Universidade do Minho
Escola de Engenharia

José Miguel Crespo Garcia Rodrigues

O uso de técnicas espectrais e SAT na criptoanálise algébrica da cifra AES

Outubro de 2009

Universidade do Minho
Escola de Engenharia

José Miguel Crespo Garcia Rodrigues

O uso de técnicas espectrais e SAT na criptoanálise algébrica da cifra AES

Tese de Mestrado
Mestrado de Informática

Trabalho efectuado sob a orientação do
Professor Doutor Jose Manuel Valença

Outubro de 2009

Declaração

Nome: JOSÉ MIGUEL CRESPO GARCIA RODRIGUES
Endereço electrónico: jose.cgrodrigues@gmail.com Telefone: 918185507
Número do Bilhete de Identidade: 12741491

Título dissertação / tese
O uso de técnicas espectrais e SAT na criptoanálise algébrica da cifra AES
Orientador: Professor Doutor José M. Valença **Ano de conclusão:** 2009
Designação do Mestrado: Mestrado em Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE APENAS PARA EFEITOS DE INVESTIGAÇÃO.

Universidade do Minho, 30/10/2009

Assinatura: José Miguel Crespo Garcia Rodrigues

Agradecimentos

Começo por agradecer ao meu orientador Professor Doutor José M. Valença, não só pela constante disponibilidade e apoio prestados ao longo deste último ano, mas também pela inspiração que foi para mim ao longo de todo o meu percurso académico.

Agradeço também aos meus pais, Manuel e Carmo, por me terem ensinado o valor da honestidade e do trabalho, e àquela que comigo partilha a honra de ser filho deles, a minha irmã Mariana.

À minha namorada e melhor amiga, Emília, pelo seu apoio incondicional, a sua paciência infinita e atenção constante.

E por último, mas nem por isso menos importante, aos meus colegas e amigos, Hugo e Filipe, pelo companheirismo demonstrado ao longo deste últimos anos.

A todos, um sincero e profundo muito obrigado.

Resumo

A cifra AES foi adoptada como cifra *standard* para cifrar informação em vários campos, e da qual depende a segurança dos sistemas de informação. A cifra têm-se provado segura, mesmo após ter sido alvo de diversos ataques.

Sendo o AES uma cifra tão importante, é considerado crucial procurar explorar eventuais vulnerabilidades na cifra. Esta dissertação apresenta um estudo da influência das técnicas espectrais aliadas aos SAT-Solvers na criptoanálise algébrica da cifra.

Para efectuar este estudo, começamos por analisar o problema da *satisfatibilidade booleana* e o funcionamento dos SAT-Solvers. Depois, procuramos conhecer melhor as funções booleanas e a sua importância na criptoanálise. Seguimos com um estudo do design e estrutura algébrica do AES, assim como um levantamento da criptoanálise algébrica existente do AES.

No final da dissertação é apresentado um exemplo das técnicas espectrais aliadas ao problema de SAT e da sua possível viabilidade na criptoanálise algébrica do AES e são apresentadas sugestões para futuros estudos.

Abstract

The AES cipher has been adopted as the encryption standard in the most broad fields, and is the one in which the information services rely. The cipher has been proved as secure, even after being targeted by several attacks.

Being such an important cipher, is of crucial importance to explore eventuals vulnerabilities of the cipher. This thesis presents a study about the importance of SAT-Solvers applied to the use of spectral techniques in the algebraic cryptoanalysis of a cipher.

To accomplish this study, we start by analyzing the boolean satisfiability problem and how SAT-Solvers operate. After, we seek to discover boolean functions and their importance to cryptoanalysis. Followed, with a study of the design and algebraic structure of AES, as well as, a survey of the *state of the art* of the AES algebraic cryptoanalysis.

Finally, we demonstrate a example of resolving the spectral technique problem with a SAT-solver, and the feasibility of this solution applied to the algebraic cryptanalysis of AES. The final remarks and future studies are also considered in this last chapter.

Conteúdo

Capítulo 1

Introdução

Cifras simétricas são, de entre as diferentes classes de técnicas criptográfica, as que são nucleares a qualquer sistema de segurança da informação; são as técnicas mais antigas e delas depende a segurança de muitas outra técnicas criptográficas.

Em Novembro de 2001 foi adoptado como *Advanced Encryption Standard* (AES) a cifra por blocos *Rijndael* após cinco anos de selecção. Esta cifra foi adoptada e adaptada a todos os tipos de contexto de segurança, sendo agora usada para cifrar informações de vários campos: militares, diplomáticos, financeiros, etc... Pode-se afirmar que o AES é a técnica criptográfica da qual, hoje em dia, mais depende a segurança dos sistemas de informação. Por isso, a análise das eventuais vulnerabilidades do AES é de crucial importância.

O AES foi concebido de forma a ter um comportamento óptimo face às duas classes de criptoanálise mais desenvolvidas na altura: a criptnoanálise linear e a criptoanálise diferencial. A criptoanálise algébrica, muito pouco explorada na altura em que o AES foi criado, já foi alvo de muito estudo desde então. Este tipo de criptoanálise é muito recente na área de criptografia simétrica, onde reconhecer padrões estatísticos de bits era tradicionalmente a forma mais efectiva de criptoanálise. Assentando nas propriedades de certas funções booleanas e na combinação destas, e num novo tipo de técnicas designadas técnicas espectrais faz sentido fazer criptoanálise algébrica do AES com base em técnicas espectrais, estudando a forma como esta técnicas benificiam este tipo de criptoanálise.

As técnicas espectrais permitem reduzir problemas cruciais da criptoanálise em problemas de *satisfatibilidade booleana*, também designados por problemas SAT. Este tipo de problemas têm sido alvos já ao longo de vários anos de intensivo estudo, surgindo algoritmos cada vez mais eficientes. Por isso, parece-nos essencial usar esta experiência e verificar a sua vialibilidade na criptoanálise.

1.1 Objectivos

Pretende-se apresentar, nesta dissertação, uma criptoanálise algébrica do AES usando técnicas espectrais, utilizando o resultado para transformar os sistemas resultantes em problemas de *satisfabilidade booleana*. Os sistemas devem ser resolvidos num *SAT solver*, podendo assim estudar a viabilidade das técnicas espectrais na criptoanálise do AES.

Apresentam-se, em seguida, as principais etapas a executar para alcançar o objectivo proposto:

- Estudar o problemas de *satisfabilidade booleana* e o funcionamento algoritmos de SAT.
- O estudo das técnicas espectrais, e o levantamento da criptoanálise algébrica existente ao AES.
- Estudo da viabilidade das técnicas espectrais e SAT na criptoanálise algébrica do AES.

1.2 Estrutura da dissertação

A dissertação encontra-se estruturada em 5 capítulos, organizados, os que a este se seguem, da seguinte forma:

- **Capítulo 2** - Introdução ao problema de satisfabilidade booleana (SAT) e aos SAT solvers.

- **Capítulo 3** - É realizado um *background* matemático e criptográfico, onde se introduzem conceitos matemáticos sobre Corpos Finitos, Bases de Gröbner e Funções Booleanas, e o conceito criptográfico de cifras por blocos.

- **Capítulo 4** - Neste capítulo é apresentada a especificação e propriedades algébricas do AES. E faz-se o levantamento do estado actual criptoanálise existente sobre AES.

- **Capítulo 5** - Apresentação das Técnicas espectrais aplicadas à cripoanálise algébrica do AES. Conclusão da dissertação.

Capítulo 2

Problemas SAT

2.1 Introdução

Na teoria de Complexidade Computacional [34], o problema de SAT (também conhecido por problema de *satisfatibilidade booleana*) consiste no problema de determinar se as variáveis de uma determinada fórmula Booleana *satisfazem*, isto é, podem ter determinado valor de modo a que a formula seja VERDADEIRA. É também importante determinar se a fórmula não é SAT, isso implicaria que a fórmula é FALSA para todas os valores possíveis das variáveis.
Informalmente, podear-se-á dizer que o SAT é uma expressão booleana, na forma normal conjuntiva, formada apenas por AND ($\wedge$), OR ($\vee$), NOT ($\neg$) e variáveis proposicionais. Um problema SAT é representado usando n variáveis proposicionais x_1, x_2, ..., x_n, os quais podemos valorizar como valores 0 (falso) ou 1 (verdadeiro). Um literal l é uma variável x_i ou o seu complemento $\neg x_i$. Uma cláusula ω é uma disjunção de literais e a fórmula CNF γ é uma conjunção de cláusulas. A formula é *satisfatível* se todas as suas cláusulas também o forem.
O teorema de Cook-Levin [32] prova que o problema de satisfatibilidade booleana é um problema NP-completo, sendo este o primeiro *decision problem* provado como sendo NP-completo, podendo assim ser resolvido em tempo polinomial.

Ao longo da ultima década, novos algoritmos, cada vez mais eficientes e escaláveis têm sido desenvolvidos para o SAT, contribuindo para o avanço dos problemas de

SAT, permitindo assim resolver instancias com dezenas de milhares de variáveis e milhões de cláusulas. Existem até competições anuais de SAT.

O problema de SAT é mais significativo do que parece, existem aplicações reais de diversas áreas que dependem em encontrar solução para os problemas de SAT: diversos problemas em EDA (Electronic Design Automation) incluindo *model checking* e verificação formal, Inteligência artifical, design de hardware e algoritmia; O facto de o SAT estar em constante estudo e desenvolvimento e o facto de ser fácilmente implementável também o torna atractivo para investigadores e cientistas, podendo ser aplicável a um infinidade de problemas [23, 20, 19, 10].

2.2 Algoritmos de Resolução e SAT SOLVERS

Existem diversos algoritmos que procuram responder ao problema do SAT: várias vertentes modernas do DPLL, algoritmos genéticos e algorimos estocásticos de busca local. O algoritmo de SAT escolhido nesta dissertação foi o algoritmo de *Davis-Putnam-Logemann-Loveland (DPLL).*

2.2.1 DPLL e sua implementação - Chaff

O algoritmo DPLL (Davis-Putnam-Logemann-Loveland) deriva do algoritmo de Davis-Putnam e é num algoritmo completo, baseado em *backtracking*, que decide se um problema é *satisfatível* ou não. O algoritmo base de *backtracking* consiste em escolher um literal, atribuindo-lhe um valor verdadeiro. Depois simplica a fórmula e recursivamente verificando se a fórmula na versão simplificada é *satisfatível*. Se isto acontecer, a fórmula original é *satisfatível*, caso contrário, a mesma verificação recursiva é feita assumindo o valor de verdade contrário. Este simplificação separa todas as cláusulas que são verdadeiras da fórmula, e todos os literais que são falsos das cláusulas.

O algoritmo DPLL usa as seguintes regras a cada passo de simplificação:

- Propragação Unitária: Se a cláusula for uma *cláusula unitária*, isto é, se só contém um literal, esta cláusula pode ser satisfatível atribuíndo-lhe o valor necessário para o literal ser verdadeiro.

- Eliminação de Literais Puros: Se uma variável proposicional ocorrer apenas com um valor booleano na formula, é considerarda *pura*. Os literais puros podem ser marcados de modo a que todas as cláusulas que o contêm sejam verdadeiras. E como estes literais já não são utilizados na procura podem ser eliminados.

A insatisfatibilidade de uma atribuição parcial é detectada se uma das cláusulas for vazia: todas as suas variáveis foram valorizadas de modo a que os literais correspondentes sejam falsos. A satisfatibilidade de uma fórmula é detectada se todas as cláusulas forem satisfeitas ou se nenhuma cláusula vazia for gerada.

O algoritmo Chaff [25], consiste na implementação do algoritmo DPLL, com algumas melhorias para melhorar eficiência. Foi implementado em C++ na Universidade de Princeton, USA. A implementação a aconselhada nesta dissertação é o SAT-SOLVER zchaff [14, 25], vencedor do SAT COMPETITION 2004 e 2005.

2.2.2 Formato CNF

Uma fórmula booleana está na sua forma normal conjuntiva (*CNF*) se for uma conjunção de *cláusulas*, onde cada cláusula é uma disjunção de um determinado número de variáveis ou a sua negação. Se representarmos as variáveis proposicionais por x_i, e considerando que só assumem o valor *verdadeiro* ou *falso*, um exemplo de uma CNF seria:

Exemplo 2.2.2.1 $(x_1 \vee x_3 \vee \neg x_4) \wedge (x_4) \wedge (x_2 \vee \neg x_3)$

Os problemas de SAT podem ser representados informáticamente pelo tipo de ficheiros de extensão SAT ou extensão CNF, sendo este último o formato mais usado pelos SAT-SOLVERS. Pretende representar as formas normais conjuntivas.

