3.7).

Essas Nb palavras são adicionadas nas colunas do *estado*, tal que,

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round \cdot Nb + c}] \quad \text{para } 0 \leq c < Nb$$

⁵Round Key

onde $[w_i]$ são as chaves de escalonamento descritas na Sec. 3.7, e $round$ é um valor na gama $0 \leq round \leq Nr$.

Na cifra, a adição inicial da chave de rotação ocorre quando $round = 0$, antes da primeira aplicação da função de rotação (ver Fig. 7).

A aplicação da transformação $AddRoundKey()$ às Nr rotações da cifra ocorre quando $1 \leq round \leq Nr$.

A acção desta transformação é ilustrada na Fig. 3.7, onde $l = round * Nb$. O endereçamento de bytes entre as palavras da chave de escalonamento foi descrito na Sec. 3.3.1

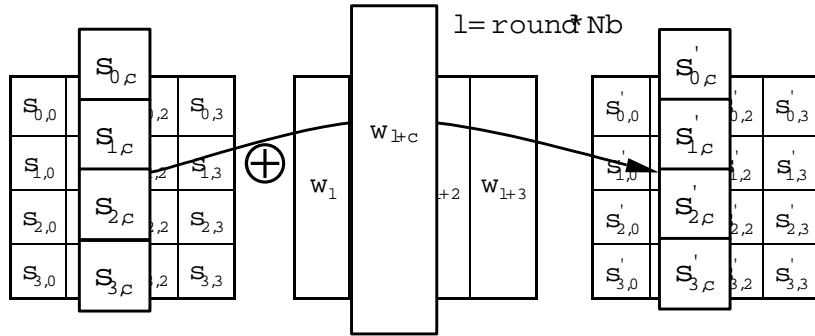


Figura 3.7: Transformação $AddRoundKey()$

3.6 Expansão da Chave

O algoritmo AES pega na chave de cifra, K , e executa uma rotina de expansão de chave para gerar a chave de escalonamento.

A expansão de chave gera um total de $Nb(Nr + 1)$ palavras: o algoritmo requer um conjunto inicial de Nb palavras, e cada uma das Nr rotações requer Nb palavras de dados.

A chave de escalonamento resultante consiste num vector linear de palavras de 4 bytes, notadas como $[w_i]$, estando i na gama $0 \leq i \leq Nb(Nr + 1)$.

A expansão da chave de entrada na chave de escalonamento segue-se segundo o pseudo código apresentado na Fig. ?? $SubWord()$ é uma função que tem como entrada um palavra de 4 bytes, e aplica a $S-box$ (Sec. 3.5.2, Fig. 3.4) a cada um dos quatro bytes de forma a produzir uma palavra de saída.

A função $RotWord()$ tem com entrada uma palavra $[a_0, a_1, a_2, a_3]$, executa uma permutação ciclica, e devolve como saída a palavra $[a_1, a_2, a_3, a_0]$.

O vector constante de rotação, $Rcon[i]$, contém os valores dados por $[x^{j-l}, \{00\}, \{00\}, \{00\}]$, sendo x^{j-l} uma potência de x (x é notado como $\{02\}$) no domínio $GF(2^8)$, conforme discutido na Sec.3.4.2 (de notar que i começa em 1 e não 0).

Através da Fig. 3.6, pode ser notado que as primeiras Nk palavras da chave de expansão são preenchidas com a chave da cifra. Cada palavra seguinte, $w[i]$, é igual à operação XOR com a palavra anterior, $w[i - 1]$, e a palavra Nk posições anteriores, $w[i - Nk]$. Para palavras em posições multiplas de Nk , é aplicada uma transformação a $w[i - 1]$ antes da operação XOR, seguida da operação XOR

com uma constante de rotação, $Rcon[i]$. Esta transformação consiste numa rotação ciclica dos bytes na palavra ($RotWord()$), seguida da aplicação de uma pesquisa na tabela para todos os quatro bytes da palavra ($SubWord()$).

É importante notar que a rotina de expansão da chave para chaves de cifra de 256 bits ($Nk = 8$) é ligeiramente diferente daquelas de 128 e 192 bits. Se $Nk = 8$ e $i - 4$ for um multiplo de Nk então $SubWord()$ é aplicada a $w[i - 1]$ antes da operação XOR.

Figura 3.8: Pseudo-código para a expansão da chave

```

KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
  word temp
  i=0
  while (i<Nk)
    w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i=i+1
  end while
  i=Nk
  while (i < Nb * (Nr+1))
    temp = w[i-1]
    if (i mod Nk = 0)
      temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk = 4)
      temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] xor temp
    i = i + 1
  end while
end

```

3.7 Cifra Inversa

As transformações na cifra na Sec. 3.5.1 podem ser invertidas e implementadas na ordem inversa de forma a produzir uma cifra inversa para o algoritmo AES. As transformações individuais usadas na cifra inversa - $InvShiftRows()$, $InvSubBytes()$, $InvMixColumns()$ e $AddRoundKey()$ processam o Estado e estão descritas nas secções seguintes.

A cifra inversa é descrita no pseudo código da Fig. 3.7. Na Fig. 3.7, o vector $w[]$ contém a chave de escalonamento, que foi descrita previamente na Sec. 3.6

Figura 3.9: Pseudo-código para a cifra inversa

```

InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)
  end for
  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])
  out = state
end

```

3.7.1 Transformação *InvShiftRows()*

InvShiftRows() é a inversa da transformação *ShiftRows()*. Os bytes nas ultimas três linhas do *estado* são ciclicamente rodadas por diferentes números de bytes (offset).

A primeira linha, $r = 0$, não é alterada. As três últimas linhas são ciclicamente rodadas através de $Nb - \text{shift}(r, Nb)$ bytes, onde o valor da rotação $\text{shift}(r, Nb)$ depende do número da linha, e é dado pela Eq. 3.11 (Ver Sec. 3.5.3). A transformação é feita da seguinte maneira:

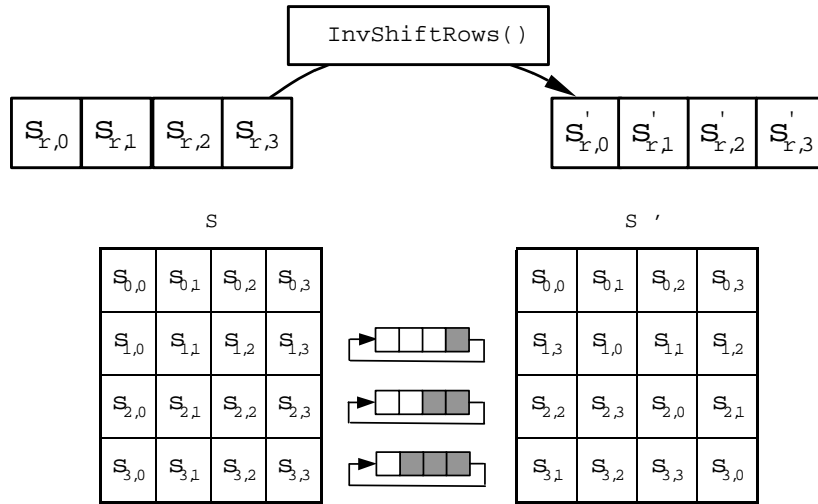
$$S'_{r, (c + \text{shift}(r, Nb)) \bmod Nb} = S_{r, c} \quad \text{para } 0 < r < 4 \text{ e } 0 \leq c \leq Nb$$

A Fig. 3.10 ilustra a transformação *InvShiftRows()*.

3.7.2 Transformação *InvSubBytes()*

InvSubBytes() é a inversa da função de substituição de bytes, na qual é a *S-box* invertida que é aplicada a cada byte do *estado*.

Isto é conseguido através da aplicação da inversa da função de afinidade (Sec. 3.5.1) seguida de uma multiplicação inversa em $GF(2^8)$. A *S-box* inversa utilizada na transformação *InvSubBytes()* é apresentada na Fig. 3.11.

Figura 3.10: A transformação *InvShiftRows()*

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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 3.11: *S-Box* Inversa

3.7.3 Transformação *InvMixColumns()*

InvMixColumns() é a inversa da transformação *MixColumns()*.

A transformação *InvMixColumns()* opera no *estado* coluna a coluna, tratando cada coluna como um polinómio de quatro termos, conforme descrito na Sec. 3.4.4.

As colunas são consideradas polinómios em $GF(2^8)$ e multiplicada de módulo $x^4 + 1$ com um polinómio constante $a^{-1}(x)$ dado por:

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

Conforme descrito na Sec. 3.4.4, pode ser visto como uma multiplicação de matrizes. Sendo

$$\begin{aligned} s'(x) &= a^{-1}(x) \otimes s(x) \\ \begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} &= \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \quad \text{para } 0 \leq c \leq Nb \end{aligned}$$

Como resultado desta multiplicação, os quatro bytes numa coluna são substituídos por:

$$\begin{aligned} s'_{0,c} &= (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s'_{1,c} &= (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c}) \\ s'_{2,c} &= (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c}) \end{aligned}$$

3.7.4 Inversa da transformação *AddRoundKey()*

AddRoundKey(), que foi descrita na Sec. 3.5.5, é a sua própria inversa dado que esta apenas envolve a aplicação da operação *XOR*.

3.7.5 Equivalente da Cifra Inversa

Na cifra inversa apresentada na Sec. 3.7 e na Fig. 3.7, a sequência das transformações difere da sequência da cifra, enquanto a forma das chaves de escalonamento para a encriptação e descriptação se mantêm.

