Técnicas Criptográficas

José Manuel E. Valença *

8 de Junho de 2009

*Departamento de Informática, Universidade do Minho, Campus de Gualtar Braga

3.Funções Booleanas

O projecto e segurança de muitas técnicas criptográficas estão ligadas ao estudo das funções da forma $f : \mathbb{B}^n \longrightarrow \mathbb{B}^n$ ou $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ que tomam como argumento uma sequência finita de *bits* de comprimento fixo e devolvem uma sequência do mesmo tamanho ou então um simples bit.

Das várias representações possíveis para tais funções escolhemos algumas que são particularmente importantes:

1. Funções booleanas de n variáveis booleanas

$$f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$$

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

$$f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)$$

3.1 Funções de argumento $\mathrm{GF}(2)^n$

Nas funções booleanas $f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$ os argumentos x são vectores de n bits; as operações básicas sobre os argumentos são:

1. Selecção $\mathrm{sel} : \mathbb{Z}_n \times \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$,
$$\mathrm{sel}(i, x) = x_i$$
i.e. dado $i \in \mathbb{Z}_n$, seleciona a componente de ordem i do vector x.
2. Operações binárias $\oplus$, $* : \mathrm{GF}(2)^n \times \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^n$
são as duas funções binárias que aplicam *xor* e *and* bit a bit.

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

A noção de *índice* pode ser gereralizada; podemos tomar como índice um qualquer conjunto de inteiros $i \in \mathbb{Z}_n$; por exemplo, se tivermos $n = 8$, um índice será, por exemplo, $\{0, 3, 7\}$. O valor booleano seleccionado por este índice será

$$x_{0,3,7} \;\doteq\; x_0 \cdot x_3 \cdot x_7$$

Genéricamente um **índice** para funções boolenas de n argumentos booleanos é um sub-conjunto $u \subseteq \mathbb{Z}_n$; o conjunto desses índices representa-se por $\mathbb{U}_n$ e identifica-se com $\wp(\mathbb{Z}_n)$.

A função selecção $\mathsf{sel} : \mathbb{U}_n \times \mathsf{GF}(2)^n \longrightarrow \mathsf{GF}(2)$ generaliza-se facilmente

$$\mathsf{sel}(u, x) = \Pi_{i \in u}\, x_i \qquad , \qquad \mathsf{sel}(\emptyset, x) = 1 \tag{56}$$

150 DEFINIÇÃO
Para $u \in \mathbb{U}_n$ *designa-se por* x_u *o monómio* $\mathsf{sel}(u, x)$. *Designa-se por* $[x]_u$ *o polinómio* $\mathsf{sel}(u, x) \cdot \Pi_{i \notin u}(1 + x_i)$.

Por exemplo, se for $n = 4$ e $u = \{0, 2\}$, temos

$$x_u \;=\; x_0 \cdot x_2 \qquad \text{e} \qquad [x]_u = x_0 \cdot (1 + x_1) \cdot x_2 \cdot (1 + x_3)$$

Toda a função booleana de domínio $\mathsf{GF}(2)^n$ pode ser escrita como um polinómio a n variáveis $x_0, x_1, \ldots, x_{n-1}$ dado por

$$f(x) \;=\; \sum_{u \in \mathbb{U}_n} a(u)\, x_u \tag{57}$$

para uma função $a : \mathbb{U}_n \longrightarrow \mathsf{GF}(2)$.

Esta é a chamada **forma normal algébrica** de f e é completamente determinada pela função a dita *função de índices* ou *espectro de índices* ou, simplesmente, *espectro* se não existirem ambiguidades [34].

Uma forma equivalente de representar a mesma função consiste em considerar o conjunto $\mathcal{A} \subseteq \mathbb{U}_n$ de índices u para os quais $a(u) = 1$; essencialmente $\mathcal{A}$ é o conjunto que tem a função de coeficientes como função característica.

Nesse caso (57) escreve-se

$$f(x) = \sum_{u \in \mathcal{A}} x_u \tag{58}$$

Daqui se deduz que o número total de funções booleanas de argumento $\mathrm{GF}(2)^n$ é 2^{2^n}.