Dado o conjunto de cláusulas C_1, C_2, ..., C_m e as variáveis proposicionais x_1, x_2, ..., x_m, o problema de satisfatibilidade é determinado se a fórmula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

é *satisfatível*. Isto é, podemos atribuir determinados valores às cláusulas de modo a que a fórmula apresentada anteriormente seja considerada valorada como verdadeira: cada C_i têm que ser valorado como verdadeiro.

No ficheiro de input CNF, a forma normal conjuntiva é representada num é um ficheiro de texto.

Usando o Exemplo anterior:

Exemplo 2.2.2.2 *(*$x_1 \vee x_3 \vee \neg x_4$*)* $\wedge$ *(*x_4*)* $\wedge$ *(*$x_2 \vee \neg x_3$*)*

Em formato CNF seria:

Exemplo 2.2.2.3 *Ficheiro de Exemplo:*
c EXEMPLO DE CNF
c p cnf 4 3
1 3 -4 0
4 0
2 -3

Capítulo 3

Background Matemático e Criptográfico

Neste capítulo são apresentadas alguns conceitos matemáticos e de criptografia, que embora não sejam essencias para a implementação do AES, são importantes para o entendimento da cifra e desta dissertação. As primeiras secções fornecem teoria dos números de Corpos Finitos, Funções Booleanas e Bases de Gröbner. A ultima secção introduz conceitos importantes de Cifras de Blocos.

3.1 Corpos Finitos

3.1.1 Grupos, Anéis e Corpos

Os Grupos, os Aneis e os Corpos constituem as estruturas básicas da álgebra abstracta [21, 17]. Um grupo Abeliano $< G, + >$ é um conjunto G e a operação $'+'$ definida pelos seus elementos:

$+ : G \times G \rightarrow : (a, b) \mapsto a + b.$

De modo a ser considerado um grupo Abeliano, a operação tem que satisfazer as seguintes condições:

1. *fechada*: $\forall a, b \in G : a + b \in G$

2. *associatividade*: $\forall a, b, c \in G : (a + b) + c = a + (b + c)$

3. *comutatividade*: $\forall a, b \in G : a + b = b + a$

4. *elemento neutro*: $\exists 0 \in G, \forall a \in G : a + 0 = a$

5. *elementos invertíveis*: $\forall a \in G, \exists b \in G : a + b = 0$

Exemplo 3.1.1.1 *O grupo de inteiros* $\mathbb{Z}$ *com a operação adição é um grupo Abeliano. Se n for um inteiro positivo, o grupo de inteiros* $\mathbb{Z}_n = 0, \ldots, n - 1$ *dentro da operação adição módulo n forma um grupo Abeliano de ordem n.*

Um anel R é um conjunto não-vazio $< R, +, \cdot >$, com duas operações $+$ e $\cdot$. É considerado um *anel* se as seguintes condições se verificarem:

1. $< R, + >$ é um grupo Abeliano.

2. A operação $\cdot$ é distribuitiva em $+$, para todos os $r, r', r'' \in R$,

$$r \cdot (r' + r'') = r \cdot r' + r \cdot r'' \quad e \quad (r' + r'') \cdot r = r' \cdot r + r'' \cdot r.$$

3. Existe um elemento $1 \in R$ de modo a que $1 \cdot r = r \cdot 1 = r$ para todo $r \in R$.

O elemento 1 é chamado de *elemento identidade* do anel $(R, +, \cdot)$. Se a operação $\cdot$ é comutativa, então o anel é um anel *comutativo*.

Um Corpo é um anel de estrutura $< F, +, \cdot >$ onde:

1. $< F, +, \cdot >$ é um anel comutativo.

2. Para todos os elementos de F, existe um elemento inverso em F da operação '$\cdot$', excepto para o elemento 0, o *elemento neutro* de $< F, + >$.

3.1.2 Corpos Finitos

O Design do AES é baseado em Corpos Finitos [8]: todas as operações usadas pelo AES podem ser representadas por operações algébricas entre corpos finitos com a mesma característica.

Um Corpo Finito ou *Corpo de Galois (GF)* é um corpo que contém um número finito de elementos [21, 17]. Um Corpo de Galois é classificado pela ordem dos elementos (número de elementos do corpo). A ordem pode ser um número primo p, ou então uma potência de número primo p^n, onde p é a *característica* do número primo, sendo n um número inteiro positivo. Para $\mathbb{Z}_p = \{0, \ldots, p-1\}$, a notação utilizada é $GF(p)$.

Corpos Finitos com a mesma ordem, isto é, com o mesmo número de elementos, são *isomórficos*: apresentam exactamente a mesma estrutura algébrica diferindo apenas na representação dos elementos. Ou seja, para cada potência de número primo existe apenas um corpo finito $GF(p^n)$.

3.1.3 Polinómios em Corpos Finitos

Um polinómio de um corpo F pode-se representar da seguinte maneira:

$$b(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \ldots + b_2x^2 + b_1x + b_0,$$

sendo x o *indeterminante* do polinómio, e o $b_i \in F$ o *coeficiente*.

Os polinómios podem ser representados em strings de *bits*, marcando os coeficientes como positivos ("1"):

Exemplo 3.1.3.1 *Por exemplo, o polinómio:* $m(x) = x^6 + x^4 + x + 1$ *em binário ficaria "01010011", em hexadecimal* 53.

A **Adição**, no caso particular dos Corpos Finitos de característica 2, GF(2), é o OR exclusivo (XOR), $\oplus$:

$$c(x) = a(x) + b(x) \Leftrightarrow c_i = a_i + b_i, 0 \leq i < n$$

Exemplo 3.1.3.2 *Seja F um corpo GF(2). A soma dos polinómios* 57 *e* 83 *é D4:*

$$\begin{aligned}(x^6 + x^4 + x^2 + x + 1) \oplus (x^7 + x + 1) &= x^7 + x^6 + x^4 + x^2 + (1 \oplus 1)x + (1 \oplus 1) \\ &= x^7 + x^6 + x^4 + x^2\end{aligned}$$

O que equivale em binário a $01010111 \oplus 10000011 = 11010100$.

Para a **Multiplicação** de dois polinómios em GF(2) *fechada* é necessário seleccionar um polinómio $m(x)$ írredutivel. A Multiplicação entre dois polinómios $a(x)$ e $b(x)$ é definida como o produto algébrico dos dois polinómios módulo polinómio $m(x)$:

$$c(x) = a(x) \cdot b(x) \Leftrightarrow c(x) \equiv a(x) \times b(x) \pmod{m(x)}.$$

Exemplo 3.1.3.3 *No Corpo $GF(2^8)$, usando o polinómio irredutível do AES como polinómio de redução: $m(x) = x^8 + x^4 + x^3 + x + 1$. $57 \times 83 = C1$:*

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

e

$$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \equiv x^7 + x^6 + 1 \pmod{x^8 + x^4 + x^3 + x + 1}$$

3.2 Funções Booleanas

Um função Booleana é uma função na forma $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ ou $f : \mathbb{B}^n \rightarrow \mathbb{B}$, em que $\mathbb{B}^n$ é uma sequência de *n-bits* [39]. Os Corpos Finitos são representados por variáveis Booleanas 0 e 1 (bits). De entre as várias representações possíveis para funções booleanas, vamos-nos concentrar nas seguintes:

1. Funções booleanas de n bits

$$f : GF(2)^n \rightarrow GF(2)$$

2. Funções booleanas sobre o corpo de Galois de ordem n

$$f : GF(2^n) \rightarrow GF(2)$$

3.2.1 Funções de argumento GF(2^n)

Os argumentos das funções boolenas são vectores de *bits*, se os vectores forem da mesma dimensão podemos aplicar as seguintes operações binárias: $\oplus$ (*bitwise xor*) e a $*$ (*bitwise and*) [39]

$$(x \oplus y)_i = x_i + y_i \quad (x * y)_i = x_i \cdot y_i$$

As funções booleanas podem ser representadas na chamada **forma normal algébrica** [11]. Representando a função booleana de domínio $GF(2)^n$ como um somatório, para n inputs [33]:

$$f(x) = a_0 \oplus \bigoplus_{1 \le i \le n} a_i x_i \oplus \bigoplus_{1 \le i \le j \le n} a_{ij} x_i x_j \oplus \ldots \oplus a_{12\ldots n} x_1 x_2 \ldots x_n.$$

Temos então um polinómio a de n variáveis $x_0, x_1, \ldots, x_{n-1}$ [39], obtido em:

$$f(x) = \sum_{u \in \mathbb{U}_n} a(u) x_u \quad e \quad a : \mathbb{U}_n \to GF(2)$$

em que $\mathbb{U}_n = 2^{\mathbb{Z}_n}$ representa o conjunto de índices. A função a é a *função de índices*, ou *espectro de índices* e determina a **forma normal algébrica** de f. Equivalentemente podemos representar as funções booleanas apenas pelos respectivos espectros, consideremos as funções booleanas seguintes:

$$y(x) = 1 + x_0 + x_1 + x_3$$

e

$$z(x) = x_0 + x_3 \cdot x_4$$

Os espectros das funções y e z, respectivamente seriam, $\{\emptyset,\ \{0\},\ \{1\},\ \{3\}\}$ e $\{\{0\},\ \{3,\ 4\}\}$.

3.2.2 Funções de argumento GF$(2)^n$

Consideremos uma função booleana vectorial $f : GF(2)^n \rightarrow GF(2)^n$. Esta função pode ser descrita por um vector de n funções booleanas escalares:

$$f(x_1, x_2, \ldots, x_n) = \begin{bmatrix} h_1(x_1, x_2, \ldots, x_n) \\ h_2(x_1, x_2, \ldots, x_n) \\ \cdots \\ h_n(x_1, x_2, \ldots, x_n) \end{bmatrix}$$

Ou como é designado em criptografia, uma **S-BOX** $n \times n$. Nestas funções os argumentos são vistos como elementos do corpo de Galois de ordem n. Como o domínio é finito, podemos descrever a função como um polinómio de grau inferior 2^{n-1} [39].

$$f(x) = \sum_{i=0}^{2^n-1} a_i x^i, \quad a_i \in GF(2^n)$$

Como iremos ver no capítulo 4, o AES especifica uma função ByteSub, da qual faz parte uma transformação afim que opera em $GF(2)^8$ (e não em $GF(2^8)$). É necessário então proceder à mudanda de representação linear das funçoes boolenas de $GF(2)^n$ para $GF(2^n)$.

Consideremos uma função $(\cdot)_\sigma : GF(2^n) \rightarrow GF(2^n)^n$. Esta função transforma um elemento x do corpo de Galois no vector formado por todos os $\sigma^k(x)$. Sendo $\sigma : x \rightarrow x^2$ as potências do morfismo de Frobenius. Temos então:

$$x_\sigma \doteq (x, \sigma(x), \sigma^2(x), \ldots, \sigma^{n-1}(x))$$

[39]. As funções lineares do tipo $f : GF(2^n) \rightarrow K$, com K uma extensão de $GF(2)$, são funções que preservam operações escalares[39]. O que nos permite escrever uma função linear f como um polinómio:

$$f(x) = \sum_{k=0}^{n-1} a_k \sigma^k(x), \quad com\ a_k \in K$$

Sendo a o vector $(a_0, a_1, \ldots, a_{n-1})$, podemos representar o polinómio como um produto escalar de dois vectores:

$$f(x) = a * x_\sigma$$

Considerando $f(x) = y$. Podemos calcurar y_σ:

$$y_\sigma = Ax_\sigma,$$

em que A é uma matriz: $A_{ij} = (a_k)^{2i}$ com $k = j - i \pmod{n-1}$ [39]

Seja $\mathcal{B}$ uma base de $GF(2^n)$. Esta base determina n funções booleanas (*projeções*) $p_\mu : GF(2^n) \to GF(2)$, em que cada elemento $\mu \in \mathcal{B}$. De modo a podermos construir $x \in GF(2^n)$ apartir dos elementos μ e de $p_\mu(x)$:

$$x = \sum_{\mu \in \mathcal{B}} p_\mu(x)\mu$$

Definimos $(\cdot)^\sim : GF(2^n) \to GF(2)^n$ como a função que associa x ao vector das suas projecções. [39] Vectorialmente:

$$x = \mathcal{B} \cdot x^\sim$$

Para cada elemento de $\mu \in \mathcal{B}$ de $GF(2^n)$ existe um elemento $\mu' \in GF(2^n)$, o qual designamos *elemento dual* de μ, de modo a que $\forall x \in GF(2^n)$,

$$p_\mu(x) = tr(\mu' x) = (\mu')_\sigma \cdot x_\sigma$$

onde $tr(x)$ é a função traço. A *base dual* de $\mathcal{B}$ em $GF(2^n)$ é o conjunto de todos elementos duais, é representada pelo conjunto $\mathcal{B}' = \{\mu' \mid \mu \in \mathcal{B}\}$. Podemos provar [39] que, considerando a matriz dos traços cruzados $T_{\mu v} = tr(\mu v)$, temos que $\mathcal{B}' = T^{-1}\mathcal{B}$.