No entanto, várias propriedades no algoritmo AES permitem uma cifra equivalente inversa com a mesma sequência de transformações da cifra (com as transformações substituídas pelas suas inversas).

Isto é conseguido através da alteração da chave de escalonamento.

As duas propriedades que permitem esta cifra equivalente inversa são :

1. As transformações *SubBytes()* e *ShiftRows()* comutam, isto é, uma transformação *SubBytes()* imediatamente seguida por uma transformação *ShiftRows()* é equivalente a uma transformação *ShiftRows()* imediatamente seguida por uma transformação *SubBytes()*. O mesmo é válido para as suas inversas, *InvSubBytes()* e *InvShiftRows()*.

2. As operações de transformação das colunas - *MixColumns()* e *InvMixColumns()* - são lineares no respeitante à entrada da coluna, isto é, $InvMixColumns(state \text{ XOR Round Key}) = InvMixColumns(state) \text{ XOR } InvMixColumns(RoundKey)$

Estas propriedades permitem que a ordem de *InvSubBytes()* e *InvShiftRows()* seja trocada.

A ordem de *AddRoundKey()* e *InvMixColumns()* também pode ser trocada, desde que as colunas (palavras) da chave de descriptação de escalonamento sejam também modificadas através da transformação *InvMixColumns()*.

A cifra inversa equivalente é definida através da troca da ordem de *InvSubBytes()* e *InvShiftRows()* mostrada na Fig. 3.7, e através da comutação da ordem de *AddRoundKey()* e *InvMixColumns()* usadas no ciclo de rotação, depois de primeiro modificar a chave de descriptação de escalonamento para $round = 1$ até $Nr - 1$ usando *InvMixColumns()*.

As primeiras e últimas Nb palavras da chave de descriptação de escalonamento não devem ser modificadas desta maneira.

Dadas estas alterações, a cifra inversa equivalente oferece uma estrutura mais eficiente do que a descrita na Sec. 3.7 e na Fig. 3.7.

O pseudo código para a cifra inversa equivalente é mostrado na Fig. 3.7.5. (O vector de palavras $dw[]$ contém a chave de descriptação de escalonamento.

A modificação à rotina de expansão da chave é também fornecida na Fig. 3.7.5)

Figura 3.12: Pseudo-código para a cifra inversa equivalente

```

EqInvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  for round = Nr-1 step -1 downto 1
    InvSubBytes(state)
    InvShiftRows(state)
    InvMixColumns(state)
    AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
  end for
  InvSubBytes(state)
  InvShiftRows(state)
  AddRoundKey(state, dw[0, Nb-1])
  out = state
end

Para a cifra eq. inversa, adicionar o seguinte
pseudo-código ao fim da rotina de expansão da chave
for i=0 step 1 to (Nr+1)*Nb-1
  dw[i]=w[i]
end for
for round=1 step 1 to Nr-1
  InvMixColumns(dw[Round*Nb, (round+1)*Nb-1])
end for

```

Capítulo 4

Modos de Funcionamento

Um modo de operação, é um algoritmo que fazendo uso de um algoritmo de bloco de chave simétrica fornece serviços tais como a confidencialidade, autenticação, cifragem autenticada, funções de *hashing* e mesmo a geração de bits aleatórios.

Com o aparecimento desta nova cifra de bloco, sentiu-se a necessidade de actualização dos modos de operação utilizados anteriormente no DES e no 3-DES. O NIST está actualmente no processo de especificar modos de operação numa série de publicações especiais. O primeiro documento público foi *Recommendation for Block Cipher Modes of Operation - Methods and Techniques*[22], no qual são descritos cinco modos de confidencialidade que são ligeiramente introduzidos nas secções seguintes.

Espera-se um segundo documento (ainda não disponível na altura em que este projecto se realizou) sobre modos de autenticação actualizados para a cifra AES. Encontra-se no entanto, uma apresentação dos vários modos propostos, não sendo contudo recomendável especular neste relatório quais os que irão ser adoptados ou mesmo recomendados.

Os modos de funcionamento, para confidencialidade, recomendados pelo NIST para o uso de um algoritmo de bloco de chave simétrica (como o AES implementado) são:

1. Electronic Codebook (ECB)
2. Cipher Block Chaining (CBC)
3. Cipher Feedback (CFB)
4. Output Feedback (OFB)
5. Counter (CTR)

Quando usados com um algoritmo de cifra como o AES, estes modos fornecem confidencialidade para diversos tipos de dados.

Neste capítulo pretendo apenas referenciar os modos nos quais a cifra implementada pode ser usada, não pretendendo fornecer uma explicação exaustiva dos respectivos modos.

Para isso, remeto para o documento *Recommendation for Block Cipher Modes of*

Operation - Methods and Techniques[22], do NIST (*National Institute of Standards and Technology*), já editado em Dezembro de 2001, tendo sido actualizado para a cifra agora implementada.

4.1 Electronic Codebook

No modo *ECB*, para uma dada chave, a cada bloco de texto *normal* corresponde um único bloco texto cifrado. Pode-se fazer uma analogia deste método com um *livro de códigos* em que para uma dada chave, existe uma correspondência bilateral entre um dado bloco de texto normal e um bloco de texto cifrado.

Analiticamente, o modo ECB pode ser representado por :

Modo Directo : $C_j = CIPH_k(P_j)$ para $j = 1 \dots n$

Modo Inverso : $P_j = CIPH_k^{-1}(C_j)$ para $j = 1 \dots n$

em que,

j : corresponde ao índice numa sequência de blocos de dados.

C_j : representa o j -ésimo bloco de texto cifrado.

P_j : representa o j -ésimo bloco de texto normal.

$CIPH_k(X)$: corresponde à função de cifra directa, segundo a chave k , aplicada ao bloco X .

$CIPH_k^{-1}(X)$: corresponde à função de cifra inversa, segundo a chave k , aplicada ao bloco X .

Graficamente, o modo ECB encontra-se representado na Fig 4.1

O modo ECB para uma operação directa (passagem de blocos de texto simples para blocos de texto cifrado), corresponde à aplicação da cifra na sua operação directa (aplicada no bloco de texto simples, resultando daí um bloco de texto cifrado).

Dado que cada bloco é tratado separadamente, é possível a computação paralela de várias operações de cifra directa e/ou inversa em simultâneo.

Em aplicações em que não se pretenda que a cada bloco de texto *normal* corresponda um só bloco de texto *cifrado*, este modo não deve ser utilizado.

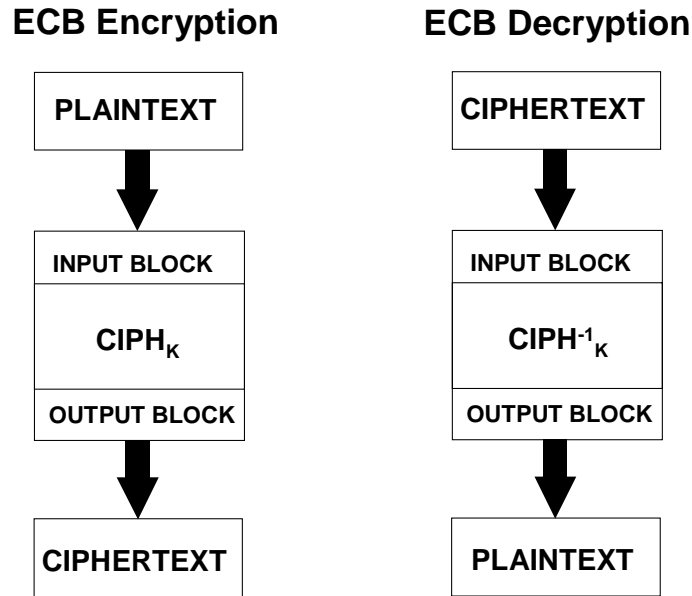
4.2 Cipher Block Chaining

Neste modo, a cifragem de um bloco de dados *encadeia-se* com o último bloco de dados cifrado.

Para a combinação com o primeiro bloco, é então necessário um vector de inicialização. Este vector não necessita de ser secreto, mas sim imprevisível. A geração destes vectores de inicialização é analisada na bibliografia[22]. Analiticamente, este modo pode ser representado por:

Modo Directo : $C_1 = CIPH_k(P_1 \otimes IV)$

$C_j = CIPH_k(P_j \otimes C_{j-1})$ para $j = 2 \dots n$

Figura 4.1: O modo *Electronic Codebook*

Modo Inverso : $P_1 = CIPH_k^{-1}(C_1) \otimes IV$
 $P_j = CIPH_k^{-1}(C_j) \otimes C_{j-1}$ para $j = 2 \dots n$

sendo,

IV : vector de inicialização

\otimes : operação *OU exclusiva*

Este modo encontra-se representado graficamente na Fig. 4.2. Numa cifragem no modo CBC, dado que é necessário o cálculo anterior para se prosseguir com o seguinte, não é possível uma execução paralela de várias operações de cifragem. Na cifra inversa, no entanto, dado que os blocos de entrada para a cifra inversa (i.e., os blocos de dados cifrados) já se encontram disponíveis, é possível uma operação paralela de funções de cifra inversa.