O **grau** de f é definido como $(\max_{u \in \mathcal{A}} |u|)$: i.e. a maior cardinalidade de um índice em $\mathcal{A}$.

Se o grau é 0 ou 1 a função diz-se **afim**.

Claramente que o número total de funções afins é 2^{n+1} metade das quais são **lineares**: i.e. funções f tais que $a(\emptyset) = 0$ ou, equivalentemente, $\emptyset \notin \mathcal{A}$.

[34] Veremos adiante que é importante definir um outra forma de espectro de funções booleanas: o espectro de *Walsh-Hadamard*.

EXEMPLO 21: Considere-se as funções booleanas $f : \mathbb{B}^4 \longrightarrow \mathbb{B}$ com 4 argumentos booleanos. Um exemplo de uma função na forma normal algébrica será

$$f(x) = 1 + x_0 + x_2 + x_1\,x_2 + x_1\,x_3 + x_0\,x_1\,x_3$$

O conjunto dos índices que correspondem a termos não nulos é

$$\mathcal{A} \;=\; \{\emptyset \;,\; \{0\} \;,\; \{2\} \;,\; \{1,2\} \;,\; \{1,3\} \;,\; \{0,1,3\}\}$$

O maior deles tem 3 elementos; por isso f tem grau 3.

Esta função não é afim. Um exemplo de função afim será

$$g(x) = 1 + x_0 + x_2$$

que não é uma função linear devido à presença da constante 1. Um exemplo de função linear será

$$h(x) \;=\; x_0 + x_2$$

Em termos de espectros, o de g é $\{\emptyset \;,\; \{0\} \;,\; \{2\}\}$ enquanto que o de h é $\{\{0\} \;,\; \{2\}\}$.

O **símbolo de Kronecker** de ordem n é a função $\delta : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$ que verifica $\delta(x) = 1$ se e só se $x = (0, 0, \cdots, 0)$.

151 FACTO

Para todo $x \in \mathsf{GF}(2)^n$ *verifica-se*

$$\delta(x) = (1 + x_0) \cdot (1 + x_1) \cdot (1 + x_2) \cdots (1 + x_{n-1}) = [x]_\emptyset$$

Para quaisquer $X, Y \subseteq \mathbb{U}_n$ define-se a sua **união disjunta** por

$$X \uplus Y \doteq (X \setminus Y) \cup (Y \setminus X)$$

e a sua **convolução** por

$$X \circledast Y \doteq \biguplus_{x \in X} \{ x \cup y \mid y \in Y \}$$

152 FACTO

Sejam $\mathcal{A}, \mathcal{B} \subseteq \mathbb{U}_n$ *os espectros de funções* f, g *respectivamente; i.e.*

$$f(x) = \sum_{u \in \mathcal{A}} x_u \qquad g(x) = \sum_{v \in \mathcal{B}} x_v$$

(i) Se f *é uma função constante então: se for* $f(x) = 1$, *o seu espectro é* $\{\emptyset\}$ *e, se for* $f(x) = 0$, *o seu espectro é* $\{\ \}$.

(ii) *Se* $\mathcal{A} = \mathbb{U}_n$ *então* f *coincide com o símbolo de Kronecker* δ. *Se* $\mathcal{A} = \mathbb{U}_n \setminus \emptyset$ *então* $f = 1 + \delta$ *(i.e.* $f(x) = 1$ *se e só se* $x \neq 0$*).*

(iii) $(f + g)$ *tem espectro* $\mathcal{A} \uplus \mathcal{B}$ *e* $(f \cdot g)$ *tem espectro* $\mathcal{A} \pitchfork \mathcal{B}$.

$$(f + g)(x) = \sum_{u \in \mathcal{A} \uplus \mathcal{B}} x_u \qquad (f \cdot g)(x) = \sum_{u \in \mathcal{A} \pitchfork \mathcal{B}} x_u$$

Corolário: $1 + f$ *(a negação de* f*) tem espectro* $\mathcal{A} \uplus \emptyset$.

(iv) *Se* $g(x) = f(z * x)$, *para algum* $z \in \mathrm{GF}(2)^n$, *então*

$$\mathcal{B} = \{\, u \in \mathcal{A} \mid z_u = 1 \,\}$$

Comentários: O uso de tratamentos espectrais têm longa tradiação em Engenharia. Essencialmente procura-se uma representação alternativa para funções de tal forma que o estudo dessas funções possa ser feito (de maneira mais simples) na representação espectral.