O *elemento dual* de x na base $\mathcal{B}$, representando por x', é o elemento de $GF(2^n)$ que tem exactamente as mesmas componentes de x mas na base dual B'. Para todo μ:

$$p_\mu(x) = tr(\mu' x) = tr(\mu x') = p_{u'}(x')$$

O operador $(\cdot)^{\sim}$ associa cada elemento de $GF(2^n)$ ao vector das suas projecções na base B:

$$x'^{\sim} = T^{-1}x^{\sim} \quad , \quad x' = \mathcal{B} \cdot x'^{\sim} = \mathcal{B}' \cdot x^{\sim}$$

O operador $(\cdot)_\sigma$ definido anteriormente pode ser extendido para elementos $GF(2^n)$. Dado $A \in GF(2^n)^n$ a matriz $A_\sigma \in GF(2^n)^{n \times n}$ tem, por linha de ordem i, o vector $(A_i)_\sigma$. Ou seja, $(A_\sigma)_{ik} = \sigma^k(A_i)$. [39]

Sendo $\mathcal{B}$ e $\mathcal{B}'$ um par de bases duais então:

1. $(\mathcal{B}')_\sigma = T^{-1}\mathcal{B}_\sigma$
2. $x^{\sim} = (\mathcal{B}')_\sigma x_\sigma \quad e \quad x_\sigma = (\mathcal{B}_\sigma)^t x^s im$
3. $(\mathcal{B}_\sigma)^t \mathcal{B}_\sigma = T \quad e \quad (\mathcal{B}_\sigma)^t (\mathcal{B}^t)_\sigma = I$
4. $\mathcal{B}_\sigma (x')_\sigma = (\mathcal{B}')_\sigma x_\sigma$

Todas estas noções apresentadas são essencias ao estudo de funções booleaas que requerem mudança de representação $GF(2^n)$ para a representação $GF(2)^n$. Permite-nos resolver o seguinte problema:

"Dada uma base de representação $\mathcal{B}$ em $GF(2^n)$, dada uma função $f^{\sim}$ de domínio $GF(2)^n$, construir a função f de domínio $GF(2^n)$ que verifica $f^{\sim}(x^{\sim}) = f(x)^{\sim}$, para todo o x. Ou, inversamente, dada a função f, construir $f^{\sim}$."[39]

3.3 Bases de Gröbner

Nesta secção explicamos o conceito de Base de Gröbner, conceito importante para a criptnoanálise algébrica de cifras. Para um estudo mais aprofundado ver [2, 7].

Seja k um Corpo Finito ($\mathbb{F}_q$ por exemplo) e $R = k[x1, \ldots, x_n]$ um anel de polinómios de múltiplas variáveis. Consideremos o seguinte sistema de equações:

$$f_1(x_1, \ldots, x_n) = \ldots = f_m(x_1, \ldots, x_n) = 0$$

O *ideal* I é gerado por $f_1, \ldots, f_m$. Um monómio em $x_1, \ldots, x_n$ é um termo em $x_1, \ldots, x_n$ e também um coeficiente. Escolhemos $<$ para ordenar os monómios em $x_1, \ldots, x_n$. $LT(I)$ define o grupo de todos os *leading terms* de elementos de I, e $\langle LT(I) \rangle$ denota o ideal gerado por todos os monómios em $LT(I)$. Um grupo finito $G = g_1, \ldots, g_s \subset I$ é considerado uma *Base de Gröbner* de I se

$$\langle LT(g_1), \ldots, LT(g_s) \rangle = \langle LT(I) \rangle$$

Portanto G é uma Base de Gröbner de I se e só se o *leading term* de qualquer polinómio em I for divisível por pelo menos um dos *leading terms* $\{LT(g_1), \ldots, LT(g_s)\}$

Seja K um corpo que contém k, podemos então definir o grupo de soluções em K, baseando-nos no *Teorema de Bases de Hilbert* [12] a *variedade algébrica* é:

$$V_K = \{(z_1, \ldots, z_n) \in K \quad | \quad f_i(z_1, \ldots, z_n) = 0, i = 1, \ldots, m\}$$

Sendo esta a solução (grupo de raízes) do sistema de equações iniciais.

As Bases de Gröbner são um conceito muito poderoso e útil, com várias aplicações em álgebra comutativa, geometria algébrica e álgebra computacional.

O principal uso das Bases de Gröbner em criptografia é a capacidade de resolver grandes sistemas de equações polinomiais. Se temos o sistema de equações:

$$f_1(x_1, \ldots, x_n) = 0, \ldots, f_m(x_1, \ldots, x_n) = 0,$$

então conseguimos encontrar o conjunto de soluções computando as Bases de Gröbner para o ideal $I = \langle f_1, \ldots, f_m \rangle$ e computando a variedade algébrica associada $V(I)$.

3.4 Cifras por Blocos

As cifras por blocos transformam blocos de tamanho fixo n_b de texto limpo (*plaintext*) em texto cifrado (*ciphertext*), utilizando uma chave secreta k. São cifras simétricas, a chave criptográfica utilizada para *cifrar* o texto, é a mesma utilizada para *decifrar* o "texto cifrado". Mais precisamente, um cifra por blocos é um conjunto de permutações Booleanas entre vectores de n_b-bit. Se o número de bits da cifra for n_k, a cifra consiste em 2^{nk} permutações Booleanas [22, 15].

Nas *Cifras por Blocos Iterativas*, os *round transformation* (permutações Booleanas) são iterativos. Cada aplicação da permutação booleana é um *round*. Para cada round é derivada uma chave correspondente através de uma permutação da chave da cifra. Este processo chama-se *Key schedule*. Considerando r o número de *rounds*, temos:

$$B[k] = \rho^{(r)}[k^{(r)}] \circ \ldots \circ \rho^{(2)}[k^{(2)}] \circ \rho^{(1)}[k^{(1)}]$$

Nesta expressão, $p^{(i)}$ é o round-i da cifra e $k^{(i)}$ é a chave correspondente ao round i da cifra.

As chaves $k^{(i)}$ de cada round são computadas apartir da chave secreta k inicial, através do algoritmo de *Key shedule*.

As cifras por blocos que usam a mesma *round transformation* em todos os *rounds* (com a excepção do round inicial ou final) são chamadas *Cifra por Blocos Iterativa* [8]. Como podemos observar na figura seguinte:

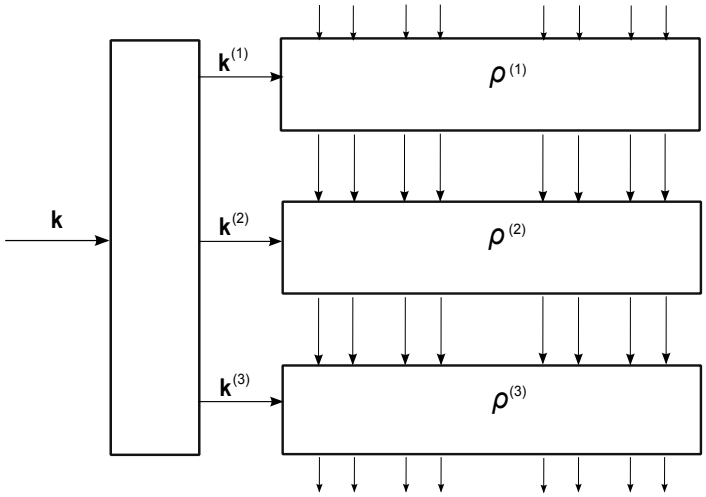

Figura 3.1: *Cifra por Blocos Iterativa*

Existe outro tipo de cifras por blocos, as *Key-Alternating Block Cipher* [8]. Observando a figura seguinte, podemos verificar que este tipo de cifras é muito similar às cifras por blocos iterativas. No entanto esta cifra têm algumas particularidades: a cifra é definida alternando a aplicação de *rounds transformation* independentes da

chave (*Key independent rounds*) e rounds em que se adiciona a chave (*Key addition rounds*). Sendo a *Key Addition* um simples XOR ($\oplus$), o que permite também uma fácil implementação.

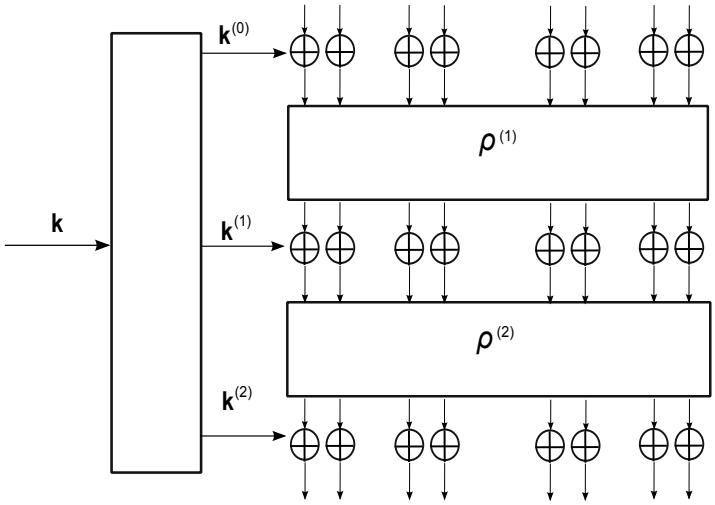

Figura 3.2: *Key-Alternating Block Cipher*

Temos então:

$$B[k] = \sigma[k^{(r)}] \circ \rho^{(r)} \circ \sigma[k^{(r-1)}] \circ \ldots \circ \sigma[k^{(1)}] \circ \rho^{(1)}\sigma[k^{(0)}]$$

sendo $\sigma[k]$ o processo de *Key Addition*.

Este tipo de cifra tende a ser resistente a diversos tipos de criptoanálise. Existe também um caso especial de *Key-Alternating Block Cipher*, as *Key-iterated block cipher*. Neste ultimo caso, todos os rounds da cifra usam a mesma round transformation [8]:

$$B[k] = \sigma[k^{(r)}] \circ \rho \circ \sigma[k^{(r-1)}] \circ \ldots \circ \sigma[k^{(1)}] \circ \rho \circ \sigma[k^{(0)}]$$

em que ρ é a *round transformation* da cifra.

Este ultimo tipo de cifras é o usado pelo AES.

Capítulo 4

A cifra AES

4.1 Introdução

Em Janeiro de 1997, a *National Institute of Standards and Technology (NIST)*) anunciou o desenvolvimento de um novo standard de criptografia: o *Advanced Encryption Standard (AES)*, com o intuito de especificar um algoritmo de funcionameto público capaz de proteger dados governamentais sensíveis durante o próximo século. O AES prometia assim, ser o substituto ideal para o DES e o triple-DES [28].

O NIST lançou um concurso público para o desenvolvimento do AES (ao contrário do DES, SHA-1 e DSA), quinze cifras candidatas foram propostas, sendo apenas cinco seleccionadas para o 1º round. A cada submissão, a comunidade de criptografia iria efectuar ataques e fazer criptoanálise das cifras candidatas. Qualquer indivíduo poderia enviar os seus testes e comentários para a NIST quer via web, quer nas conferências do AES.

Em Outubro de 2000, o NIST anunciou que o Rijndael como cifra vencedora. Publicaram também um relatório a explicar as razões que motivaram esta escolha, realçando características importantes e úteis: a consistência e facilidade de implementação da cifra quer em hardware, quer em software. A fácil configuração das chaves, o pouco uso de memória e transparência. Rijndael também provou ser uma cifra segura contra *Timing* e *Power Attacks*. O desenho por rounds também é uma vantagem, permitindo a eficiente implementação com paralelismo [27].

No início o AES iria só vir a ser usado nos EUA, no entanto, devido ao seu grande sucesso como sucessor do DES, foi adoptado como standard em vários campos: militares, diplomáticos, financeiros... e aprovado como standard na *International Organization for Standardization (ISO), Internet Engineering Task Force (IETF* e *Institute of Electrical and Electronics Engineers (IEEE).* As razões para o AES ser tão rapidamente aceitado pela indústria e comunidade em geral foi a sua facilidade de implementação em vários tipos de plataformas e o facto de ser grátis.

A única diferença entre o Rijndael e o AES é o *range* dos valores suportados para o tamanho de bloco e o tamanho da cifra. Enquanto o Rijndael suporta qualquer múltiplo de 32 bits (com um mínimo de 128 bits e um máximo de 256 bits). O AES especifica o tamanho do bloco em 128 bits e suporta chaves de 128, 192 ou 256 bits [36].