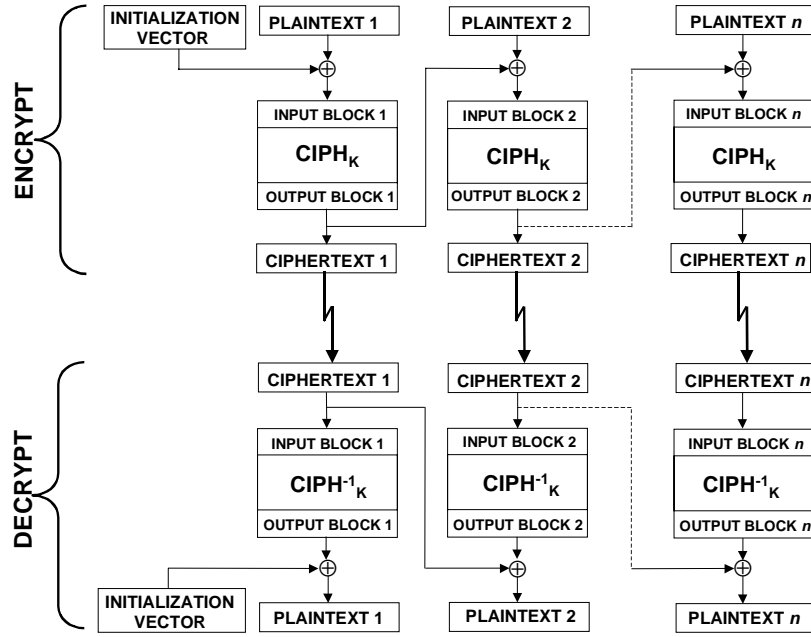
4.3 Cipher Feedback

Neste modo, existe uma realimentação dos sucessivos segmentos de texto cifrado nos blocos de texto simples, que, depois de aplicada a cifra directa, são gerados blocos que por sua vez, e com a operação *OU exclusiva* aos blocos de texto simples, são assim gerados os blocos de texto cifrado (e vice versa).

Analiticamente, é representado por :

Modo Directo:

$$I_1 = IV;$$

Figura 4.2: O modo *Cipher Block Chaining*

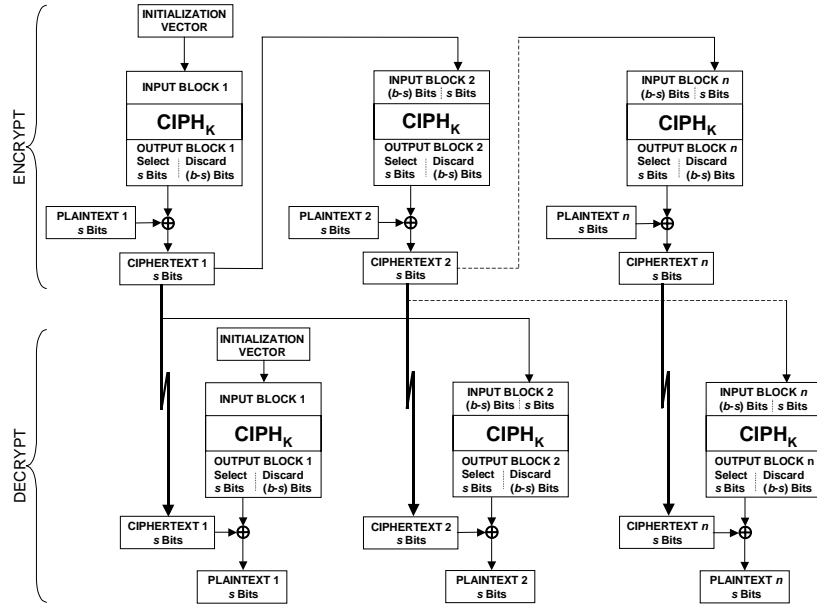
$$\begin{aligned}
 I_j &= LSB_{b-s}(I_{j-1}) | C_{j-1}^\# \quad \text{para } j = 2, \dots, n; \\
 O_j &= CIPH_k(I_j) \quad \text{para } j = 1, 2, \dots, n; \\
 C_j^\# &= P_j^\# \otimes MSB_s(O_j) \quad \text{para } j = 1, 2, \dots, n;
 \end{aligned}$$

Modo Inverso:

$$\begin{aligned}
 I_1 &= IV; \\
 I_j &= LSB_{b-s}(I_{j-1}) | C_{j-1}^\# \quad \text{para } j = 2, \dots, n; \\
 O_j &= CIPH_k(I_j) \quad \text{para } j = 1, 2, \dots, n; \\
 P_j^\# &= C_j^\# \otimes MSB_s(O_j) \quad \text{para } j = 1, 2, \dots, n;
 \end{aligned}$$

É necessário um *inteiro* s , no domínio $1 \leq s \leq b$ (em que b é o tamanho de bloco (128 bits no caso da AES)). Na especificação do modo CFB, cada segmento de texto simples $P_j^\#$ e segmento de texto cifrado $C_j^\#$ consistem em s bits. O valor de s é muitas vezes incorporado no próprio nome do modo, i.e., modo CFB de 1 bit, modo CFB de 8 bits, modo CFB de 64 bits e modo CFB de 128 bits. O modo CFB é representado na Fig. 4.3.

Tal como no modo CBC, a operação directa não pode ser executada em paralelo. A operação inversa pode, desde que primeiro sejam construídos os blocos de entrada (a partir do vector de inicialização e do texto cifrado).

Figura 4.3: O modo *Cipher Feedback*

4.4 Output Feedback Mode

Neste modo é usada a iteração da cifra directa num vector de inicialização para gerar uma sequência de blocos de saída que, por sua vez, são operados com uma função de *OU exclusiva* com o texto simples para produzir o texto cifrado e vice-versa.

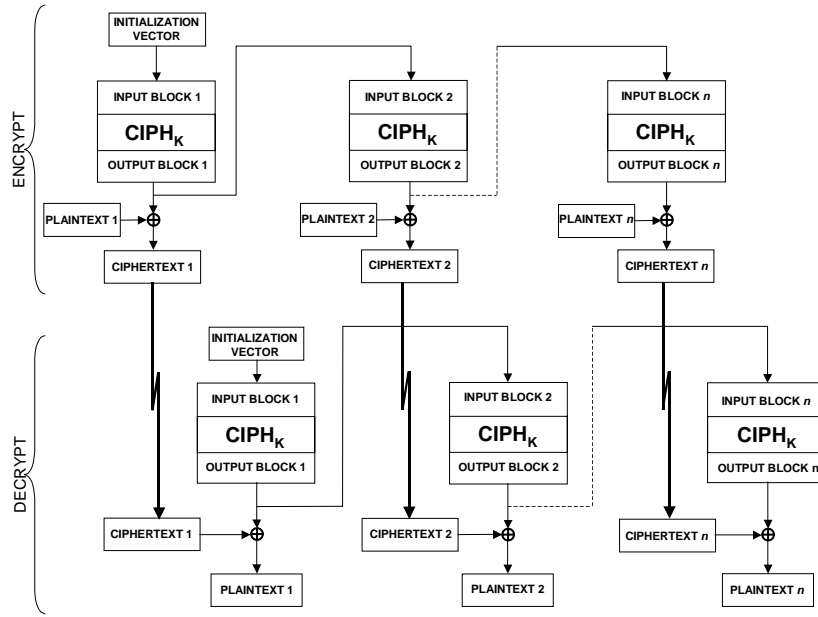
Este modo requer que o vector de inicialização seja único para cada execução do modo numa dada chave.

O modo OFB é definido para o modo directo como :

$$\begin{aligned}
 I_1 &= IV \\
 I_j &= O_{j-1} \quad \text{para } j = 2, \dots, n; \\
 O_j &= CIPH_k(I_j) \quad \text{para } j = 1, 2, \dots, n; \\
 C_j &= P_j \otimes O_j \quad \text{para } j = 1, 2, \dots, n-1; \\
 P_n^\# &= C_n^\# \otimes MSB_u(O_n)
 \end{aligned}$$

Para o modo inverso,

$$\begin{aligned}
 I_1 &= IV \\
 I_j &= O_{j-1} \quad \text{para } j = 2, \dots, n; \\
 O_j &= CIPH_k(I_j) \quad \text{para } j = 1, 2, \dots, n; \\
 P_j &= C_j \otimes O_j \quad \text{para } j = 1, 2, \dots, n-1; \\
 C_n^\# &= P_n^\# \otimes MSB_u(O_n)
 \end{aligned}$$

Figura 4.4: O modo *Output Feedback*

A Fig. 4.4 mostra a operação neste modo.

4.5 Counter

Neste modo, a aplicação da cifra no modo directo a um conjunto de blocos de entrada, chamados de *contadores*, produz uma sequência de blocos de saída, que por sua vez, operados por uma função *OU exclusiva* com os blocos de texto simples, geram o texto cifrado.