Tradicionalmente procura-se analisar as relações entre propriedades no domínio das funções e propriedades nos domínios dos espectros e formas simples de determinar umas e outras.

Este resultado e os que se seguem vêm dentro dessa tradição. São apresentados várias formas particulares de construir funções e é analisada a forma equivalente no domínio dos espectros.

(i) As funções constantes são polinómioas de grau zero; o espectro de $f(x) = 1$ deriva directamente da definição de $x_\emptyset$. De forma semelhante temos o polinómio vazio que origina $f(x) = 0$.

(ii) Se tivermos $x = (0, 0, \ldots, 0)$ então temos $x_u = 1$ se e só se $u = \emptyset$. Sendo $f(x) = \sum_{x \in \mathbb{U}_n} x_u$ teremos claramente $f(x) = 1$.
Se for $x \neq (0, 0, \ldots, 0)$ então seja $\bar{x} \doteq \{ i \mid x_i = 1 \}$; claramente $x_u = 1$ se e só se $u \subseteq \bar{x}$. O número de índices u tais que $x_u = 1$ é, assim, um número par (é dado por $2^{|\bar{x}|}$); consequentemente $f(x) = \sum_{x \in \mathbb{U}_n} x_u = 0$ já que é a a soma de um número par de elementos não nulos.

(iii) Os espectros de $(f + g)$ e $(f \cdot g)$ resultam directamente da expansão destas duas expressões substituindo $f(x)$ e $g(x)$ pelas respectivas somas.

(iv) Sendo $f(x) = \sum_{u \in \mathcal{A}} x_u$ então

$$g(x) = f(z * x) = \sum_{u \in \mathcal{A}} (z * x)_u = \sum_{u \in \mathcal{A}} z_u \cdot x_u = \sum_{u \in \mathcal{A} \wedge z_u = 1} x_u$$

O conjunto $f^{-1}(1) = \{ x \mid f(x) = 1 \}$ chama-se **suporte** de f e é representado por $\mathrm{supp}(f)$.
Chama-se **peso** de f (representado por $\mathrm{wt}(f)$) à cardinalidade desse conjunto – $\mathrm{wt}(f) = |f^{-1}(1)|$.
A função é **balanceada** quando, para metade dos seus argumentos, o valor for 1; isto é, $\mathrm{wt}(f) = 2^{n-1}$.

153 FACTO
Para um qualquer $z \in \mathrm{GF}(2)^n$, seja $\delta^{(z)}(x) \doteq \delta(x \oplus z)$ (i.e., $\delta^{(z)}(x) = 1$ se e só se $x = z$). Então:

(i) $\delta^{(z)} \cdot \delta^{(w)} = \delta(z \oplus w) \cdot \delta^{(z)}$

(ii) Qualquer função booleana f pode ser representada por

$$f = \sum_{z \in \mathsf{supp}(f)} \delta^{(z)} \tag{59}$$

(iii) O espectro de $\delta^{(z)}$ é o conjunto $\uparrow \bar{z} \doteq \{ u \mid \bar{z} \subseteq u \}$ sendo $\bar{z} \doteq \{ i \mid z_i = 1 \}$ (i.e., $\bar{z}$ é o maior índice u tal que $z_u = 1$).

(iv) Para todo $z \in \mathsf{GF}(2)^n$ tem-se $[x]_{\bar{z}} = \delta^{(z)}(x)$.

Notas O primeiro resultado traduz apenas propriedades de $\delta^{(z)}$ que resultam directamente da definição.

O segundo resultado é consequência imediata do primeiro e prova-se considerando a expansão de $f(x)$ nos casos em que $x \in \mathsf{supp}(f)$ e $x \notin \mathsf{supp}(f)$. Tem muita importância computacional quando o suporte da função é tem baixa cardinalidade (f tem pouco peso) e pode ser calculado facilmente. Nestas circunstâncias (59) é uma forma conveniente de representar a função.