O AES baseia-se, tal como o DES, nas ideias de Shannon e nos conceitos de difusão e confusão. A difusão é alcançada através do uso de permutações e têm como objectivo difundir a influência de todo o input por todos os blocos. A confusão pretende tornar a relação entre o *plaintext*, o *ciphertext* e a chave usada complexa, isso é alcançado pelo uso *Substitution Boxes* [36].

4.2 Estrutura do AES

$$S = \begin{array}{|c|c|c|c|} \hline \mathcal{B}_0 & \mathcal{B}_4 & \mathcal{B}_8 & \mathcal{B}_{12} \\ \hline \mathcal{B}_1 & \mathcal{B}_5 & \mathcal{B}_9 & \mathcal{B}_{13} \\ \hline \mathcal{B}_2 & \mathcal{B}_6 & \mathcal{B}_{10} & \mathcal{B}_{14} \\ \hline \mathcal{B}_3 & \mathcal{B}_7 & \mathcal{B}_{11} & \mathcal{B}_{15} \\ \hline \end{array}$$

Tabela 4.1: *Array de Bytes do AES representando o estado*

O AES é uma *key iterated block cipher*: consiste em sucessivas aplicações de *rounds tranformation* no *estado*. O número de rounds é chamado de N_r e depende do tamanho do bloco e da chave. Podemos considerar que o AES é uma série de operações num

array quadrado de 16 bytes (AES-128), ou equivalentemente, uma coluna de vector de bytes, representando o *estado*, como se pode observar na tabela 4.1. Ao longo do processo de cifra, a estrutura de bytes é inteiramente respeitada.

Um byte pode ser visto como uma sequência ordenada de 8 bits: $b_7b_6b_5b_4b_3b_2b_1b_0$ e pode ser interpretado como um vector de dimensão 8 elemento de GF(2).

No standard do AES [37], cada byte pertence a um corpo finito GF(2^8) definido pelo polinómio irredutível:

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

Considerando este corpo $\mathbb{F}$. Temos que $\mathbb{F} = \frac{GF(2)\ [x]}{\langle m(x)\rangle}$

O byte é equivalente a:

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

É comum representar o byte usando notação hexadecimal, por exemplo, $F0$ representa o vector de bits 11110000 e o elemento $x^7 + x^6 + x^5 + x^4$ do corpo.

O AES-128 transforma *plaintext* em *ciphertext* através de estados de 128 bits, ou seja, estados de 16 bytes. Observando a tabela 4.1 podemos verificar que cada estado pode ser representado pela string de bits:

$$\mathcal{B}_0\mathcal{B}_1\mathcal{B}_2\ \ldots\ \mathcal{B}_{15}$$

O array S de dimensão 4×4 definido por

$$S_{i,j} = P_{4j+i}\ (0 \leq i,\ j \leq 3)$$

representa o *estado*.

4.2.1 Cifrar

Existem quatro operações básicas no AES, essas operações processam-se sobre o array de 16 bytes:

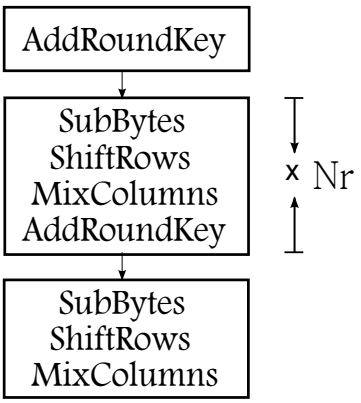

Figura 4.1: *Cifra no AES*

- *S*ubBytes modifica os bytes no array independentemente.
- *S*hiftRows faz a rotação de quatro casas no array independentemente.
- *M*ixColumns modifica as quartro colunas do array independentemente.
- *A*ddRoundKey adiciona os bytes da *round key* e do array.

As quatro operações formam um round de cifra. O *Round-0* (round inicial) apenas tem a operação *AddRoundKey* seguido de N_r rounds de computação, rounds estes que variam conforme a versão do AES, no caso do AES-128 seriam 10 rounds. O último round não têm a operação de *MixColumns*. Como esquematizado na figura 4.1.

4.2.1.1 SubBytes

A operação de SubBytes é a única transformação não-linear da cifra. É aplicada uma permutação a cada byte inicialmente, usando a S-BOX do Rijndael, à qual chamamos de S_{RD} ilustrada na tabela 4.2 sob a forma de *lookup table*.

Esta operação modifica o array-estado S da seguinte forma:

$$S_{i,j} \mapsto S[S_{i,j}] \ (0 \leq i,\ j \leq 3)$$

ou equivalentemente:

$$\mathcal{B}_i \mapsto S[\mathcal{B}_i] \ (0 \leq i \leq 15)$$

S_{RD} =

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

Tabela 4.2: S-BOX usada em *SubBytes*. O valor $S[xy]$ é dado pela intersecção da linha x e a coluna y.

Conforme especificado em [36], a S_{RD} foi desenhado com os seguintes objectivos:

- Não-linearidade: A amplitude de correlação máxima de input/output deve ser mínima e a diferença entre as probabilidades de propagação deverá ser o mais-pequena possível.
- Complexidade Algébrica: A expressão algébrica de S_{RD} em $GF(2^8)$ deve ser complexa.

A operação de ByteSub pode ser vista como a função composta de duas funções: ByteSub $= f \circ g$. A função g, também conhecida como função *auto-inversa* é definida da seguintes forma em GF(2^8):

$$g(x) = x^{-1}$$

A função *auto-inversa*, embora simples, é conhecida por ter boas propriedades não-lineares.

Por definição, g é descrita por uma expressão algébrica simples, o que permitiria manipulações algébricas que poderiam ser usadas para atacar a cifrar. Então, o ByteSub têm uma função f afim. Esta função não têm influência nas propriedades não-lineares e permite uma expressão algébrica mais complexa.

A transformação afim representada por f define-se por:

$$f(x) = Hx \oplus b$$

$$\Updownarrow$$

$$\begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

ou representando cada byte em base hexadecimal e a matriz como um vector de colunas:

$b = 63 \quad H = (8F, C7, E3, F1, F8, 7C, 3E, 1F)$

Podemos considerar então que: $S_{RD}[a] = f(g(a))$

4.2.1.2 ShiftRows

Este passo consiste na transposição nas linhas da matriz do estado. Os bytes recebem um shift para a esquerda, como se pode observar na figura 4.2.

A linha 0 mantêm-se, a segunda linha (1) recebe um shift para a esquerda. As linhas 2 e 3 recebem 2 e 3 sihifts, respectivamente. Este passo foi implementado com o objectivo de aumentar a difusão óptima, aumentado a resistência contra *saturations attacks* e *Truncated differential attacks.*

$\mathcal{B}_0$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{12}$
$\mathcal{B}_1$	$\mathcal{B}_5$	$\mathcal{B}_9$	$\mathcal{B}_{13}$
$\mathcal{B}_2$	$\mathcal{B}_6$	$\mathcal{B}_{10}$	$\mathcal{B}_{14}$
$\mathcal{B}_3$	$\mathcal{B}_7$	$\mathcal{B}_{11}$	$\mathcal{B}_{15}$

$\xrightarrow{ShiftRows}$

$\mathcal{B}_0$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{12}$
$\mathcal{B}_5$	$\mathcal{B}_9$	$\mathcal{B}_{13}$	$\mathcal{B}_1$
$\mathcal{B}_{10}$	$\mathcal{B}_{14}$	$\mathcal{B}_2$	$\mathcal{B}_6$
$\mathcal{B}_{15}$	$\mathcal{B}_3$	$\mathcal{B}_7$	$\mathcal{B}_{11}$

Figura 4.2: Operação ShiftRows

4.2.1.3 MixColumns

Este passo traduz-se por uma permutação *bricklayer*, operando no estado coluna por coluna. A operação foi implementada de modo a permitir uma transformação *bricklayer* operando numa coluna de 4 bytes, aumentado a difusão da cifra. A operação deve ser linear em $GF(2)$. O MixColumns foi desenhado de modo a ter boa performance em processadores 8 bits.

Cada coluna do estado é tratada como um polinómio em $GF(2^8)$ e depois é multiplicado módulo $x^4 + 1$ com o polinómio $c(x)$, dado por:

$$c(x) = 3x^3 + x^2 + x + 2$$

O polinómio é co-primo com $x^4 + 1$ e como tal é invertível.

Esta operação pode ser descrita como uma operação matricial.

Seja $r(x) = c(x) \cdot a(x) \ \ (mod x^4 + 1)$.

Então:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Equivalentemente:

$$r_0 = 2a_0 + a_3 + a_2 + 3a_1$$

$$r_1 = 2a_1 + a_0 + a_3 + 3a_2$$

$$r_2 = 2a_2 + a_1 + a_0 + 3a_3$$

$$r_3 = 2a_3 + a_2 + a_1 + 3a_0$$

4.2.1.4 AddRoundKey

A operação de AddRoundKey é a operação que combina a sub-chave gerada pelo com o estado. A sub-chave é gerada através do algoritmo de *Key Schedule*, sendo do tamanho do estado. A combinação consiste num simples *bitwise* XOR, como se pode observar na figura 4.3.

$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|} \hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array}$$

Figura 4.3: Operação *AddRoundKey*, a sub-chave gerada é adicionada ao estado através de um *bitwise* XOR

4.2.2 Key Schedule

O *Key Schedule* do AES divide-se em duas componentes: a geração da chave Expandida *Expanded Key* e a selecção das sub-chaves para os rounds correspondentes (*round-keys*).

O número total de bits da *Expanded Key* é igual ao tamanho do bloco multiplicado pelo número de rounds mais 1. Por exemplo, para um bloco de 128bits e 10 rounds, são necessários 1408 bits.

A *Expanded Key*, gerada na expansão da chave, é um array linear de palavras de 4-bytes que consistem em 4 linhas e N_C $(N_r + 1)$ colunas. Consideremos este array por $W[4][N_b(N_r+1)]$. *A round key* da *round-i*, $ExpandedKey[i]$ é dada pelas colunas $N_b \cdot i$ até $N_b \cdot (i+1)\cdot$ de W:

$ExpandedKey[i] =$

$$W[\cdot][N_b \cot i] \;||\; W[\cdot][N_b \cdot i + 1] \;||\; \ldots \;||\; W[\cdot][N_b \cdot (i+1) - 1], \quad 0 \leq i \leq N_r$$

Nas primeiras N_k colunas de W são preenchidos pela *Cipher Key*, as colunas

seguintes, por sua vez são definidas recursivamente. A recursão usa bytes da coluna anterior, bytes da célula N_k posições atrás e as constantes de round RC[i].

A função de recursão depende então da posição da coluna. Se a coluna i não for múltipla de N_k a coluna i é o resultado de um *bitwise XOR* entre a coluna i-N_k e $i-1$. Caso contrário é o resultado de bitwise XOR entre a coluna i-N_k e a função não linear da coluna $i-1$. Para tamanhos de chave superior a 6. A função não-linear consiste na aplicação da S-Box do Rijndael aos 4 bytes da coluna, uma rotação cíclica dos bytes da coluna e a adição de uma constante de Round (para eliminação da simetria).

As constantes de round são independentes de N_k, e são definidas pela seguintes regra de recursão dem $GF(2^8)$:

$RC[1] = x^0$

$RC[2] = x$

$RC[j] = x \cdot RC[j-1] = x^{j-1} \ , \ j > 2$

4.2.3 Decifragem

O algoritmo para decifrar consiste simplesmente no usa das operações inversas: *InvSubBytes, InvShiftRows, InvMixColumns* e *AddRoundKey* na ordem inversa.

4.2.3.1 InvSubBytes

A S-box $S^{-1}[\cdot]$ é facilmente calculada. A operação modifica o estado: $B_i \mapsto S^{-1}[B_i]$ $(0 \leq i \leq 15)$. Como se pode observar na *lookup table* 4.3.

4.2.3.2 InvShiftRows

Consiste na rotação de i-posições para a direita de cada linha de estado. O byte na posição j da linha move-se i posições para a posição $(j + C_i)$ mod N_B (em que C_1 é o número da coluna e N_b o número de bytes).

S_{RD} =

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

Tabela 4.3: S-BOX usada em *InvSubBytes*. O valor $S[xy]$ é dado pela intersecção da linha x e a coluna y.

4.2.3.3 InvMixColumns

A operaçao *InvMixColumns* consiste na inversa de *MixColumns*. Cada coluna é transformada multiplicando-a por um polinómio $d(x)$, definido por:

$$(03 + x^3 + 01 \cdot x^2 + C1 \cdot x + 02) \cdot d(x) \equiv 01 \pmod{x^4 + 1}$$

Então:

$$d(x) = 0B + x^3 + 0D \cdot x^2 + 09 \cdot x + 0E$$

Esta operação na forma matricial:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 0E & 09 & 0D & 0B \\ 0B & 0E & 09 & 0D \\ 0D & 0B & 0E & 09 \\ 09 & 0D & 0B & 0E \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

4.2.3.4 AddRoundKey

A operação inversa de AddRoundKey mantêm-se a mesma, pois o *bitwise* XOR é simétrico.