A sequência dos contadores deve ter a propriedade de que cada bloco na sequência tem de ser diferente de cada outro bloco. Esta restrição não abrange uma só mensagem, mas sim todas as mensagens em que a chave seja a mesma. Para o modo directo, é definido como :

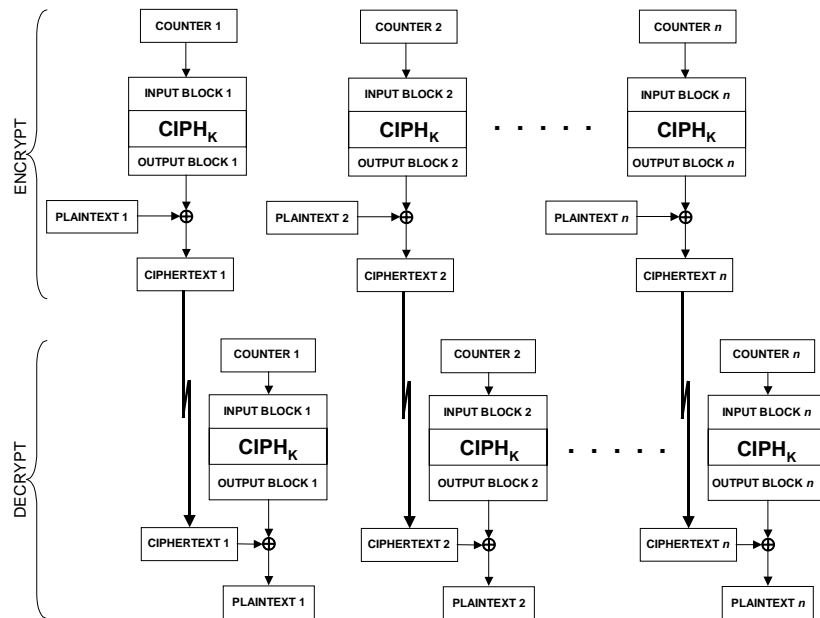
$$\begin{aligned} O_j &= CIPH_k(T_j) & \text{para } j = 1, 2, \dots, n; \\ C_j &= P_j \otimes O_j \\ C_n^\# &= P_n^\# \otimes MSB_u(O_n). \end{aligned}$$

Para o modo inverso,

$$\begin{aligned} O_j &= CIPH_k(T_j) & \text{para } j = 1, 2, \dots, n; \\ P_j &= C_j \otimes O_j \\ P_n^\# &= C_n^\# \otimes MSB_u(O_n). \end{aligned}$$

Neste modo, ambas as funções de cifra e de cifra inversa podem ser executadas em paralelo.

Este modo encontra-se representado na Fig. 4.5.

Figura 4.5: O modo *Counter*

Capítulo 5

Implementação

A implementação prática consistiu no desenvolvimento de software que possibilitasse a operação da cifra e da cifra inversa num microcontrolador de 8 bits da família *i51*.

A linguagem de programação utilizada foi *C* e *Assembly* e foi utilizado um sistema de controlo de versões CVS (Ver anexo A).

Em seguida referem-se os aspectos a ter em conta na implementação, os pormenores da implementação no microcontrolador em questão e uma comparação com outras implementações efectuadas.

5.1 Aspectos a ter em conta

Num microcontrolador de 8 bits, como o utilizado, a cifra e a cifra inversa podem ser implementadas através do desenvolvimento das sucessivas transformações que as compreendem. São então consideradas como rotinas independentes cada operação da cifra.

As operações de rotação de linhas (*ShiftRow*) e adição da chave de cada ronda (*RoundKey*) são quase directas. A rotina *ByteSub* necessita de uma tabela de 256 bytes.

Nestas 3 rotinas é conveniente uma operação serial, byte a byte, em cada elemento do *estado*. Através de código que explicitamente defina e opere em cada byte, é assim minimizado o gasto relativo à indexação de cada byte do *estado*.

A transformação *MixColumns* requer uma multiplicação de matrizes no domínio GF (2^8), cuja implementação pode ser efectuada de uma maneira eficiente para cada coluna do estado.

A linguagem recomendada para este tipo de operações é claramente o *Assembly*, de forma a optimizar a rapidez de execução e para um controlo preciso da simetria do tempo de execução. Optei pela linguagem *c* quer para reduzir a complexidade, quer para aumentar a legibilidade e facilitar uma possível aplicação numa dada aplicação.

5.1.1 Ataques de *temporização*

Com um algoritmo cujo tempo de execução seja independente do conteúdo do bloco de entrada e da chave, não é possível através da análise precisa do tempo de execução ficar com uma ideia do conteúdo da chave ou do bloco de entrada. Uma das técnicas para evitar este tipo de ataques é a utilização de tabelas, tal como foi feito na implementação da rotina *ByteSub*.

Na maioria dos compiladores de *c*, a utilização de tabelas, leva a um código que é executado sempre com o mesmo tempo, sendo assim evitado este tipo de ataques. No entanto, a memória ocupada para o alojamento das tabelas é uma desvantagem que é necessário ponderar.

5.1.2 Paralelismo

Analizado o procedimento da cifra e da cifra inversa, pode-se observar que existe um certo paralelismo em cada operação nos bytes, nas linhas ou colunas do *estado*, para cada operação de rotação. Caso se pretenda uma implementação extremamente eficiente, é aconselhável a exploração deste paralelismo. Também as operações de procura nas tabelas podem ser feitas em paralelo. A maioria das operações *EXOR* podem também ser feitas em paralelo.

A expansão da chave é claramente de uma natureza sequencial (o valor de $w[i]$ é necessário para o cálculo de $w[i + 1]$). Para a maioria das aplicações (em que é utilizada a mesma chave para diversas operações de cifra e de cifra inversa), a rotina de expansão da chave é feita apenas uma vez, não sendo assim possível implementar qualquer tipo de paralelismo.

No entanto, para aplicações onde a velocidade de execução é crítica, e em que sejam utilizadas várias chaves (no extremo, uma diferente a cada operação de cifra ou de cifra inversa), é recomendável a execução *simultânea* da operação de cifra com a operação de expansão de chave da próxima cifra.

Dada a plataforma para a qual se destinava a implementação (não permite execuções *simultâneas*), esta característica não pode ser aproveitada, sendo no entanto, recomendável para implementações completamente em hardware (*FP-GAs* por exemplo).

5.1.3 Implementação da cifra inversa

Para a escolha do polinomial para a operação *MixColumns* e para a operação de expansão da chave foi claramente tida em conta a performance da cifra. A cifra inversa tem uma estrutura idêntica, mas usa um polinómio diferente para a operação *MixColumns* e (em alguns casos) uma rotina de expansão da chave diferente. É notada uma degradação de performance na execução da cifra inversa no caso de execução em microprocessadores de 8 bits.

Esta assimetria é devida ao facto de que a performance da cifra é inversa é tida como menos importante em relação à operação de cifra. Em diversas aplicações como o cálculo de MACs (*Message Authentication Code*), no modo CFB (*Cipher Feedback Mode* - Ver Sec. ??) e no modo OFB (*Output Feedback Mode* - Ver Sec. ??), a operação de cifra inversa nem sequer é utilizada.

Esta assimetria de performances de execução, provém do facto de que na operação de cifra directa, os coeficientes na operação *MixColumns*, são limitados a

01, 02 e 03, que corresponde a uma execução eficiente através da utilização de *xtime()*. Para a operação de cifra inversa (*InvMixColumns*), são usados coeficientes 09, 0E, 0B e 0D. Para uma utilização de *xtime()* estes cálculos demoram um tempo acrescido considerável.

A velocidade de execução, quando se pretende utilizar a operação de cifra inversa, pode ser acelerada, com a utilização de tabelas à custa de uma degradação do espaço de memória ocupado.

A rotina de expansão da chave que gera W está definida de forma a que seja possível começar com as últimas Nk palavras da chave de rotação e calcular a chave de cifra original. O cálculo, em tempo real, das chaves de rotação começando pela chave de cifra inversa é assim possível.

5.2 A Multiplicação

A multiplicação na cifra AES é uma operação extremamente usada (Ver Fig. 5.1, pelo que a sua implementação é essencial para a performance da cifra. São assim demonstradas duas alternativas eficientes para o seu cálculo.

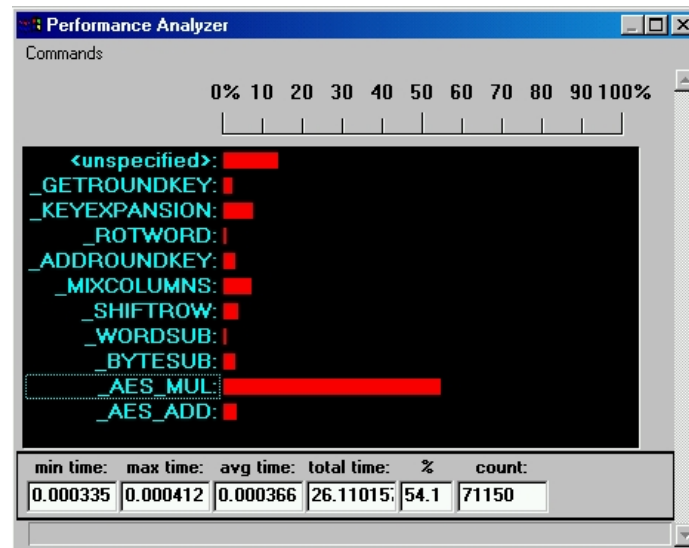


Figura 5.1: A importância da operação de multiplicação na cifra

5.2.1 No domínio finito 2^8

A multiplicação, conforme foi explicada na Sec.3.4.2, consiste na adição dos índices, seguida da redução através do módulo $m(x)$. A multiplicação de $\{57\} \bullet \{83\}$ (ou $(x^6 + x^4 + x^2 + x + 1) \bullet (x^7 + x + 1)$) é então calculada da seguinte maneira:

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

$$(x^6 + x^4 + x^2 + x + 1) \bullet 1 = x^6 + x^4 + x^2 + x + 1$$

A soma destes resultados parciais é:

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

Uma divisão pelo polinómio irreduzível $m(x)$ ($x^8 + x^4 + x^3 + x + 1$) consiste em:

$$(x^8 + x^4 + x^3 + x + 1) \bullet x^5 = x^{13} + x^9 + x^8 + x^6 + x^5$$

O x^5 provém de $x^{13} - x^8$.