O resultado (iii) é importante nomeadamente porque sugere um ataque a uma função boolena $\mathcal{Q} : \mathbb{B}^n \longrightarrow \mathbb{B}$ como a procura de um $k \in \mathbb{B}^n$ tal que $\mathcal{Q}(k) = 1$.

Suponhamos que tínhamos a certeza que existia só um k nestas condições mas que esse valor era, obviamente, desconhecido. Isso é equivalente a dizer que $\mathcal{Q}$ tem a forma $\delta^{(k)}$, para um k desconhecido. Suponhamos também (e isto é uma suposição muito forte) que existia um modo qualquer de determinar o espectro $\mathsf{Sp}(\mathcal{Q})$ de $\mathcal{Q}$.

O resultado anterior diz-nos que o espectro $\mathsf{Sp}(\mathcal{Q})$ é $\uparrow\bar{k}$. Portanto é um conjunto que tem um limite inferior $\bar{k}$ que pode ser calculado como

$$\bar{k} \;=\; \bigcap_{u\in \mathsf{Sp}(\mathcal{Q})} u$$

Supondo finalmente que esse limite era computável; então fica determinado $\bar{k}$ e, consequentemente, fica determinado k.

Para o provar considere-se a função f que tem $\uparrow\bar{z}$ por espectro. Notando que $x_u = 1$ se e só se $u \subseteq \bar{x}$, tem-se

$$f(x) \;=\; \sum_{u\supseteq \bar{z}} x_u \;=\; \sum_{u\supseteq \bar{z}\wedge u\subseteq \bar{x}} 1$$

Se $x = z$ existe apenas um índice u que satisfaz a condição $u \supseteq \bar{z} \wedge u \subseteq \bar{x}$ (nomeadamente $u = \bar{x} = \bar{z}$). Neste caso o somatório tem uma só parcela e o resultado é 1. Se $x \neq z$ então o conjunto de todos os índices u que satisfazem a condição ou é vazio (quando $\bar{x} \subset \bar{z}$) ou tem um número par de elementos; em qualquer dos casos o somatório é zero. Donde $f(x) = 1$ se e só se $x = z$; portanto, $f \equiv \delta^{(z)}$.

154 TEOREMA

O espectro $\mathsf{Sp}(f)$ *de uma função booleana* f *verifica*

$$\mathsf{Sp}(f) \;=\; \biguplus_{z\in \mathsf{supp}(f)} \uparrow\bar{z} \tag{60}$$

Seja $f^{(z)}$ *a função definida por* $f^{(z)}(x) = f(z \oplus x)$ *. Então*

$$\mathsf{Sp}(f^{(z)}) \;=\; \biguplus_{y \in \mathsf{supp}(f)} \uparrow(\bar{z} \uplus \bar{y}) \tag{61}$$

A prova do primeiro resultado é consequência imediata de (59) e do facto 152.

Para provar o segundo basta notar que

$$f^{(z)}(x) = \; f(z \oplus x) = \sum_{y \in \mathsf{supp}(f)} \delta^{(y)}(z \oplus x) = \sum_{y \in \mathsf{supp}(f)} \delta^{(y \oplus z)}(x)$$

Se atender-mos que $\overline{(y \oplus z)} = \{\, i \mid y_i + z_i = 1 \,\} = \bar{y} \uplus \bar{z}$ e que o espectro de $\delta^{(y \oplus z)}$ é $\uparrow\overline{y \oplus z} = \uparrow(\bar{y} \uplus \bar{z})$ obtém-se a expressão pretendida para o espectro.

A representação do espectro em termos do suporte da função f não é muito conveniente já que é difícil, quase sempre, calcular esse suporte. Por isso é necessária uma abordagem alternativa ao cálculo de $\mathsf{Sp}(f)$ e, para isso, é necessário introduzir uma representação de espectros inspirada nos *Binary Decision Diagrams* ou BDD's e uma interpretação desses diagrama análoga à usada no método de *Davis-Purtnam*.

Começa-mos por um exemplo.