4.2.4 Filosofia de Design do AES

Cada componente do AES foi cuidadosamente escolhida e possuí uma tarefa específica [8, 9], com diversos critérios de design:

- Segurança: Provavelmente o critério mais importante, a cifra deve ser segura. A sua estrutura interna deve ser resistente a criptoanálise existente.
- Eficiência: A cifra deve usar poucos recursos para a cifragem/decifragem dos dados.
- Versatibilidade: A cifra deve ser compatível e fácilmente implementada em várias arquitecturas/sistemas diferentes. Não interessando por exemplo o número de bits em que o processador opera.
- Simplicidade: O AES é uma cifra simples, que pode ser dividida em várias sub-processos simples.
- Simétrica: o facto do AES ser uma cifra simples, permite simetria: Existe simetria em todos os passos, em todas as transformações, em todos os rounds. A simetria permite paralelismo. A ordem de como as S-Boxes são processadas não é relevante, podendo ser computadas em paralelo. A simetria permite também vário tamanhos de bloco.

Cada round do AES pode ser dividido em três componentes: O primeiro é o SubBytes, em que ocorre uma substituição em cada byte do array de estado, representanto a camada de substituição (*Substitution Layer*). A segunda parte é a operação de ShiftRows seguido da operação de MixColumns. Esta parte introduz difusão no array de estado. A parte final do AES consiste na adição da chave no AddRoundKey.

A camada de substituição é baseada na S-Box do AES, S-Box esta que consiste na composição de três operações:

- Inversão: A inversão do AES consiste na multiplicação no corpo finito do AES, extendido de modo a $0 \rightarrow 0$. O byte de input da S-Box é tomado como um elemento do corpo e para $w \neq 0$, é output x satisfaz $x = w^{-1}$ e $wx = 1$
- Transformação Afim: É aplicada uma transformação afim linear $GF(2)^8 \rightarrow GF(2)^8$.
- A constate da S-Box: O output da transformação afim como um elemento do corpo e adiciona o elemento 63.

A inversa multiplicativa introduz resistência local [29, 30] à criptoanálise diferêncial e à criptoanálise linear. A transformação afim e a adição da constante aumentam a complexidade algébrica da S-Box.

A camada de difusão do AES é introduzida pela matriz 4×4 do corpo F usado na MixColumns. A matriz é uma matriz *MDS (maximam distance separable*) [18], que protege o AES contra ataques diferenciais e lineares.

4.3 Propriedades algébricas do AES

4.3.1 Representações algébricas do AES

A base de muitos ataques ao AES consiste em descrever a cifra de blocos como um sistema de equações [40] polinomiais extendidas para o corpo $\mathbb{F}_2$. Uma chance de quebrar o AES seria reduzir o espaço de chaves, pois um ataque extensivo no AES actualmente não é computacionalmente possível.

Se aplicarmos consecutivamente as quatro operações do AES (ByteSub, ShiftRow, MixColumns e AddRoundKey) a uma coluna de 4 bytes obtemos a seguinte equação

algébrica:

$$\begin{pmatrix} s'_{3,c} \\ s'_{2,c} \\ s'_{1,c} \\ s'_{0,c} \end{pmatrix} = S[s_{3,c(3)}] \cdot \begin{pmatrix} 02 \\ 03 \\ 01 \\ 01 \end{pmatrix} \oplus S[s_{2,c(2)}] \cdot \begin{pmatrix} 01 \\ 02 \\ 03 \\ 01 \end{pmatrix} \oplus S[s_{1,c(1)}] \cdot \begin{pmatrix} 01 \\ 01 \\ 02 \\ 03 \end{pmatrix}$$

$$\oplus S[s_{0,c(0)}] \cdot \begin{pmatrix} 03 \\ 01 \\ 01 \\ 02 \end{pmatrix} \oplus \begin{pmatrix} k_{3,c} \\ k_{2,c} \\ k_{1,c} \\ k_{0,c} \end{pmatrix}.$$

Com a S-Box (S) e a round key k_i. O $c(0) = c$ e $c(x) = [c + h(r, N_c)]$ $(mod\ N_c)$, sendo que $h(r, N_c)$ é 1, 2, 3 se o número de colunas $N_c \in [4, 5, 6]$; 1, 2, 4 se $N_c = 7$ e 1, 3, 4 se $N_c = 8$. Em cada round é usada a equação anterior à excepção do último round (que não têm a operação de MixColumns). A operação inversa também pode ser descrita por uma equação algébrica similar usando a S-Box inversa.

A S-Box do AES também pode ser desenhada como uma equação na forma [13, 8]:

$$S(x) = w_8 + \sum_{d=0}^{7} w_d x^{255-2^d},$$

para algumas constantes $w_0, \ldots, w_8$. Aproveitando-se desta estrutura, já se foi capaz de demonstrar [13], que o AES pode ser descrito como uma equação algébrica, uma fracção contínua (generalizando) em F_{2^8}.

Seja $a_{i,j}^{(r)}$ o input e $k_{i,j}^{(r)}$ a sub-chave do round na posição (i, j) do round r respectivamente. Definimos $E = \{0, 1, 2, 3\}$ e $D = \{0, 1, \ldots, 7\}$. Aplicando a S-Box a cada byte de input do estado obtemos:

$$S_{i,j}^{(r)} = S[a_{i,j}^{(r)}] = \sum_{d=0}^{7} w_{d_r} (a_{i,j}^{(r)})^{-2^d r}.$$

Aplicando a transformação ShiftRow:

$$t_{i,j}^{(r)} = S_{i,i+j}^{(r)} = \sum_{d=0}^{7} w_{d_r} (a_{i,i+j}^{(r)})^{-2^d r}.$$

Continuando com a transformação MixColumns:

$$m_{i,j}^{(r)} \;=\; \sum_{e_r=0}^{3} v_{i,e_r} t_{e_r,j}^{(r)} \;,$$

onde v_i, j são os coeficientes da matriz MDS usada na transformação de MixColumns.

Podemos concluir que um round do AES satisfaz

$$a_{i,j}^{(r+1)} = k_{i,j}^{(r)} + \sum_{\substack{e_r \in E \\ d_r \in D}} \frac{w_{i,e_r,d_r}}{(a_{e_r,e_r+k}^{(r)})^{2^{d_r}}}.$$

Substituindo com mais um round obtemos:

$$a_{i,j}^{(3)} = k_{i,j}^{(2)} + \sum_{\substack{e_2 \in E \\ d_2 \in D}} \frac{w_{i,e_2,d_2}}{k_{e_2,e_2+j}^{(1)} + \sum_{\substack{e_1 \in E \\ d_1 \in D}} \frac{w_{e_2,e_1,d_1}}{(a_{e_1,e_1+e_2+j}^{(1)})^{2^{d_1}}}}.$$

Generalizando para a cifra toda:

$$x = K + \sum \frac{C_1}{K^* + \sum \frac{C_2}{K^* + \sum \frac{C_3}{K^* + \sum \frac{C_4}{K^* + \sum \frac{C_5}{K^* + p_*^*}}}}},$$

em que K é um byte que depende em vários bytes da chave expandida, C_i é uma constante e cada $*$ é um expoente conhecido.

Esta equação final têm cerca de 2^{25} termos [31]. Para quebrar o Rijndael de 10 rounds (AES-128), o atacante poderia usar para cada byte intermédio 2 equações deste tipo. A primeira expressa as variáveis intermédias depois de 5 rounds em função dos bytes de *plain text*. A segunda equação cobre os rounds de 6 a 10, descrevendo as mesmas variáveis intermédias em função dos *ciphertext bytes*. Combinando as duas equações resultaria uma equação de 2^{26} variáveis, repetindo esta equação, com $2^{26}/16$ pares de plaintext/ciphertext conhecidos teriamos informação suficiente para quebrar a cifra.

4.3.2 Big Encryption System

Já foram propostas diversas representações para o AES, representações estas que tentam explorar a estrutura da cifra e são geralmente construídas definindo um homomorfismo do estado do AES e o espaço de chave. Uma representação do AES é

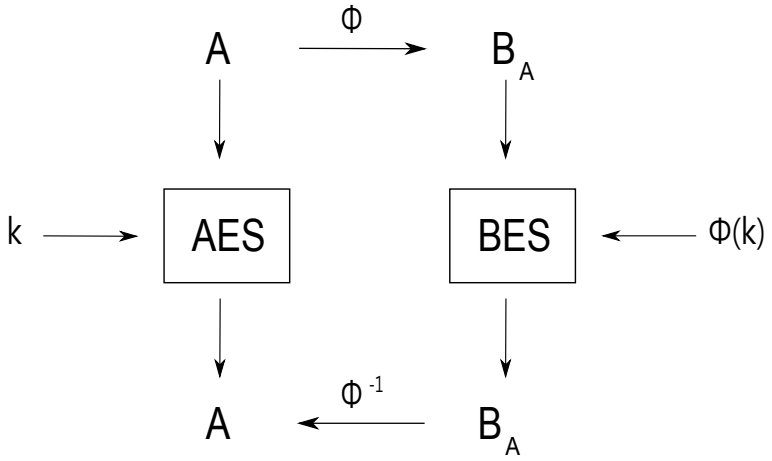

Figura 4.4: *Big Encryption System*

derivada "introduzindo"o AES numa cifra maior chamada de *Big Encryption System* (BES) [26]. O BES é definido de maneira a replicar o AES usando operações algébricas simples em $\mathbb{F}$ (Corpo Finito do AES). O Bes opera em blocs de 128-bytes com chaves de 128-bytes e possui uma estrutura algébrica simples. Um round do BES consiste na inversão de cada um desses 128-bytes e na transformação afim.

$$\mathbb{F} \ = \mathbb{F}_{2^8} \ = \frac{\mathbb{F}_2[X]}{(X^8 + X^4 + X^3 + X - 1)}$$

Para mapearmos o AES no BES designamos o *space state* do BES por $\mathbb{B} = \mathbb{F}^{128}$ em vez do AES, em que $A = \mathbb{F}^{16}$. O mapeamento do AES no BES é baseado no vector $\phi : \mathbb{F} \to \mathbb{F}^8$, que mapeia um elemento de $\mathbb{F}$ para um vector de dimensão 8. ϕ é um anel homomórfico injectivo dado por:

$$a \mapsto (a^{2^0}, a^{2^1}, a^{2^2}, a^{2^3}, a^{2^4}, a^{2^5}, a^{2^6}, a^{2^7})^T$$

Podemos extender a definição para uma função $\phi : A \to B$ definido por:

$$(a_0, \ldots, a_{15})^T \to (\phi(a_0 =, \ldots, \phi(a_{15}))^T,$$

também um anel homomórfico injectivo.

Consideremos $B_A = \phi(A) \subset B$, a cifra BES é definida de modo a ficar comutativa.

Analisemos então a estrutura do BES (para uma descrição mais detalhada ver [26]).