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

Como o resultado parcial ainda não se encontra no domínio finito 2^8 repete-se o procedimento:

$$(x^8 + x^4 + x^3 + x + 1) \bullet x^3 = x^{11} + x^7 + x^6 + x^4 + x^3$$

O x^3 provém de $x^{11} - x^8$.

$$(x^{11} + x^4 + x^3 + 1) - (x^{11} + x^7 + x^6 + x^4 + x^3) = x^7 + x^6 + 1$$

que já se encontra no domínio finito 2^8 , e que corresponde a $\{C1\}$.

5.2.2 Através de *shift*

O elemento de domínio finito $\{00000010\}$ é o polinómio x , o que significa que a sua multiplicação por outro elemento aumenta os índices de polinómio do outro elemento por um. Isto é o equivalente a uma rotação para a esquerda por um bit, na sua representação em bytes, de forma a que o bit na posição i passe para a posição $i + 1$.

Caso o bit mais significativo já esteja a 1 ocorre um excesso e é criado um termo em x^8 , no qual o polinómio modular é adicionado de forma a cancelar o bit adicional, deixando um resultado que cabe em apenas 1 byte.

Por exemplo, a multiplicação de $\{11001000\}$ por x , que é $\{00000010\}$, resulta inicialmente em $1\{10010000\}$. O bit em excesso é removido através da adição de $1\{00011011\}$, o polinómio modular, usando a operação *EXOR* o que resulta em $\{10001011\}$.

Repetindo o processo, um elemento no domínio finito pode ser multiplicado por todas as potências de x desde 0 até 7. A multiplicação deste elemento por outro elemento no domínio finito pode então ser conseguida através da adição dos resultados para os índices apropriados de x .

Por exemplo, a Tab. 5.1 exemplifica os passos para o cálculo do produto entre $\{57\}$ e $\{83\}$.

p	$\{57\} \bullet x^p$	$\oplus m(x)$	$\{57\} \bullet x^p$	$\{83\}$	\oplus ao resultado	resultado
0	{01010111}		{01010111}	1	{01010111}	{01010111}
1	{10101110}		{10101110}	1	{10101110}	{11111001}
2	1{01011100}	1{00011011}	{01000111}	0		
3	{10001110}		{10001110}	0		
4	1{00011100}	1{00011011}	{00000111}	0		
5	{00001110}		{00001110}	0		
6	{00011100}		{00011100}	0		
7	{00111000}		{00111000}	1	{00111000}	{11000001}

Tabela 5.1: Multiplicação através da rotação

5.3 Pormenores da implementação no microcontrolador em questão

Para a implementação no microcontrolador da família i51 optei apenas por uma chave de tamanho de 128 bits (16 bytes), não sendo possível utilizar directamente a cifra efectuada para as extensões de 24 ou 32 bytes. Assim, não foi diminuída a dificuldade do código, mas apenas o número de condições *if-then-else*. É relativamente fácil a conversão da cifra efectuada (128 bits) para as diversas extensões, graças a um código sempre que possível parametrizado. Na implementação tive o especial cuidado de seguir sempre o standard[1], pelo que a implementação efectuada esteja conforme o standard.

5.3.1 Especificação da cifra

Dada a possibilidade de interligar a operação de cifragem com outras aplicações, o código de cifragem é o menos intrusivo possível. A sua especificação para o programa principal compreende a definição de algumas variáveis globais necessárias à maioria das subrotinas para as quais se tornava *caro* a sua passagem no argumento da função. São assim :

```
unsigned char xdata initkey[0x10];
unsigned char xdata in[0x10];
unsigned char data * pout;
```

Devido à multiplicidade de aplicações nas quais a cifra se pode inserir, optei pela inicialização da chave de cifra no aplicação principal. Cabe assim à aplicação definir os valores da chave da cifra.

A chamada à função de cifra directa e inversa é feita através de :

```
unsigned char * cifra(unsigned char *, unsigned char *,
                    bit);
```

```
pout=cifra(in,initkey,0); //Cifra Directa!
```

em que :

pout é um apontador para o vector com tamanho de 128 bits que contém a saída da cifra (inversa).

in é um apontador para o vector com tamanho de 128 bits que contém a entrada da cifra (inversa).

initkey é um apontador para o vector com tamanho de 128 bits que contém a chave de cifra (inversa).

0 é um bit que quando é 0 corresponde à operação da cifra directa e quando é 1 corresponde à operação de cifra inversa.

Como se pode observar, torna-se extremamente simples a inclusão da função de cifra directa/inversa numa dada aplicação.

5.3.2 A rotina *cifra*

Esta rotina é encarregue de executar a função de cifra ou de cifra inversa (mediante o terceiro parâmetro na sua chamada), chamando para isso as diversas subrotinas que compreendem a sua operação.

Inicializa as variáveis essenciais ao funcionamento da cifra, como o *estado*, o vector que alberga a chave expandida e a chave de rotação.

Inicialmente é chamada a rotina de expansão da cifra, de forma a que, finda esta rotina, seja possível facilmente meter num ciclo as diversas rotações que a cifra tem de fazer.

De seguida é verificado se é pretendida uma operação de cifra ou de cifra inversa, executando assim, código diferente conforme essa opção.

```
unsigned char * cifra(unsigned char *pin,
                      unsigned char *pinitkey,
                      bit reverse){
    unsigned char data state[0x10];
    unsigned char pdata keyexpanded[0xB0];
    unsigned char data roundkey[0x10];
    unsigned char data i;
    for (i=0;i<=0x0F;i++){
        state[i]=*pin;
        pin++;
    }
    keyexpansion(pinitkey,keyexpanded);
    if (reverse){
        \\ printf{Executando a cifra inversa}
        getroundkey(roundkey,keyexpanded,0x0A);
```

```

    addroundkey(state,roundkey);
    for (i=0x09;i>0;i--){
        shiftrow(state,1);
        bytesub(state,1);
        getroundkey(roundkey,keyexpanded,i);
        addroundkey(state,roundkey);
        mixcolumns(state,1);
    }
    shiftrow(state,1);
    bytesub(state,1);
    getroundkey(roundkey,keyexpanded,0);
    addroundkey(state,roundkey);
}else{
    \\ printf{Executando a cifra directa}
    getroundkey(roundkey,keyexpanded,0);
    addroundkey(state,roundkey);
    for (i=1;i<0x0A;i++){
        bytesub(state,0);
        shiftrow(state,0);
        mixcolumns(state,0);
        getroundkey(roundkey,keyexpanded,i);
        addroundkey(state,roundkey);
    }
    bytesub(state,0);
    shiftrow(state,0);
    getroundkey(roundkey,keyexpanded,0x0A);
    addroundkey(state,roundkey);
}
return{&state}
}

```

5.3.3 GetRoundKey

De forma a facilitar a parametrização do código, optei por incluir uma rotina adicional que retorna um apontador para a chave de rotação do vector de chaves de rotação. Ficou assim facilitada toda a percepção do código.

5.3.4 Considerações quanto ao resto das rotinas

De forma a evitar estender o tamanho deste relatório, com informação duplicada optei por não proceder à descrição exaustiva de cada rotina.

As operações das rotinas restantes seguem as operações já descritas no Cap. 3, sendo o código autodescritivo e encontram-se relativamente bem comentadas. Sempre que me foi possível, deixei o código que usei para a depuração comentado, pelo que considero facilmente perceptível a operação de cada rotina.

5.4 Validação da implementação efectuada

Para a confirmação da implementação da cifra, estão disponíveis vários vectores de teste, que mostram os valores de entrada e de saída, passo a passo. Com estes vectores é assim possível a verificação de uma cifra implementada.

Depois de algum código para amostrar os valores intermédios, confirmei que realmente a cifra correspondia à especificação. A comparação dos vectores de teste, com os valores de saída da cifra implementada encontra-se amostrada na Fig. 5.2.

Como esperado, observa-se uma simetria de todos os resultados intermédios e finais, implicando assim, uma correcta implementação da cifra.

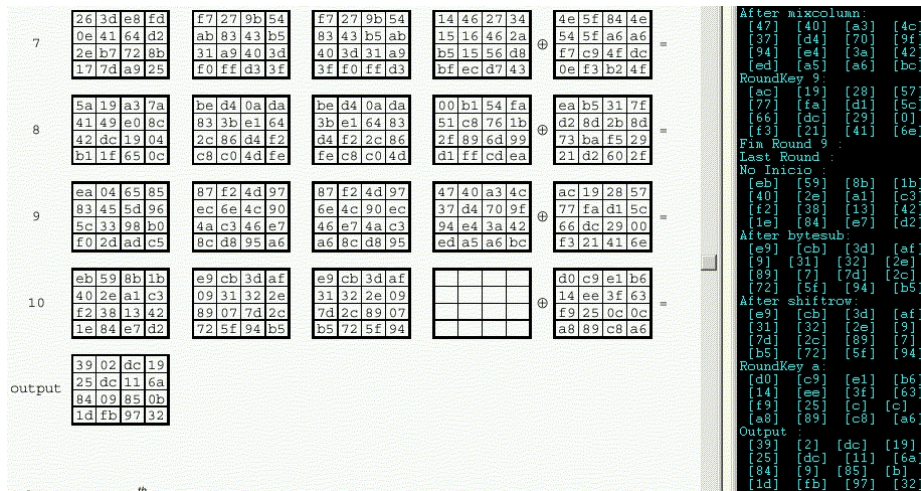


Figura 5.2: Comparação do resultado da cifra com os valores de teste

5.5 Comparação com outras implementações

As implementações existentes, neste tipo de microcontrolador, tiveram como objectivo demonstrar a eficiência da cifra em microcontroladores de 8 bits.

Na cifra implementada, optei por, dar predominância à legibilidade do código e à sua sequencialidade de forma a que seja relativamente fácil a interligação da função de cifra e de cifra inversa numa outra qualquer aplicação.