EXEMPLO 22: Consider-se a função booleana em $\mathrm{GF}(2)^4$

$$f(x_0, x_1, x_2, x_3) = 1 + x_0 \cdot x_1 + x_0 \cdot x_2 + x_1 \cdot x_2 \cdot x_3$$

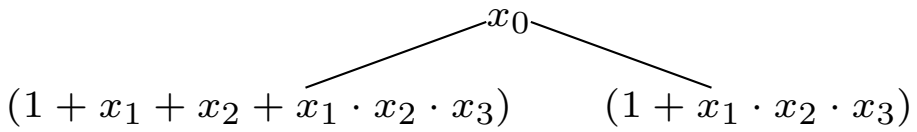
Pode-se sempre escrever

$$f(x_0, x_1, x_2, x_3) \;=\; x_0 \cdot f(1, x_1, x_2, x_3) + (1 + x_0) \cdot f(0, x_1, x_2, x_3)$$

ou seja, com alguma manipulação

$$= x_0 \cdot (1 + x_1 + x_2 + x_1 \cdot x_2 \cdot x_3) + (1 + x_0) \cdot (1 + x_1 \cdot x_2 \cdot x_3)$$

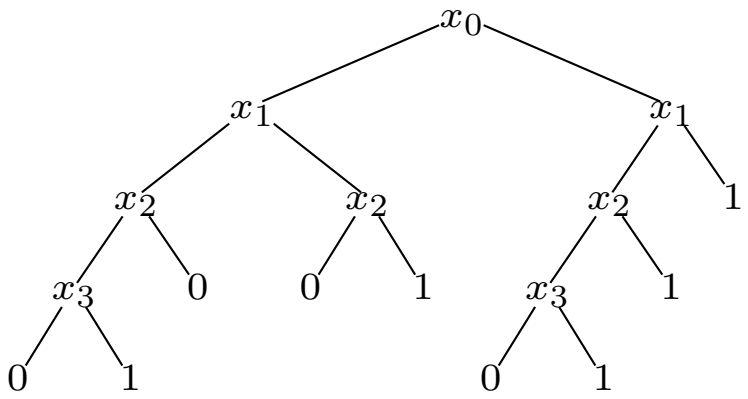
O que sugere uma representação arbórea



Repetindo o processo para as restantes variáveis, vemos que

$$\begin{aligned}(1 + x_1 + x_2 + x_1 \cdot x_2 \cdot x_3) &= x_1 \cdot x_2 \cdot (1 + x_3) + (1 + x_1) \cdot (1 + x_2) \\ (1 + x_1 \cdot x_2 \cdot x_3) &= x_1 \cdot (x_2 \cdot (1 + x_3) + (1 + x_2) \cdot 1) + (1 + x_1) \cdot 1\end{aligned}$$

O que sugere a seguinte árvore



Note-se que os percursos iniciados na raiz determinam valores de x e os respectivos valores $f(x)$ consoante a folha onde os percursos terminam. Os percursos são determinados pelas regras muito simples: $x_i = 1$ significa "*virar à esquerda* no nodo x_i", enquanto $x_i = 0$ significa "*virar à direita*.no nodo x_i".

Nesta árvore, por exemplo, os seguintes percursos terminam no valor 0

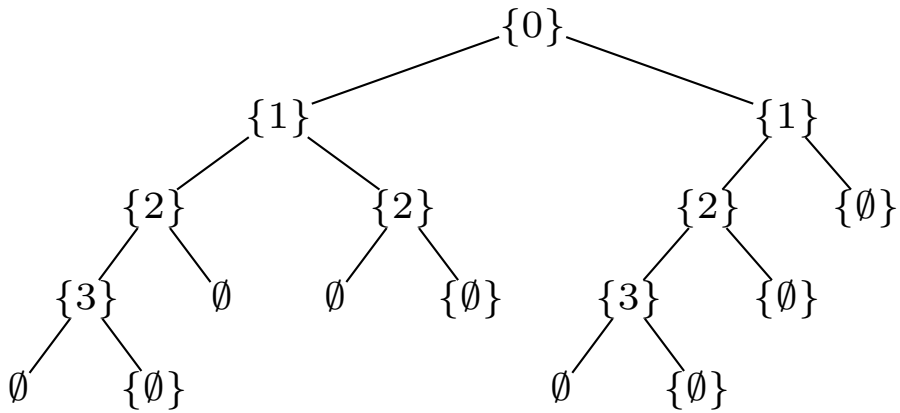
$$\{x_0 = 1\ ,\ x_1 = 0\ ,\ x_2 = 1\ ,\ x_3 = ?\} \quad , \quad \{x_0 = 1\ ,\ x_1 = 1\ ,\ x_2 = 0\ ,\ x_3 = ?\}$$
$$\{x_0 = 1\ ,\ x_1 = 1\ ,\ x_2 = 1\ ,\ x_3 = 1\} \quad , \quad \{x_0 = 0\ ,\ x_1 = 1\ ,\ x_2 = 1\ ,\ x_3 = 1\}$$