Seja $b \in B$ um vector de estado do BES (vector, ao invés de uma matriz como seria no AES). $k_b \in B$ é uma sub-chave do round i. A operação de ShiftRows do AES pode ser representada pela matriz $R_A : \mathbb{F}^{16} \to \mathbb{F}^{16}$. Substituindo cada 1 em R_A pela matriz identidade I_8 e cada 0 por uma matriz 8×8 preenchida por 0's obtemos a matriz $R_B : \mathbb{F}^{128} \to \mathbb{F}^{128}$. A componente afim $L_A : \mathbb{F} \to \mathbb{F}$ (actuando em cada byte) da S-Box do AES pode ser representada pelo polinómio $f : \mathbb{F} \to \mathbb{F}$

$$f(a) = \sum_{k=0}^{7} \lambda_k a^{2^k},$$

onde $(\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7) = (05, 09, F9, 25, F4, 01, B5, 8F)$, ou na forma polinomial,

$$\lambda_0 = t^2 + 1;\ \lambda_4 = t^7 + t^6 + t^5 + t^4 + t^2;$$

$$\lambda_1 = t^3 + 1;\ \lambda_5 = 1;$$

$$\lambda_2 = t^7 + t^6 + t^5 + t^4 + t^3 + 1;\ \lambda_6 = t^7 + t^5 + t^4 + t^2 + 1;$$

$$\lambda_3 = t^5 + t^2 + 1;\ \lambda_7 = t^7 + t^2 + t + 1;$$

L_A pode ser extendida para B por

$$L_B(a) = \phi(L_A(a)) = (f(x)^{2^0}, \ldots, f(a)^{2^7}).$$

Definindo a transformação $Lin_B : \mathbb{F}^{128} \to \mathbb{F}^{128}$ como uma matriz diagonal com 16 blocos iguais a $L_B = [l_{ij}]_{i,j=0,\ldots,7}$, onde $l_{ij} = \lambda^{2^i}_{(8-i+j)\ (mod 8)^*}$. A constante c_A da transformação da S-Box é

$$c_A = \theta^6 + \theta^5 + \theta + 1 \in \mathbb{F}$$

O homomorfismo ϕ aplicado ao c_A:

$$\phi(c_A) = (63, C2, 35, 66, D3, 2F, 39, 36),$$

Repetindo ϕ 16 vezes obtemos a constante:

$$c_B = (\phi(c_A), \ldots, \phi(c_A));\ [c_B]_i = [\phi(c_A)]_{i(mod8)^*}$$

O passo de MixColumns pode ser descrito na forma polinomial por:

$$C_A = \begin{pmatrix} \theta & 1 & 1 & \theta+1 \\ \theta+1 & \theta & 1 & 1 \\ 1 & \theta+1 & \theta & 1 \\ 1 & 1 & \theta+1 & \theta \end{pmatrix}$$

e a transformação de MixColumns é dada por $Mix_A : \mathbb{F}^{16} \rightarrow \mathbb{F}^{16}$. Extendendo para uma matriz diagonal $M_b : \mathbb{F}^{128} \rightarrow \mathbb{F}^{128}$ [38] com 4 cópias consecutivas da matriz $C_B^{(k)}$ para todos os k possíveis:

$$C_B^{(k)} = \begin{pmatrix} \theta^{2^k} & 1 & 1 & (\theta+1)^{2^k} \\ (\theta+1)^{2^k} & \theta^{2^k} & 1 & 1 \\ 1 & (\theta+1)^{2^k} & \theta^{2^k} & 1 \\ 1 & 1 & (\theta+1)^{2^k} & \theta^{2^k} \end{pmatrix}$$

com a reduções apropriadas temos:

$$\theta^{2^0} = \theta;\ \ \theta^{2^1} = \theta^2;\ \ \theta^{2^2} = \theta^4;$$

$$\theta^{2^3} = \theta^4 + \theta^3 + \theta + 1;\ \ \theta^{2^4} = \theta^6 + \theta^4 + \theta^3 + \theta^2 + \theta;$$

$$\theta^{2^5} = \theta^7 + \theta^6 + \theta^5 + \theta^2;\ \ \theta^{2^6} = \theta^6 + \theta^3 + \theta^2 + 1;$$

$$\theta^{2^7} = \theta^7 + \theta^6 + \theta^5 + \theta^4 + \theta^3 + \theta..$$

Representando a transformação $Mix_B : \mathbb{F}^{128} \rightarrow \mathbb{F}^{128}$ por $Mix_B = Perm_B^{-1} \cdot M_B \cdot Perm_B$, em que a matriz $Perm_B$ consiste em sub-matrizes P_{hk} 16×8 definidas por $[P_{hk}]_{ij} = 1$ se $i = k$ e $j = k$, se não $[P_{hk}]_{ij} = 0$. A matriz da permutação inversa $Perm_B^{-1}$ é descrita similarmente em [38]. Todas as 32 matrizes (4×4) em Mix_B são matrizes MDS, contribuindo para a difusão da cifra.

Um round no BES aceita como input $b \in B$ e uma sub-chave $(h_B)_i$ que funciona da seguinte maneira

$$\begin{aligned} R_i(b, k_b)_i &= Mix_B(R_B(Lin_B(b^{-1}) + c_b)) + (h_b)_i \\ &= M_b \cdot (b^{-1}) + (C_b(c_b)) + (h_b)_i \\ &= M_b \cdot b^{-1} + (k_b)_i, \end{aligned}$$

onde $M_B = Mix_B \cdot R_B \cdot Lin_B$ é a matriz 128×128 em $\mathbb{F}$, e executa a difusão linear no BES, $C_B = Mix_B \cdot R_B$, $(h_B)_i = C_B(c_b) + (h_B)_i$.

O algoritmo de *Key Schedule* do AES também é extendido para o BES da seguinte maneira. A função que executa uma função cíclica no AES ($Rotword[b_3, b_2, b_1, b_0] = [b_0, b_3, b_2, b_1]$) é representada no AES pela seguinte matriz $RW_A : \mathbb{F}^4 \to \mathbb{F}^4$:

$$RW_A \ = \ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Extendendo ao BES, $RW_B : \mathbb{F}^{32} \to \mathbb{F}^{32}$ é obtida, substituindo os 1's da matriz indentidade I_8 e os 0's por matrizes de zeros 8×8. Definimos $Lin_B^k : \mathbb{F}^{32} \to \mathbb{F}^{32}$ como uma matriz diagonal com 4 blocos iguais a L_B [38]. A constante envolvida no processo de *Key Scheduling*, c_B^k é:

$$c_B^k = \phi(c_A, c_A, c_A, c_A) \ = \ (\phi(c_A), \ldots, \phi(c_A)); \ [c_B^k] \ = \ [\phi(c_A)]_{i(mod 8)^*}$$

O array de palavras $Rcon[i]$ do AES é mapeado em

$$Rcon_B[i] = \phi(Rcon[i]) = (0, \ldots, 0, \phi(\theta^{i-1})).$$

O mapeamento do round i do BES é

$$\phi_B^i(x) = Lin_B^k(RW_B(x))^{-1} + c_B^k + Rcon_B[i],$$

e as matrizes das chaves genéricas do AES e do BES, respectivamente:

$$MK_A^i = \begin{pmatrix} 0 & 0 & 0 & \phi_A^i \\ 0 & I_4 & 0 & \phi_A^i \\ 0 & I_4 & I_4 & \phi_A^i \\ 0 & I_4 & I_4 & I_4 + \phi_A^i \end{pmatrix}$$

$$MK_B^i = \begin{pmatrix} 0 & 0 & 0 & \phi_B^i \\ 0 & I_{32} & 0 & \phi_B^i \\ 0 & I_{32} & I_{32} & \phi_B^i \\ 0 & I_{32} & I_{32} & I_{32} + \phi_B^i \end{pmatrix}$$

com a computação de *Key Schedule* a ser dada por $h_i = MK_B^i(h_{i-1})$.

Considerando $p \in B$ o *plaintext* e $c \in B$ o *ciphertext*, o vector de estado antes e depois da i-inversão $w_i \in B$ e $x_i \in B$ $(0 \leq i \leq 9)$, respectivamente. Sabendo que o último round do BES, tal como o no AES não usa a operação de MixColumns, sendo $M_B^k = R_B \cdot Lin_N = Mix_B^{-1} \cdot M_B$. O processo de cifragem do BES é dado por [26, 38].

$$\begin{aligned} w_0 &= p + k_0, \\ x_i &= w_i^{-1}, para 0 \leq 1 \leq 9, \\ w_i &= M_B x_{i-1} + k_i, para 1 \leq i \leq 9, \\ c &= M_B^* x_9 + k_{10}, \end{aligned}$$

incluindo a operação inicial de adição da chave e uma matriz diferente no último round.

O BES pode ser assim descrito por operações algébricas simples no corpo finito $\mathbb{F}$, permitindo outra abordagem à cifra: Usando o BES somos capazes de obter sistemas de equações quadráticas multivariavéis em $GF(2^8)$ que descrevem o AES. Veremos à frente que este sistema é mais simples do que obtemos inicialmente no AES.

4.3.3 Sistema de Equações para o BES

Como analisamos anteriormente, as expressões algébricas apresentadas para o processo de cifragem do AES demonstraram-se demasiado complexas. Iremos então, derivar um sistema de equações para o BES partindo o AES.

Analisando as expressões algébricas anteriores podemos obter o seguinte sistema para o processo de cifragem de BES:

$$
\begin{aligned}
0 &= w_{0,(j,m)} + p_{(j,m)} + k_{0,(j,m)}, \\
0 &= x_{i,(j,m)} w_{i,(j,m)} + 1 \; para 0 \leq 1 \leq 9, \\
0 &= w_{i,(j,m)} + (M_B x_{i-1})_{(j,m)} + k_{i,(j,m)} \; para \; 1 \leq i \leq 9, \\
0 &= c_{(j,m)} + (M_B^* x_9)_{(j,m)} + k_{10,(j,m)},
\end{aligned}
$$

onde (j, m) $(0 \leq j \leq 15, 0 \leq m \leq 7))$ denota o componente $(8j + m)$ de todos os vectores. Se considerarmos $\alpha_{(j,m)}$ e $\beta_{(j,m)}$ as entradas de M_B, M_B^* e assuminto que nunca acontece a inversão do 0 (o que acontece para 53% das cifragens e 85% das chaves de 128-bits [26]) obtemos o seguinte sistema de equações quadráticas multi variáveis (*MQ Equations*):

$$
\begin{aligned}
0 &= w_{0,(j,m)} + p_{(j,m)} + k_{0,(j,m)}, \\
0 &= x_{i,(j,m)} w_{i,(j,m)} + 1 \; para 0 \leq 1 \leq 9, \\
0 &= w_{i,(j,m)} + k_{i,(j,m)} + \sum_{(j',m')} \alpha_{(j,m),(j',m')} x_{i-1,(j',m')} \; para \; 1 \leq i \leq 9, \\
0 &= c_{(j,m)} + k_{10,(j,m)} + \sum_{(j',m')} \beta_{(j,m),(j',m')} x_{9,(j',m')}.
\end{aligned}
$$

O processo de Cifragem do BES também pode ser descrito como um sistema MQ com 2688 equações (em $\mathbb{F}$), das quais 1280 são quadráticas e o resto linear. Este sistema contém 5248 termos (2560 variáveis de estado e 1408 variáveis de chave). Seja δ_i cada um dos componentes de $cR_i = cB^k + Rcon_B[i]$, que representa o vector em cada round. O *Key Schedule* do BES pode ser descrito por um sistema MQ

[5, 26, 38]:

$$
\begin{aligned}
0 &= z_{i,(\tilde{j},m)} + h^{254}_{i-1,(12+[(\tilde{j}+1)(mod4)],m)} \\
0 &= h_{i,(\tilde{j},m)} + \delta_{i,(\tilde{j},m)} + \sum_{(\tilde{j}',m')} \gamma_{(\tilde{j},m)(\tilde{j}',m')} z_{i,(\tilde{j}',m')} \\
0 &= h_{i,(4s+\tilde{j},m)} + h_{i,(4(s-1)+\tilde{j},m)} + h_{i-1,(4s+\tilde{j},m)} \quad i \leq s \leq 3 \\
0 &= k_{i,(t,m)} + (C_B(c_B))_{(t,m)} + h_{i,(t,m)} \\
0 &= z^2_{i,(\tilde{j},m)} + z_{i,(\tilde{j},m+1)} \\
0 &= h_{i,(\tilde{j},m)} + h_{i,(\tilde{j},m+1)},
\end{aligned}
$$

onde $\gamma_{(j,m)}$ são os valores de entrada de Lin^k_B e $1 \leq i \leq 10$, $0 \leq \tilde{j}, \tilde{j}' \leq 3$, $0 \leq m$ e $m' \leq 7$.

A cifra do AES enquanto cifra embebida do BES, gera mais equações MQ, nomeadamente:

$$
\begin{aligned}
0 &= w_{0,(j,m)} + p_{(j,m)} + k_{0,(j,m)} \\
0 &= w_{i,(j,m)} + k_{0,(j,m)} + \sum_{(j',m')} \alpha_{(j,m),(j',m')} x_{i-1,(j',m')} \quad para\ 0 \leq i \leq 9 \\
0 &= c_{(j,m)} + k_{10,(j,m)} + \sum_{(j',m')} \beta_{(j,m),(j',m')} x_{9,(j',m')} \\
0 &= x_{i,(j,m)} w_{i,(j,m)} + 1 \quad para\ 0 \leq 1 \leq 9 \\
0 &= x^2_{i,(j,m)} + x_{i,(j,m+1)} \quad para\ 0 \leq 1 \leq 9 \\
0 &= w^2_{i,(j,m)} + w_{i,(j,m+1)} \quad para\ 0 \leq 1 \leq 9.
\end{aligned}
$$

Podemos contar assim o número de equações do processo de cifragem do AES [5, 26, 38]. Temos 5248 equações, das quais 3840 são quadráticas e 1408 são lineares. O sistema MQ contém 7808 termos, 2560 variáveis de estado e 1408 variáveis de chave. O *Key Schedule* do AES pode ser expresso como um sistema MQ com 2560 equações (em F), das quais 960 são quadráticas e 1600 equações lineares. O sistema contém 2368 termos com 2048 variáveis (1408 são variáveis de chave e 640 são variáveis auxiliares) [26].