O tamanho da implementação efectuada encontra-se expresso na Tab. 5.2
A velocidade de execução encontra-se na Fig. 5.3 A velocidade de execução e o seu tamanho da implementação feita para o concurso da cifra AES encontram-se expressos na Tab. 5.3.

Rotina	Hexadecimal	Decimal (bytes)
MixColumns	0x04A9	1193
KeyExpansion	0x022A	554
Cifra	0x01BC	444
ShiftRow	0x00D9	217
WordSub	0x0099	153
RotWord	0x0074	116
GetRoundKey	0x0071	113
ByteSub	0x006D	109
AddRoundKey	0x0066	102
AESMul	0x0060	96
AESAdd	0x0004	4
Total :	0x0C1D	3101

Tabela 5.2: Tamanho do Código Implementado

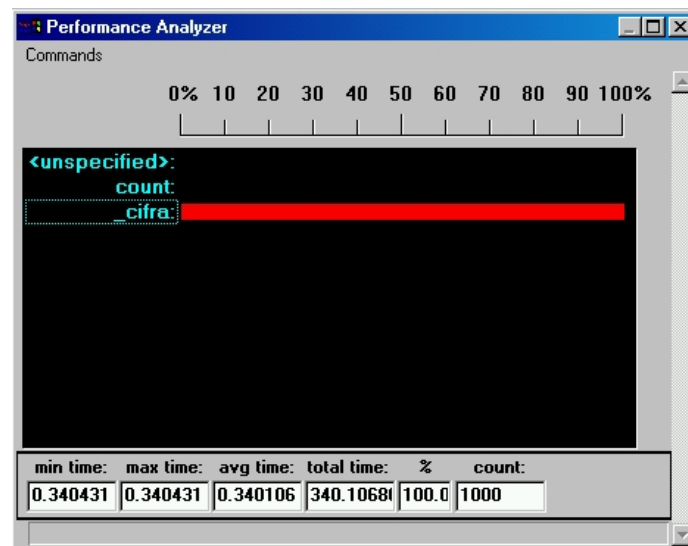


Figura 5.3: Tempo de execução do Código Implementado

Facilmente se observa a assimetria de resultados.
Na implementação pelos criadores da cifra, para um tamanho de código de 1016 bytes a execução é de 3168 ciclos.
Na implementação agora criada, para um tamanho de código de 3101 bytes a

Tamanho da Chave/Bloco	Numero de Ciclos	Tamanho do Código
128/128(a)	4065	768
128/128(b)	3744	826
128/128(c)	3168	1016
192/128	4512	1125
256/128	5221	1041

Tabela 5.3: Resultados da implementação usada para o concurso AES

execução é de 340431 ciclos.

Varios factores levam a estes resultados:

Objectivos : Enquanto que na cifra agora implementada, o objectivo principal era o estudo da cifra, e a um nível académico mostrar como era implementada e quais os passos a seguir, na implementação efectuada pelos próprios criadores da cifra, o objectivo era mostrar a performance numa implementação. Claramente seria muito difícil, senão mesmo impossível, conseguir uma implementação melhor que a que foi efectuada pelos criadores da cifra.

Assembly vs C : Tal como a implementação *de referência*, utilizei a linguagem C. No entanto, a cifra com melhor performance, levou um *fine tuning* manual para uma completa optimização. Na implementação que efectuei, não seria útil para este projecto e para a minha valorização de conhecimentos, a completa revisão do código gerado, para identificar e optimizar todo o código.

Escolhas de Implementação : Em quase todas as funções da cifra, segui fielmente o standard, pois essa era a melhor maneira de perceber claramente o funcionamento da cifra. Tal como é referenciado no standard - "*Given the same input key and data (plaintext or cyphertext), any implementation that produces the same output (plaintext or cyphertext) as the algorithm specifies in this standard is an acceptable implementation of the AES*" - não tem necessariamente de seguir todos os passos especificados, para obter os mesmos resultados, i.e., existem várias hipóteses de implementação. É provável que na implementação pelos criadores, tenham sido utilizados pormenores de implementação referentes à cifra que acelerem a sua execução neste tipo de microcontroladores.

Pormenores Específicos : Como foi mostrado na Sec. 5.2, existem várias secções da cifra que influenciam claramente a sua performance, nomeadamente a multiplicação e também a rotina de expansão da chave. Para aplicações em que a performance seja essencial, é imperativo que se proceda à optimização completa destas secções.

Code size vs. Execution Speed : De notar que são apresentadas várias combinações de implementação ((a), (b), (c) na Tab. 5.3) para o mesmo tamanho da chave e de bloco (128 bits). Isto deve-se ao facto de que uma implementação em que o tamanho de código seja essencial, necessariamente, vai ser executada com menos performance. A utilização (ou não)

de tabelas por exemplo, é um destes factores. Por exemplo, a multiplicação pode ser gerada muito rapidamente caso sejam utilizadas tabelas, no entanto, o espaço de código (ou de dados, caso seja essa a opção) irá ser aumentado. É neste compromisso, que também cabe ao integrador escolher a melhor solução.

Como era esperado, a implementação agora efectuada, não teria a performance e a optimização de código que foi conseguida pelos criadores da cifra. É útil, no entanto, para a percepção dos vários passos que a cifra tem de executar, e para uma interligação fácil com outras aplicações.

Capítulo 6

Ensaio

Para chegar à aplicação final, que foi mostrada na apresentação final, foram vários os passos efectuados.

6.1 Descrição da aplicação

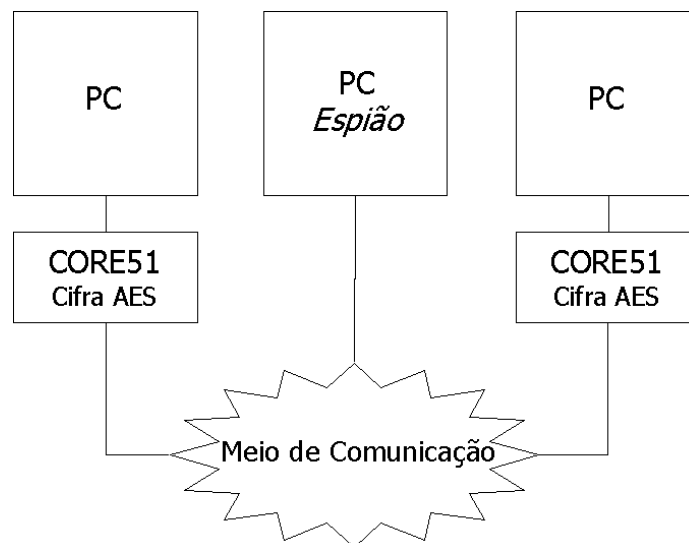


Figura 6.1: Esquema da apresentação efectuada

Para a aplicação *final* optei por demonstrar as operações de cifra directa e inversa, numa comunicação serial entre 2 PCs. Um terceiro PC analisa os dados seriais que se encontram no meio de comunicação (Ver Fig. 6.1). Estabelece-se assim um canal seguro de comunicação serial entre 2 PCs.

6.2 Descrição do hardware

6.2.1 Placa utilizada

O hardware utilizado foi a placa de desenvolvimento *CORE51* disponível para este tipo de aplicações. Esta placa integra, entre outros componentes, um microcontrolador 8031 com um cristal de 11.0952Mhz, um interface serial e um *bootloader* na eeprom que além de possibilitar o carregamento e execução de ficheiros no formato hexadecimal Intel, integra-se transparentemente no depurador *DScope* da *Keil*. É assim, um sistema ideal para o desenvolvimento de sistemas deste tipo.

6.2.2 Pormenores de hardware

Dado que o microcontrolador utilizado tem apenas 1 porta serial UART, foi necessário o desenvolvimento de uma segunda porta serial por software. Para isso, e como está previsto na placa de desenvolvimento, são utilizados 2 pinos do porto 1 (P1.2 como RXD e P1.3 como TXD) do microcontrolador. No entanto, para evitar uma amostragem contínua do estado do porto P1.2 de forma a analisar a recepção de dados, optei por interligar o porto P1.2 à entrada de interrupção externa INT0. Isso foi feito através de um condutor ligado na própria placa.

Para a aplicação final, existe então um canal serial de comunicação entre as 2 placas CORE51 que foi implementado através da própria UART do microcontrolador. A comunicação é feita no modo 8,N,1 sem controlo de fluxo, e com uma taxa de transmissão de 57600bps. A segunda comunicação serial é emulada por software, e corresponde à ligação com o PC. Esta comunicação é feita no modo 8,N,1, também sem controlo de fluxo e com uma taxa de transmissão de 9600bps.

Finalmente bastou configurar os *jumpers* da placa para um correcto funcionamento para esta aplicação, i.e, *JP10* e *JP11* activados e *JP12* desactivo, de forma a possibilitar a segunda porta de comunicação serial. Todos os restantes *jumpers* são mantidos na sua configuração por omissão.

6.3 Pormenores do software

A nível da aplicação final, optei por uma comunicação *full-duplex* entre as placas, de forma a possibilitar a visualização da operação da cifra inversa no terceiro PC. Para esta implementação, foi então necessário coordenar as diversas funções do programa.