Todos os restantes percursos terminam no valor 1. Isto diz-nos que

$$f(1,0,1,0) = f(1,0,1,1) = f(1,1,0,0) =$$
$$= f(1,1,0,1) = f(1,1,1,1) = f(0,1,1,1) = 0$$

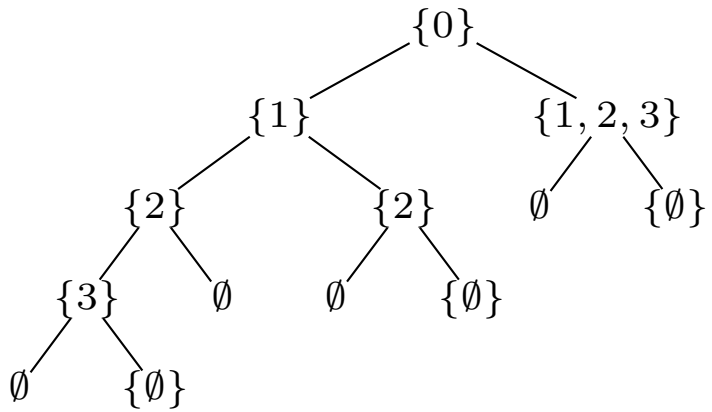
Uma árvore equivalente à anterior pode ser construída colocando nos nodos e nas folhas os espectros das expressões

que ocorrem na árvore anterior.



Admitindo que os nodos podem ter quaisquer índices (e não apenas índices singulares) a árvore pode-se simplificar

para



Esta árvore permite, agora, reconstruir o espectro da função inicial. De facto é fácil de verificar:

(i) Se a árvore for uma folha o espectro é o que a folha indica: $\emptyset$ ou $\{\emptyset\}$.

(ii) Se a árvore for um triplo $\alpha = \langle \sigma \ , \ \alpha^+ \ , \ \alpha^- \rangle$, o seu espectro $\mathcal{A}$ é

$$\mathcal{A} \ = \ \mathcal{A}^- \uplus \{\sigma\} \barwedge (\mathcal{A}^+ \uplus \mathcal{A}^-)$$

sendo $\mathcal{A}^+, \mathcal{A}^-$ os espectros representados pelas sub-árvores α^+ e α^-.

Para sistematizar esta construção seja $\mathbb{U}_{k,n} \doteq \wp(\mathbb{Z}_n \setminus \mathbb{Z}_k)$; isto é, o conjunto de todos os índices formado por elementos $k \leq i < n$.

155 DEFINIÇÃO
[35] *Dado um polinómio reduzido (sem monómios repetidos)* $f = \sum_{u \in \mathcal{A}} x_u$ *com* $\mathcal{A} \subseteq \mathbb{U}_{k,n}$ *o* **fraccionamento** *de* f *em* $\mathbb{U}_{k,n}$ *é:*

1. *Se* f *é uma função constante, o fraccionamento coincide com a própria função.*
2. *Se* f *não é constante (tem grau maior do que* 0*), sejam:*
   (i) $f^-(x_{k+1}, \cdots, x_{n-1})$ *o polinómio que se obtém eliminando de* f *todos os monómios que contenham* x_k*; seja* α^- *o seu fraccionamento em* $\mathbb{U}_{k+1,n}$ *determinado por este processo.*
   (ii) $f^+(x_{k+1}, \cdots, x_{n-1})$ *o polinómio que se obtém de* f *eliminando* x_k *de todos os seus monómios e reduzindo o resultado final através da eliminação de pares de monómios iguais; seja* α^+ *o seu fraccionamento em* $\mathbb{U}_{k+1,n}$ *.*