Murphy e Robshaw em [26], apontam que como as operações do BES podem ser definidas num único corpo, a representação possui várias propriedades que talvez

venham a permitir uma criptoanálise fácil, com poucos pares *plaintext/ciphertext* necessários.

4.3.4 Sistema de Equações usando Bases de Gröbner

Seguindo a aproximação de [4] e considerando a operação não-linear na S-Box como $x \rightarrow x^{254}$ em vez da inversa no Corpo Finito $\mathbb{F}$ do Rijndael. Estes sistemas contém equações que são mais densas e de grau mais complexo do que os sistemas anteriores. No entanto o sistema de equações de [4] possui algumas propriedades algébricas interessantes no processo de cifragem do AES.

Considerando $w_i = (w_{i,0}, \ldots, w_{i,15}) \in \mathbb{F}^{16}$ o input do round $(0 \leq i \leq 9)$ e $w_i = (k_{i,0}, \ldots, k_{i,15})$ a chave do round $(0 \leq i \leq 10)$. Denotamos o output da operação de SubBytes por $S(w_i) = (g(w_{i,0}), \ldots, g(w_{i,15}))$, onde o polinómio $g(z)$ é o polinómio interpolador da S-Box e é dado por

$$05z^{254} + 09z^{253} + F9z^{251} + 25z^{247} + F4z^{239} + 01z^{223} + B5z^{191} + 8Fz^{127} + 64$$

Considerando p e c como o *plaintext* e o *ciphertext* respectivamente, então o processo de cifragem do AES é dado por:

$$\begin{aligned} w_0 &= p + k_0 \\ w_i &= \bar{C}\bar{R}(S(w_{i-1})) + k_i \quad (1 \leq i \leq 9) \\ c &= \bar{R}(S(w_9)) + k_{10} \end{aligned}$$

onde $\bar{R}$ e $\bar{C}$ são matrizes 16×16 em $\mathbb{F}$, correspondendo às operações de ShiftRows e MixColumns respectivamente.

Podemos rearranjar o sistema para obter:

$$\begin{aligned} 0 &= w_0 + p + k_0 \\ 0 &= S(w_{i-1}) + (\bar{C}\ \bar{R})^{-1}(w_i + k_i) \quad (1 \leq i \leq 9) \\ 0 &= S(w_9) + \bar{R}^{-1}(k_{10} + c). \end{aligned}$$

Obtemos assim um sistema de equações com 176 equações, das quais 16 são lineares e as outras 160 têm um grau de 254 cada uma.

05	CF	B3	16	55	C0	7A	01	22	D8	6B	A6	1F	0D	BC
49	85	B4	1B	5E	BD	18	1D	6D	C5	23	09	43	68	80
6C	CC	42	9F	0F	D2	3B	2C	5F	BE	AE	E4	93	8B	CB
65	C0	1E	8E	32	1D	A5	76	A9	2C	13	05	60	FD	1B
AB	64	C1	A8	7F	55	DB	EC	20	C4	DB	7E	92	80	A3
59	91	91	81	4E	11	DD	4E	D3	E3	19	E7	03	24	45
DA	EA	87	2D	23	82	38	B7	9E	B3	2A	3E	1C	EC	C3
45	ED	D5	2A	8D	ED	37	26	E0	BC	58	E2	6C	24	55
C7	AA	09	4F	82	CA	10	EE	1A	2E	40	27	81	92	B1
02	8B	87	7F	B0	6F	53	08	CB	03	B0	DF	1F	A7	A2
FE	8E	A8	E1	71	FF	55	5A	1D	9D	BF	E8	BA	6B	72
E3	04	D9	38	D3	B9	16	52	18	19	3E	9E	03	56	A6
71	03	E4	86	F5	B0	05	D1	10	E2	E5	CB	B1	F2	8E
C7	0C	A7	BF	46	0B	01	C5	A3	50	77	EA	05	65	8E
89	D4	6D	D3	75	65	13	2F	86	AF	7C	7B	85	C8	E8
04	7B	CF	2F	8A	9A	3D	CF	21	39	D9	29	73	F6	23
40	1B	B2	C0	6D	85	1C	8A	2C	BB	90	1E	7E	F3	52

Tabela 4.4: Coeficientes do polinómio interpolador para a S-Box inversa.

Podemos rearranjar similarmente as equações do *Key Schedule* usando a S-Box inversa, obtendo um polinómio interpolador. Os coeficientes deste polinómio $h(x)$ são dados na seguinte tabela 4.4, onde

$$h(z) = 05z^{254} + CFz^{253} + \ldots + F3z + 52.$$

Obtemos assim: $(1 \leq i \leq 10)$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} h(k_{i,0} + k_{(i-1),0} + \theta^{i-1}) \\ h(k_{i,1} + k_{(i-1),1}) \\ h(k_{i,2} + k_{(i-1),2}) \\ h(k_{i,3} + k_{(i-1),3}) \\ k_{i,4} + k_{(i-1),4} \\ \vdots \\ k_{i,15} + k_{(i-1),15} \end{pmatrix} + \begin{pmatrix} k_{i-1,15} \\ k_{i-1,12}) \\ k_{i-1,13}) \\ k_{i-1,14}) \\ k_{i,0} \\ \vdots \\ k_{i,11} \end{pmatrix}$$

Construímos assim um sistema de equações em $\mathbb{F}$ para um processo de cifragem com 336 variáveis. Este sistema de equações têm 176 equações polinomiais do sistema

de cifragem e 160 do *Key Schedule*. Desses 336, 200 têm um grau de 254 e as 136 restantes são lineares. Também podemos considerar este sistema como um grupo de polinómios no anel de polinómios multi variáveis:

$$F[w_{0,0}, \ldots, w_{0,15}, k_{0,0}, \ldots, k_{10,15}, w_{1,0}, \ldots, w_{9,15}]$$

com 336 variáveis no corpo $\mathbb{F}$. Podemos encontrar um estudo mais aprofundado sobre o tema em [5].

4.4 Ataque XSL ao AES

Baseado no algoritmo XL, o ataque XSL (*eXtended Sparse Linearisation*) funciona explorando vulnerabilidades da estrutra do sistema de equações (*iterated block cipher*). O algoritmo XSL foi introzido em [6].

No XSL as equações são multiplicadas apenas por certos monómios pré-seleccionados, e as equações são multiplicadas por todos os monómios até um certo grau do algoritmo XL. O XSL gera um número elevado de equações cujos termos são produto de monómios seleccionados. O objectivo é criar menos monómios novos enquanto se gear as novas equações no sistema extendido de equações. O algoritmo XSL também incorpora um passo extra chamado *método T'*, no qual novas equações linearmente independentes são geradas sem criar novos monómios. A versão mais recente do algoritmo XLS pode ser descrita na figura 4.5.

A ideia básica do XSL é expandir o sistema original multiplicando as equações apenas pelo produto dos monómios que já existem no sistema original de equações. Para um sistema mais esparso, isto diminui potencialmente o número de monómios gerados no sistema expandido de equações. E como o algoritmo XSL é baseado no meto de linearização, o algoritmo benificiará dos sistemas *overdefined*.

O algoritmo de XSL pretende explorar a vulnerabilidade da estrutura de alguns tipos de cifras por blocos. Na sua versão mais básica, o algoritmo de XSL assume que a cifra por blocos é construída com camadas de pequenas S-Box interconectadas por uma transformação afim dependente da chave. E assumindo que a S-Box pode

1: **Input**: Sistema de equações polinomiais de cifras por blocos $f_1 = \ldots = f_m = 0$.
2: **Output**: Solução $(a_1, \ldots, a_n)$ onde $f_1(a_1, \ldots, a_n)) = \ldots = f(a_1, ..., a_n) = 0$.
3:
4: Escolher certo grupos de monómios e equações que supostamente irão ser usadas nos passos finais dos algoritmos, baseando-nos na análise do grupo original da equação.
5: Seleccionar o valor do parametro P e multiplicar as equações escolhidas pelo produto dos monómios seleccionados $P - 1$.
6: Executar o *métodos T'*, no qual algumas equações seleccionadas são multiplicadas por variáveis únicas.
7: Iterar T' com as variáveis necessárias até o sistema extendido de equações ter suficientes equações linearmente independentes para aplicar a linearização.
8:
9: **Retornar**: Solução do sistema extendido obtido através da linearização.

Figura 4.5: Descrição Heurística de um algoritmo XSL

ser descrita como um conjunto *overdefined* de equações quadráticas. Por exemplo o AES e a cifra por blocos *SERPENT*[3], ambas usam S-Boxes que dão origem a um sistema de equações quadráticas. O sistema de equações para estas cifras por blocos são esparsos e o XSL é tira vantagem disto quando o sistema é expandido, antes da linearização. Para as versões do XSL que usam equações de *Key Schedule*, o *Key Schedule* deve ter uma estrutura similar à transformação de cifragem do estado, como é o caso do AES.

A ideia por trás do XSL consiste em gerar equações com um grau mais elevado do que o set original de equações. Analisando o sistema de equações como combinações lineares e resolver o sistema linear. Existema já algumas aplicações de sucesso do XSL em versões com rounds reduzidos do Rijndael [16].

Capítulo 5

Conclusão

Neste capítulo é discutida a possível implementação da técnicas espectrais em conjunto com a criptoanálise algébrica de modo a permitir passar uma cifra como o AES para um problema de SAT. É discutida, passo a passo, a técnica a utilizar e os problemas envolvidos. A técnica é primeiro demonstrada em pequena escala, para várias *Toy Ciphers*, depois é discutida a sua implementação à criptoanálise algébrica feita ao BES no cápitulo 4, concluindo-se se é viável a utilização de SAT-Solvers neste tipo de problemas, conforme a dimensão estimada.

5.1 Análise do Problema e Método a utilizar

Pretende-se uma criptoanálise algébrica do AES usando técnicas espectrais, isto é, aplicar as técnicas espectrais à criptoanálise algébrica apresentada no capítulo 4. Utilizando o resultado passam-se os sistemas resultantes para problemas de SAT (*satisfatibilidade booleana*), de modo a serem introduzidos num SAT-SOLVER. Analisando detalhadamente a técnica pode ser dividida nos seguintes passos.

Começamos por *atacar* a cifra em questão, isto é, executar uma criptoanálise algébrica à cifra, que nos vai permitir obter uma série de equações do tipo:

$$P(x) = 0 \qquad x \in GF(2^n) \qquad (1)$$

Visto o problema de *satisfabilidade booleana* verificar se uma proposição lógica

pode ou não ser satisfeita. O SAT-Solver só aceita proposições do tipo $P(x) = 1$ (verdadeira). Então precisamos de transformar equações do tipo (1) para equações do tipo (2):

$$F(\bar{x}) = 1 \qquad x \in GF(2)^n \qquad (2)$$

Para esta transformação é necessário aplicar técnicas espectrais, como indicado no capítulo 3. Estas técnicas permitem-nos passar de equações com bytes ($GF(2^n)$ para uma representação com equações booleanas ($GF(2)^n$), passando de (1) para (2), sendo que a expressão em (1) é tornada num conjunto de equações verdadeiras (2), de modo a ser transformada num problema de SAT.

Após obtida as equações finais, e calculada a forma normal conjuntiva final, passa-se a expressão para um ficheiro CNF, de modo a ser introduzida como input no SAT-Solver. Assim verificamos se o problema é satisfatível ou não.

5.2 Exemplo em pequena escala

Aplicamos então a técnica apresentada a cifras mais simples, *Toy ciphers*. Suponhamos que após a criptoanálise algébrica das *Toy Ciphesr* obtemos as seguintes equações em $GF(2^4)$, sendo que o polinómio característico do corpo é: $\beta^4+\beta+1 = 0$.

Comecemos então por a primeira cifra, com a seguinte equação.

$$0 = x^2 + x$$

Sendo esta equação do tipo 1 apresentado anteriormente, é fácil passar isto para equações lineares:

$$x = a_0 + a_1\beta + a_2\beta^2 + a_3\beta^3$$
$$x^2 = a_0 + a_1\beta^2 + a_2(\beta + 1) + a_3(\beta^2 + \beta^3)$$

Logo,

$$x + x^2 = a_2 + (a_1 + a_2)\beta + (a_1 + a_2 + a_3)\beta^2$$

Podemos transformar $P(x) = 0$ em $P(\bar{x}) = 1$ adicionamos 1 (negamos) na primeira

das três equações lineares obtidas:

$$1 + a_2 = 0$$
$$a_1 + a_2 = 0$$
$$a_1 + a_2 + a_3 = 0$$

Isto é equivalente a termos $\delta(P_1, P_2, P_3) = 1$, partindo de aqui é fácil passar estas equações para uma forma normal conjuntiva (ficheiro CNF, como no exemplo do capítulo 2), de modo a introduzir num Sat-Solver.