Continuamente o microcontrolador faz a amostragem da possibilidade de caracteres recebidos pelo PC, e caso existam, são guardados num *buffer*. Quando esse *buffer* contém 16 caracteres, são cifrados e enviados. Na interrupção da porta serial, na recepção, é feita a operação inversa. A rotina de cifra/cifra inversa não é resistente a execuções simultâneas, pelo que foi necessário proteger até certo nível a sua execução. Assim, e como quer a recepção de dados cifrados pela outra CORE51, quer um novo carácter por parte do PC são importantes, optei

por não desactivar nenhuma função. Dado que a cifra é um processo mais lento que quer a comunicação serial quer por hardware quer por software, apenas com detecção e reenvio de dados seria possível limitar um possível *overflow* de informação recebida por parte do microcontrolador. Por outro lado, por hardware poderia ser implementado um controlo de fluxo do tipo *XON/XOFF* ou mesmo com os sinais de controlo da porta serial, no entanto, isso exigiria extensões à placa de desenvolvimento, que embora relativamente fáceis, ocupariam alguns recursos.

Dado que o objectivo principal deste projecto é a cifra em si, e não uma possível aplicação, acabei por menosprezar este aspecto, limitando a taxa de emissão de caracteres e de linhas no próprio programa de interface serial no PC.

Dada a rígida temporização que tinha de ser efectuada para a geração das taxas de transmissão, as rotinas de envio e recepção de caracteres com o PC foram feitas em linguagem *Assembly*. Embora, fosse recomendado ao nível estrutural a utilização de um temporizador para esta geração, a utilização de ciclos de espera entre cada bit da comunicação revelou-se menos problemática, e até mais fiável. Para a comunicação entre as placas CORE51 uma *nuance* do compilador *Keil* revelava um problema na comunicação ou na cifra em alguns dados de entrada, quando realmente não havia problemas. A rotina *standard putchar* converte os caracteres de *linefeed* (`\n`) em *linefeed* seguido de *carriage return* (`\n\r`). Embora tal procedimento seja recomendado na maioria das aplicações, a comunicação cifrada não é o caso. Quando o resultado da cifra continha o carácter `\n`, a operação inversa da cifra, retornava resultados diferentes aos que eram especificados em *plain text* na outra placa de desenvolvimento.

A solução passou pela reescrita desta função, de forma a que não fosse feita nenhuma conversão.

6.4 Ensaios efectuados

Dada a dificuldade de análise do correcto funcionamento quer da operação de cifra, quer de cifra inversa, o código de depuração, que mostra a cada instante os valores de entrada e de saída, revelou-se essencial.

Foram vários os ensaios efectuados, nomeadamente para a análise da cifra, e finalmente para a demonstração da aplicação. Para a análise da cifra, foram inúmeros os ensaios efectuados. A cada nova rotina, era conveniente verificar o seu correcto funcionamento, seguindo-se em seguida, uma comparação manual com os vectores de teste da implementação padrão.

Para a necessária análise de desempenho das possíveis implementações específicas de cada rotina, utilizei o módulo *Performance Analyzer* do depurador *DScope* da *Keil*. Foi-me assim possível analisar alguns pormenores de implementação para mais facilmente poder optar entre métodos de implementação.

Foram também necessários alguns ensaios para uma análise da cifra a nível de performance, para poder ser feita uma comparação com outras implementações efectuadas.

A diversidade de ensaios revelou-se essencial para detectar e corrigir alguns *bugs* que só eram detectados quando realmente era utilizado um ensaio físico. Foi também possível ter noção prática do tempo de cifra e cifra inversa, e ter percepção da possibilidade de *overflow* de informação no microcontrolador.

Capítulo 7

Conclusão

Este projecto foi-me especialmente útil pelos conhecimentos que obti na sua execução.

A nível da cifra AES, não faço, nem esperava poder fazer, comentários técnicos no quanto é segura a cifra AES. Existem concerteza entidades mais especializadas que eu, para tecer comentários acerca *do quanto* é difícil quebrar o código. Para uma chave no domínio de 2^{128} , e como, segundo especialistas, não existem atalhos para a encontrar, o método de uma procura exaustiva parece ser o único disponível. Com este método, e com o poder de processamento actual mesmo tendo em conta a evolução exponencial da tecnologia, demorará vários anos até que seja possível quebrar uma só chave. Como é usual em processos de criptografia, a parte mais vulnerável da cifra directa/inversa será mesmo o processo de criar, transferir e guardar a chave secreta.

O estudo da cifra AES remeteu-me para o estudo dos diversos métodos, conceitos e técnicas de cifragem existentes. Esse foi um aspecto bastante enriquecedor na execução deste projecto.

A implementação em si revelou-se bastante interessante dada a sua relativa dificuldade e a tolerância nula a qualquer tipo de erros. A implementação de uma aplicação prática, foi essencial não só para a demonstração da cifra em funcionamento, mas também para um conhecimento aprofundado dos requisitos de hardware.

A utilização continua de ferramentas como o μ Vision da Keil, a utilização de CVS e até mesmo de L^AT_EX, possibilitou um aprofundamento de conhecimentos nesta matéria.

Considero este relatório e os conhecimentos que obti na sua elaboração úteis quer para uma futura implementação da cifra, tendo já como ponto de partida uma cifra existente e os comentários acerca das decisões tomadas e os diversos pormenores de implementação, quer mesmo para futuros projectos no assunto de criptografia em geral.

Como resultados deste projecto, além das vantagens que obti, apresento uma implementação da cifra que facilmente pode ser integrada em aplicações futuras.

Apêndice A

CVS

A.1 Introdução

Para a realização deste projecto foi essencial a utilização de um software de controlo de versões. Permitiu a cada instante guardar as várias versões quer da implementação quer mesmo deste relatório e confirmar as alterações efectuadas. Com este apêndice pretendo mostrar o que é o CVS e incluo também um pequeno guia quer para o utilizador quer para o administrador de sistemas que instala e configura um servidor para CVS. O módulo para interface web é essencial para de modo extremamente fácil e intuitivo se poder consultar as variadas versões, as suas alterações e parâmetros temporais. Para este projecto, optei por uma instalação num meu computador pessoal do software de CVS quer para garantir a integridade dos dados quer para me ser de utilização mais fácil e rápida. Este pc pessoal encontra-se ligado à internet continuamente pelo que o conteúdo do CVS pode ser consultado online. O endereço para acesso é <http://float.dns2go.com/cgi-bin/cvsweb.cgi>.

A.2 O que é ?

CVS é um sistema de controle de versões. A sua sigla representa *Concurrent Versions System*. O controlo de versões é normalmente utilizado para código fonte, mas não é restricto a essa utilização. A vantagem principal da utilização deste sistema é que não é guardada apenas a versão actual de um ficheiro, mas cada versão do ficheiro, com registo do que foi alterado, quando, porquê e por quem.

É um conceito assente na indústria e no desenvolvimento de código fonte em geral, que todo o desenvolvimento de software deve ser feito através de um sistema de controlo de versões.

Este sistema trás bastantes benefícios até mesmo para ficheiros modificados apenas por um utilizador, dado que, caso se cometa alguma alteração não prevista no código, pode-se sempre voltar facilmente à versão anterior. São eliminados os problemas de apagar ficheiros, obtém-se um historial de alterações, e caso se encontre um bug que já se pensou ter composto, pode-se facilmente analisar o

que é que se fez.

Quando um grupo de ficheiros é alterado por várias pessoas, as vantagens são ainda mais notórias, dado que é possível várias pessoas trabalharem em ficheiros diferentes, do mesmo projecto e simultaneamente (o desenvolvimento de uns não afectam o desenvolvimento de outros até que tal seja necessário), ou ainda desenvolver os mesmos ficheiros simultaneamente (O *CVS* consegue verificar que partes de um dado ficheiro foram alteradas e, caso as alterações não sejam conflituosas, podem ser automaticamente juntas). Existem alguns conceitos que é preciso ter em conta de forma a perceber mais facilmente o *CVS*:

O repositório Pode ser analisado como uma base de dados central, que contém todos os ficheiros, em todas as suas versões, junto com o histórico e ficheiros de *log*. O repositório é acedido através do comando *cv*s.

A directoria de trabalho - É a copia de todos os ficheiros, mas no sentido normal de *ficheiro* (tudo o que é visto é a última versão desse ficheiro, a menos que seja especificado o contrário).

É efectuado um comando *CVS* para obter uma cópia da última versão de tudo o que está no repositório, (é efectuada uma cópia para a directoria de trabalho). Em seguida são efectuadas as alterações desejadas na directoria de trabalho, e (caso se pretenda que as alterações fiquem permanentes) executam-se mais comandos *CVS* para actualizar o repositório com as alterações. Existem várias maneiras para implementar um controlo de versões, cada uma com diferentes filosofias.

CVS é do tipo *disparar primeiro e fazer perguntas depois* e mete muito poucas restrições aquilo que se pretende fazer (embora possa ser configurado para, por exemplo, restringir múltiplos acessos ao mesmo ficheiro). *CVS* devolve uma cópia de um dado ficheiro, que pode ser alterado em qualquer altura, sem saber, nem se preocupar, se estão duas pessoas a editar o mesmo ficheiro, ao mesmo tempo. A única altura em que o *CVS* verifica os ficheiros, ou estes ficam disponíveis para os outros utilizadores, é quando são efectuadas alterações no repositório.

A.3 Instalar e configurar *CVS*

A.3.1 Instalação inicial

Primeiro é necessário instalar os pacotes de *CVS*. Para uma distribuição de RedHat/Mandrake Linux ou outra que tenha o gerenciamento de pacotes,

- Ir para a directoria que tem o pacote, por exemplo,

```
cd /mnt/cdrom/Redhat/RPMS
```

- Instalar:

```
rpm -i rcs*.rpm
rpm -i cvs*.rpm
```

- Verificar que ficheiros foram instalados:

```
rpm -qpl cvs*.rpm | less
```

Noutras distribuições ou em outros *gostos* de *unix* é necessário descarregar¹ os *tar balls* (*tar.gz*) e seguir os ficheiros *README* e *INSTALL*.