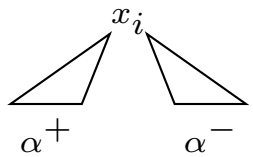
   Se $f^- = f^+$*, o fraccionamento* α *de* f *coincide com* $\alpha^+ = \alpha^-$*; em caso contrário, é o triplo* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *.*

   A **ordem** *do fraccionamento é* 0 *se* f *for constante ou, em caso contrário, é* $(n - i)$ *sendo* i *o índice da variável que constitui o primeiro elemento deste triplo.*

Esta definição sugere imediatamente uma representação arbórea para o fraccionamento de uma função cujo espectro está contido em $\mathbb{U}_{k,n}$. As funções de grau zero serão as folhas da árvore. As funções não constantes têm fraccionamentos que são triplos $\langle x_i \ , \ \alpha^+ \ , \ \alpha^- \rangle$ (com

[35] Baseada na noção de fraccionamento ("*split*")de Davis-Putnam.

$i \geq k$) formados por uma variável e dois outros fraccionamentos.

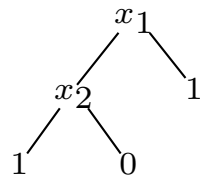


Se a função não for constante existe pelo menos um monómio com uma ou mais variáveis. Não é necessário que contenha a variável x_k; por exemplo, para $k = 0$ e $n = 3$, considere-se a função $f(x_0, x_1, x_2) = 1 + x_1 + x_1 \cdot x_2$.

$f(x_0, x_1, x_2)$ tem 3 monómios nenhum dos quais contém x_0. O cálculo de f^+ e de f^-, com o fraccionamento feito em $\mathbb{U}_{0,3}$, não modifica f uma vez que não contém x_0; teremos, assim, $f = f^+ = f^-$.

Porém, quando se efectua o fraccionamento em $\mathbb{U}_{1,3}$, tem-se $f^- = 1$ e $f^+ = 1 + 1 + x_2$ que, após redução, se simplifica em x_2.

O fraccionamento final de f está representado ao lado e, como vemos, tem ordem 2.



156 TEOREMA

Nas condições da definição anterior.

1. *Os polinómios f^+ e f^- estão reduzidos.*

2. *A condição* $f^+ = f^-$ *verifica-se se e só se* f *não contém* x_k *em nenhum dos seus monómios. Neste caso tem-se* $f^+ = f^- = f$.
3. *Sendo* $\mathcal{A}^+, \mathcal{A}^- \in \mathbb{U}_{k+1,n}$ *os espectros de* f^+ *e* f^- *então*

$$\mathcal{A}^- = \{\, u \mid u \in \mathcal{A} \ \&\ k \notin u \,\}$$

$$\mathcal{A}^+ = \{\, u \setminus \{k\} \mid u \in \mathcal{A} \,\}$$

$$\mathcal{A} = \mathcal{A}^- \uplus (\mathcal{A}^+ \uplus \mathcal{A}^-) \uplus \{\{k\}\}$$

4. *Verifica-se*

$$f^-(x_{k+1}, \cdots, x_{n-1}) = f(0, x_{k+1}, \cdots, x_{n-1})$$

$$f^+(x_{k+1}, \cdots, x_{n-1}) = f(1, x_{k+1}, \cdots, x_{n-1})$$

$$f = x_k \cdot f^+ + (1 + x_k) \cdot f^-$$

Sejam f, g duas funções booleanas e α, β os respectivos fraccionamentos em algum $\mathbb{U}_{k,n}$. Representemos por $(\alpha + \beta)$, $(\alpha \cdot \beta)$, $\alpha^{(z)}$ os fraccionamentos, respectivamente, das funções $(f + g)$, $(f \cdot g)$, $f^{(z)}$.