A segunda cifra consiste em equações do tipo

$$x + y = 0$$

Estando esta equação na forma 1

$$x = a_0 + a_1\beta + a_2\beta^2 + a_3\beta^3$$
$$y = b_0 + b_1\beta + b_2\beta^2 + b_3\beta^3$$

É fácil calcular que,

$$x + y = (a_0 + b_0) + (a_1 + b_1)\beta + (a_2 + b_2)\beta^2 + (a_3 + b_3)\beta^3$$

Negando a primeira expressão, de modo a tornar $P(\bar{x}) = 1$, obtemos as seguintes equações lineares:

$$1 + a_0 + b_0 = 0$$
$$a_1 + b_1 = 0$$
$$a_2 + b_2 = 0$$
$$a_3 + b_3 = 0$$

Apartir das equações é trivial transformalas num problema de SAT.

A ultima *Toy Cipher* consiste num cifra com equações do tipo

$$x \cdot y = 0$$

em que:

$$x = a_0 + a_1\beta + a_2\beta^2 + a_3\beta^3$$
$$y = b_0 + b_1\beta + b_2\beta^2 + b_3\beta^3$$

É fácil calcular o resultado de $x \cdot y$ com apoio de algum software matemático, neste caso foi usado o Maple® para obter a seguinte equação:

$$\begin{aligned} x \cdot y = &(a_0b_3 + a_1b_3 + a_2b_2 + a_3b_0 + a_3b_1) \\ &+ (a_1b_3 + a_3b_1 + a_0b_3 + a_3b_0 + a_3b_2 + a_2b_2 + a_2b_3)\beta \\ &+ (a_3b_2 + a_0b_1 + a_3b_3 + a_2b_3 + a_0b_0 + a_1b_0 + a_1b_1)\beta^2 \\ &+ (a_3b_3 + a_0b_2 + a_1b_2 + a_2b_0 + a_2b_1)\beta^3 \end{aligned}$$

As equações não-lineares obtidas, negando um membro da expressão:

$$\begin{aligned} 1 &= (a_0b_3 + a_1b_3 + a_2b_2 + a_3b_0 + a_3b_1) \\ 0 &= (a_1b_3 + a_3b_1 + a_0b_3 + a_3b_0 + a_3b_2 + a_2b_2 + a_2b_3) \\ 0 &= (a_3b_2 + a_0b_1 + a_3b_3 + a_2b_3 + a_0b_0 + a_1b_0 + a_1b_1) \\ 0 &= (a_3b_3 + a_0b_2 + a_1b_2 + a_2b_0 + a_2b_1) \end{aligned}$$

Sendo trivial a sua transformação para um problema de *satisfatibilidade booleana*.

Estas formas normais conjuntivas calculadas devem ser inseridas num ficheiro CNF e depois introduzidas como *input* no SAT-Solver, tendo em conta que as disjunções ($\vee$) representam xors ($\oplus$). Uma das soluções possíveis para isto é modificar o SAT-Solver de modo a aceitar a operação de XOR [35]. De modo a aumentar a eficácia do SAT é aconselhavel consultar [1].

5.3 BES

Recordemos a cifra do AES-128 enquanto cifra embebida do BES, em que cada componente (byte) é definido em $GF(2^n)$, gera o seguinte sistema de equações MQ em $\mathbb{F}$:

$$(1) \quad 0 = w_{0,(j,m)} + p_{(j,m)} + k_{0,(j,m)}$$

$$(2) \quad 0 = w_{i,(j,m)} + k_{i,(j,m)} + \sum_{(j',m')} \alpha_{(j,m),(j',m')} x_{i-1,(j',m')} \quad para\ 0 \leq i \leq 9$$

$$(3) \quad 0 = c_{(j,m)} + k_{10,(j,m)} + \sum_{(j',m')} \beta_{(j,m),(j',m')} x_{9,(j',m')}$$

$$(4) \quad 0 = x_{i,(j,m)} w_{i,(j,m)} + 1 \quad para\ 0 \leq 1 \leq 9$$

$$(5) \quad 0 = x^2_{i,(j,m)} + x_{i,(j,m+1)} \quad para\ 0 \leq 1 \leq 9$$

$$(6) \quad 0 = w^2_{i,(j,m)} + w_{i,(j,m+1)} \quad para\ 0 \leq 1 \leq 9.$$

Sabendo que $p \in B$ representa o *plaintext*, $c \in B$ representa o *ciphertext*, k_i representa a chave do round (a cifra têm 10 rounds) e x_i e w_i representam o vector estado antes e depois da inversão, respectivamente. Denotamos as componentes de x_i por $x_{i,(j,m)}$ (em que $0 \leq j \leq 15$ e $0 \leq m \leq 7$). α e β são os parámetros de entrada das matrizes M_B e M_B^* (M_B é a matriz de difusão linear e M_B^* é a matriz modificada para o último round).

Sabendo que este sistema gera 5248 equações, 7808 termos, e 2560 variavéis de estado (referentes a x e a w) e 1408 variáveis de chave (k, o que verdadeiramente se quer calcular) [26].

As equações do tipo 1, 2 e 3 são equações do tipo $x+y = 0$ para a introdução destas no SAT-Solver, devemos proceder como no exemplo em pequena escala apresentado.

As equações obtidas a partir do tipo 5 e 6 são equações lineares do tipo $x^2+x = 0$, similiares ao exemplo de pequena escala apresentado anteriormente. Procedendo de maneira igual é fácil a sua passagem para CNF, de modo a introduzir como input no SAT-Solver.

As equações do tipo 4, geram equações não lineares pois são do tipo $x \cdot y = 0$. Para actuar de modo a introduzir no SAT-Solver, devemos resolver como no exemplo em pequena escala apresentado.

Como visto anteriormente, e tratando-se de corpos de Galois $GF(2^8)$, cada uma destas equações gera 8 equações. Logo obtemos à volta 41984 equações. Obtemos também arredondadamente, sendo que cada byte são 8 bits, à volta 62464 termos,

20480 variáveis de estado, e 11264 variáveis de chave. Sendo este tipo de valores aceitáveis para um SAT-Solver.

Mostramos assim, como aplicando técnicas espectrais à criptoanálise algébrica do BES, conseguimos transformar as equações MQ obtidas num problema de *satisfatibilidade booleana*.

5.4 Considerações Finais

Nesta dissertação estudamos a importância do problema de *satisfatibilidade booleana* para a criptografia e o modo de funcionamento dos SAT-Solvers. Discutimos e analizamos também, a importância das funções booleanas para o desenho de cifras e S-Boxes. Introduzimos o conceito de *técnicas espectrais* e explicamos como este tipo de técnicas conjuntamente com um SAT-Solver benificiam a criptoanálise algébrica de cifras. Procuramos estudar o design e estrutura algébrica do AES, assim como a criptoanálise algébrica existente da cifra. Mostrou-se também que é possível transformar uma criptoanálise algébrica de uma cifra em um problema de SAT através de técnicas espectrais.

O AES, como foi referido nesta dissertação, é uma cifra desenhada com vista a ser imune a criptoanálise linear e diferencial, não tendo em conta a qualquer tipo de criptoanálise algébrica, pois o estudo neste campo era muito pouco significativo na altura da concepção da cifra. Tendo em conta que a criptoanálise algébrica está em constante desenvolvimento, consideramos que esta dissertação serve de ponte entre este tipo de técnicas e possíveis criptoanálises algébricas não abordadas nesta dissertação como por exemplo o mais recente *Big-BES* [24]. A dissertação, é também um ponto de partida para um estudo mais aprofundado das técnicas aplicadas a algumas criptoanálises já aqui referidas, como por exemplo as Bases de Gröbner ou o algoritmo de XSL.

Bibliografia

[1] G. Bard, N. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF (2) via SAT-solvers. *system*, 1:1, 2007.

[2] T. Becker, V. Weispfenning, W. Adams, and P. Loustaunau. Gröbner bases: a computational approach to commutative algebra. *Bull. Amer. Math. Soc. 33 (1996)*, 1996.

[3] E. Biham, R. Anderson, and L. Knudsen. Serpent: A new block cipher proposal. *Lecture Notes in Computer Science*, 1372:222–238, 1998.

[4] J. Buchmann, A. Pyshkin, and R. Weinmann. A Zero-dimensional Grobner Basis for AES-128. *Lecture Notes in Computer Science*, 4047:78, 2006.

[5] C. Cid, S. Murphy, and M. Robshaw. *Algebraic aspects of the advanced encryption standard*. Springer Verlag, 2006.

[6] N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. *Lecture Notes in Computer Science*, 2501:267–287, 2002.

[7] D. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*. Springer Verlag, 2007.

[8] J. Daemen and V. Rijmen. *The Design of Rijndael: AES–the Advanced Encryption Standard*. Springer, 2002.

[9] J. Daemen, V. Rijmen, and A. Proposal. Rijndael. In *Proceedings from the First Advanced Encryption Standard Candidate Conference, National Institute of Standards and Technology (NIST)*, 1998.

[10] D. De, A. Kumarasubramanian, and R. Venkatesan. Inversion attacks on secure hash functions using SAT solvers. *Lecture Notes in Computer Science*, 4501:377, 2007.

[11] P. Dunne. The complexity of Boolean networks. Volume 29 of APIC Studies in Data Processing, 1988.

[12] J. Faugere and A. Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Grobner bases. In *Advances in cryptology-CRYPTO*, volume 2729, pages 44–60. Springer, 2003.

[13] N. Ferguson, R. Schroeppel, and D. Whiting. A simple algebraic representation of Rijndael. *Lecture notes in computer science*, pages 103–111, 2001.

[14] M. Herbstritt. zchaff: Modifications and extensions. *report00188, Institut für Informatik, Universität Freiburg, July 17 2003. Thu, 17 Jul 2003 17: 11: 37 GET*, 2003.

[15] I. ISO10116. IEC 10116: Information technology-Modes of operation for an n-bit block cipher algorithm. *ISO International Standard,*, 1, 1991.

[16] E. Kleiman. *The XL and XSL attacks on Baby Rijndael.* PhD thesis, Citeseer, 2005.

[17] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1986.

[18] F. MacWilliams and N. Sloane. *The theory of error-correcting codes.* North-Holland Amsterdam, 2003.

[19] F. Massacci. Using Walk-SAT and Rel-SAT for cryptographic key search. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 290–295. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.

[20] F. Massacci and L. Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24(1):165–203, 2000.

[21] R. McEliece. *Finite fields for computer scientists and engineers*. Springer, 1987.

[22] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC press, 1997.

[23] I. Mironov and L. Zhang. Applications of SAT solvers to cryptanalysis of hash functions. *Lecture Notes in Computer Science*, 4121:102, 2006.

[24] J. Monnerat and S. Vaudenay. On some Weak Extensions of AES and BES. *Lecture notes in computer science*, pages 414–426, 2004.

[25] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference, 2001. Proceedings*, pages 530–535, 2001.

[26] S. Murphy and M. Robshaw. Essential algebraic structure within the AES. *Lecture Notes in Computer Science*, pages 1–16, 2002.

[27] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback. Report on the development of the Advanced Encryption Standard (AES). *JOURNAL OF RESEARCH-NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY*, 106(3):511–576, 2001.

[28] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback. Status report on the first round of the development of the Advanced Encryption Standard. *JOURNAL OF RESEARCH-NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY*, 104(5):435–460, 1999.

[29] K. Nyberg. Differentially uniform mappings for cryptography. *Lecture Notes in Computer Science*, 765:55–64, 1994.

[30] K. Nyberg and L. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, 1995.

[31] E. Oswald, J. Daemen, and V. Rijmen. AES-The State of the Art of Rijndael's Security, 2002.

[32] C. Papadimitriou and C. Papadimitriou. *Computational complexity*. Addison-Wesley Reading, Mass, 1994.

[33] B. Preneel. *Analysis and design of cryptographic hash functions*. Citeseer, 1993.

[34] T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM New York, NY, USA, 1978.

[35] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, page 257. Springer, 2009.

[36] A. Standard. FIPS 197. *National Institute of Standards and Technology*, 2001.

[37] D. Standard. Federal Information Processing Standards Publication 46. *National Bureau of Standards, US Department of Commerce*, 1977.

[38] I. Toli and A. Zanoni. Looking inside AES and BES. In *IFIP TCS*, pages 23–36, 2004.

[39] J. Valença. Técnicas Criptográficas, 2008.

[40] R. Weinmann. Evaluating algebraic attacks on the AES. *Diplom thesis, Technische Universit Darmstadt*, 2003.