A.3.2 Definição de variáveis de ambiente

As variáveis de ambiente seguintes devem estar configuradas em */etc/profile* (valores por omissão para todos os utilizadores). Caso tal não seja possível e/ou necessário, devem ser adicionadas ao *profile* local de utilizador (*~/.bash_profile*). Mudar conforme o *gosto*:

```
export EDITOR=/bin/vi
export CVSROOT=/home/cvsroot
export CVSREAD=yes
```

Cada utilizador pode alterar as suas variáveis de ambiente no seu ficheiro de *profile* local (*~/.bash_profile*).

```
# File ~/.bash_profile
# Overriding env variables by resetting
export EDITOR=/usr/bin/emacs
export CVSROOT=/home/anotherdir/java/cvsroot
```

A.3.3 Configuração

- Login como *Super Utilizador*

```
[nuno@Old nuno]$ su - root
Password:
```

- Definição das variáveis de ambiente para a sessão actual De notar que a directoria criada em seguida tem o nome *CVSROOT* sem qualquer espaço. Por exemplo, a *CVSROOT* não deve ser */home/my rootcvs*.

```
[root@Old nuno]# export CVSROOT=/home/cvsroot
```

- Criação de novo grupo *cvs*

```
[root@Old nuno]# groupadd cvs
```

- Criação de novo utilizador (*cvs*). A sua directoria base alberga o repositório.

```
[root@Old nuno]# useradd -g cvs -d /home/cvsroot cvs
```

¹Ver: <http://www.cvshome.org>

- Verificação da criação da directoria

```
[root@Old nuno]# ls -ld $CVSROOT
drwxrwx---    5 cvs      cvs          4096 Feb 26 17:39 \
                                     /home/cvsroot/
```

- Novas permissões com acesso de leitura e escrita a utilizadores/grupos de Unix.

```
[root@Old nuno]# chmod o-rwx $CVSROOT
[root@Old nuno]# chmod ug+rwx $CVSROOT
```

- Inicialização do repositório

```
[root@Old nuno]# cvs init
```

- Novo grupo para o utilizador nuno

```
[root@Old nuno]# usermod -G cvs nuno
```

- Login como utilizador do grupo cvs (nuno)

```
[root@Old nuno]# exit
[nuno@Old nuno]$ su - nuno
Password:
```

- Definição das variáveis de ambiente para a sessão actual

```
[nuno@Old nuno]$ export EDITOR='which emacs'
[nuno@Old nuno]$ export CVSROOT=/home/cvsroot
[nuno@Old nuno]$ export CVSREAD=yes
```

- Directoria a importar para o cvs

```
[nuno@Old nuno]$ cd ~/cvs_doc/
```

- Importação

```
[nuno@Old cvs_doc]$ cvs import cvs_doc Vendor1_0 Rev1_0
```

- Verificação

```
[nuno@Old cvs_doc]$ cd ..
[nuno@Old nuno]$ cvs checkout cvs_doc
cvs checkout: Updating cvs_doc
U cvs_doc/cvs.tex
[nuno@Old nuno]$
```

A.3.4 cvs na Web

Para um interface mais simples e intuitivo, os documentos do repositório do *CVS* podem ser acedidos através de um vulgar *browser*. O servidor web tem, no entanto, de ser configurado para disponibilizar esses documentos. Uma possibilidade é a instalação e configuração do módulo *cvsweb*². Uma configuração básica consiste em:

- Descomprimir o código fonte

```
[root@Old tmp]# tar -xzf cvsweb-1.112.tar.gz
```

- Copiar o ficheiro de configuração

```
[root@Old tmp]# cd cvsweb
[root@Old cvsweb]# cp cvsweb.conf /etc/httpd/conf/
```

- Copiar o *script cgi*

```
[root@Old tmp]# cd cvsweb
[root@Old cvsweb]# cp cvsweb.cgi /var/www/cgi-bin/
```

- Editar o ficheiro de configuração do *script cgi* e possivelmente o ficheiro de configuração do servidor *httpd* utilizado. As alterações no *script cgi* é basicamente a definição da *path* de repositório.

```
[root@Old cvsweb]# emacs /etc/httpd/conf/cvsweb.conf
[root@Old cvsweb]# emacs /etc/httpd/conf/httpd.conf
```

- Inserção do utilizador *que corre* o servidor no grupo *cvs* para o acesso ao repositório.

```
[root@Old cvsweb]# usermod -G cvs apache
```

A.4 Para o utilizador

Embora uma descrição muito básica, mas é suficiente para ficar *up & running*. Para o acesso quer de leitura quer de escrita, será necessária a utilização de *SSH*³.

SSH tem então de estar disponível na máquina local. A variável de ambiente *CVS_RSH* deve ter a *path* para *ssh*. Isto é feito na maioria dos sistemas unix (bash) através de :

```
export CVS_RSH=ssh
```

O acesso *CVS* anónimo usa o *CVS pserver* e não requer *SSH*. Caso se obtenha erros de *permission denied* sem nenhuma *prompt* por uma password, esta variável de ambiente não está bem efectuada. É conveniente corrigir isto primeiro antes de suspeitar de um problema de passwords.

²Ver: Hen's cvsweb CVS Repository

³secure shell

A.4.1 O que fazer primeiro

Fazer um *ssh* à conta de utilizador no servidor *CVS*. Isto cria a directoria *home* o que possibilita o uso de *CVS*.

O acesso a *CVS* não funcionará até que este passo seja efectuado.

A.4.2 Como importar código fonte no repositório

Na máquina local, mudar para a directoria que contém os ficheiros (e subdirectorias) que se quer importar. Tudo o que esteja nessa directoria (e subdirectorias) será importado na árvore.

Escrever o seguinte, onde *loginname* é o *login* da conta, *yourproject* é o nome (unix) do grupo projecto, e *directoryname* é o nome da nova directoria *root level* do *CVS*

```
cvs -d:ext:loginname@cvs.yourproject.sourceforge.net:/cvsroot/yo\
    urproject import directoryname vendor start
```

A.4.3 Como verificar a fonte através de SSH

Escrever os seguintes comandos, alterando o nome de utilizador e do projecto.

```
cvs -d:ext:loginname@cvs.yourproject.sourceforge.net:/cvsroot/yo\
    urproject co directoryname
```

Depois de uma verificação inicial, pode-se mudar para essa directoria e executar os comandos *cvs* sem a *tag -d*. Por exemplo :

```
cvs update
cvs commit -m "comments for this commit"
cvs add myfile.c
```

A.4.4 Como verificar a fonte anónimamente através do *pserver*

Escrever os seguintes comandos, alterando o nome de utilizador e do projecto.

```
cvs -d:pserver:anonymous@cvs.yourproject.sourceforge.net:\
    /cvsroot/yourproject login
```

Depois a autenticação anónima :

```
cvs -z8 -d:pserver:anonymous@cvs.yourproject.sourceforge.\
    net:/cvsroot/yourproject co directoryname
```

Depois de uma verificação inicial, pode-se mudar para essa directoria e executar os comandos *cvs* sem a *tag -d*. Por exemplo :

```
cvs update
```

A.5 Referências

1. Per Cederqvist, Version Management with CVS
2. Alavoor Vasudevan, CVS-RCS HOW-TO Document for Linux
3. Per Cederqvist, CVS Manual
4. Karl Fogel, Open Source Development with CVS

Bibliografia

- [1] Federal Information Processing Standards Publication 197, 26/11/2001, Advanced Encryption Standard (AES)
- [2] Joan Daemen e Vincent Rijmen, 03/09/1999, AES Proposal: Rijndael
- [3] Brian Gladman, 04/05/2001, A Specification for The AES Algorithm
- [4] Richard Smith, 05/03/2001, Deciphering the Advanced Encryption Standard
- [5] Elaine Barker, 16/10/2000, NIST Cryptographic Toolkit
- [6] Joan Daemen e Vincent Rijmen, 23/10/2000, Rijndael
- [7] Niels Ferguson, others, 16/05/2001, A simple algebraic representation of Rijndael
- [8] S.V. Raghavan, A Write up on AES
- [9] Haris Domazet, September 26, 2001, AES - Advanced Encryption Standard
- [10] Marcus Leech, 15/08/2000, AES and Beyond - The IETF and Strong Crypto
- [11] Gerwin Sturm, 11/2000, AES vs. DES
- [12] Larry Loeb, November 2000, Exit DES, enter Rijndael
- [13] Niels Ferguson, others, Improved Cryptanalysis of Rijndael
- [14] Jorge Nakahara Jr, July 13, 1999, Key Schedule Analysis of AES Candidates
- [15] Jorg J. Buchholz, December 19, 2001, Matlab Implementation of AES
- [16] Geoffrey Keating, 15 April 1999, Performance Analysis of AES candidates on the 6805 CPU core
- [17] Various, Public Comments on the standard for the AES
- [18] Joan Daemen, Supporting Documentation - Proposal Rijndael
- [19] Joan Daemen, Vincent Rijmen Technical Overview of Rijndael - The AES

- [20] Reinhard Wobst The Advanced Encryption Standard (AES) The Successor of DES
- [21] B. Schneier: Applied Cryptography. Addison-Wesley, (1996).
- [22] Morris Dworkin, December 2001, Recommendation for Block Cipher Modes of Operation.