157 FACTO
Verifica-se:

1. $(\alpha + 0) = (\alpha \cdot 1) = \alpha$, $(\alpha \cdot 0) = 0$, $0^{(z)} = 0$, $1^{(z)} = 1$, $\alpha \cdot \alpha = \alpha$ *e* $\alpha + \alpha = 0$.
2. *Redução:* $\langle x \ , \ \alpha \ , \ \alpha \rangle$ *simplifica em* α .
3. *Se for* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *e* β *é um fracionamento de ordem inferior à de* α*, então*

$$(\alpha + \beta) = \quad \langle x_k \ , \ \alpha^+ + \beta \ , \ \alpha^- + \beta \rangle$$
$$(\alpha \cdot \beta) = \quad \langle x_k \ , \ \alpha^+ \cdot \beta \ , \ \alpha^- \cdot \beta \rangle$$

4. *Se* α *e* β *têm a mesma ordem e se for* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *e* $\beta = \langle x_k \ , \ \beta^+ \ , \ \beta^- \rangle$ *, então*

$$(\alpha + \beta) = \quad \langle x_k \ , \ \alpha^+ + \beta^+ \ , \ \alpha^- + \beta^- \rangle$$
$$(\alpha \cdot \beta) = \quad \langle x_k \ , \ \alpha^+ \cdot \beta^+ \ , \ \alpha^- \cdot \beta^- \rangle$$

5. *Se for* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *, então*

$$\alpha^{(z)} = \begin{cases} \langle x_k \ , \ (\alpha^+)^{(z)} \ , \ (\alpha^-)^{(z)} \rangle & \text{se} \quad z_k = 0 \\ \langle x_k \ , \ (\alpha^-)^{(z)} \ , \ (\alpha^+)^{(z)} \rangle & \text{se} \quad z_k = 1 \end{cases}$$

□

Frequentemente as funções boolenas aparecem agrupadas em vectores. Uma função booleana vectorial $\mathbf{S} : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^n$ pode ser descrita por um vector de n funções booleanas escalares

$$\mathbf{S}(x_1, x_2, \cdots, x_n) \quad = \quad \begin{bmatrix} h_1(x_1, x_2, \cdots, x_n) \\ h_2(x_1, x_2, \cdots, x_n) \\ \cdots \\ h_n(x_1, x_2, \cdots, x_n) \end{bmatrix}$$

Na terminologia criptográfica, uma tal função é designada por uma $n \times n$ **S-Box**.

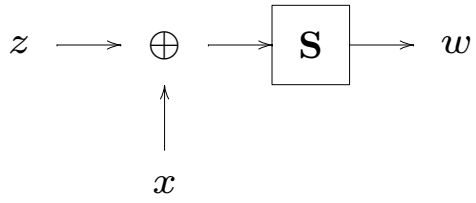
Facilmente se generaliza o conceito para S-Boxes não quadradas: uma $n \times m$ **S-Box** é uma função $\mathbf{S} : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^m$ definida por um vector de m componentes em que, cada uma, é uma função $h_i : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$, com $i \in 0..m-1$.

EXEMPLO 23: As três funções boolenas

$$\begin{bmatrix} h_0(x) = & 1 + x_0 \cdot x_1 \\ h_1(x) = & x_0 \cdot x_2 \cdot (1 + x_1) \\ h_2(x) = & 1 + x_0 + x_1 + x_2 \end{bmatrix}$$

definem uma SBox quadrada 3×3.

O exemplo mais corrente desta representação pode ser descrito pela figura seguinte



(62)

Pode-se ver x como uma chave, z como o texto de uma mensagem a cifrar e w como o criptograma resultante.

Problemas directos:

(I) determinar se existe algum valor de x que seja solução da equação

$$\mathbf{S}(z \oplus x) = w \tag{63}$$

(II) Caso exista gerar aleatoriamente **uma** solução

(III) Caso existam, enumerar **todas** as soluções.

O problema de tipo **I** procura saber, apenas, se existe uma chave, o problema de tipo **II** procura descobrir uma chave e o problema de tipo **III** procura enumerar todas as chaves.

EXEMPLO 24: Recuperemos a **SBox** 3×3 do exemplo 23 e suponhamos o seguinte par *entrada-saída*

$$z = (1, 0, 1) \qquad w = (0, 1, 0)$$

A equação que resulta de (65) (com $w_0 = 0, w_1 = 1, w_2 = 0$) será

$$h_0(z \oplus x) \cdot (1 + h_1(z \oplus x)) \cdot h_2(z \oplus x) = 1$$

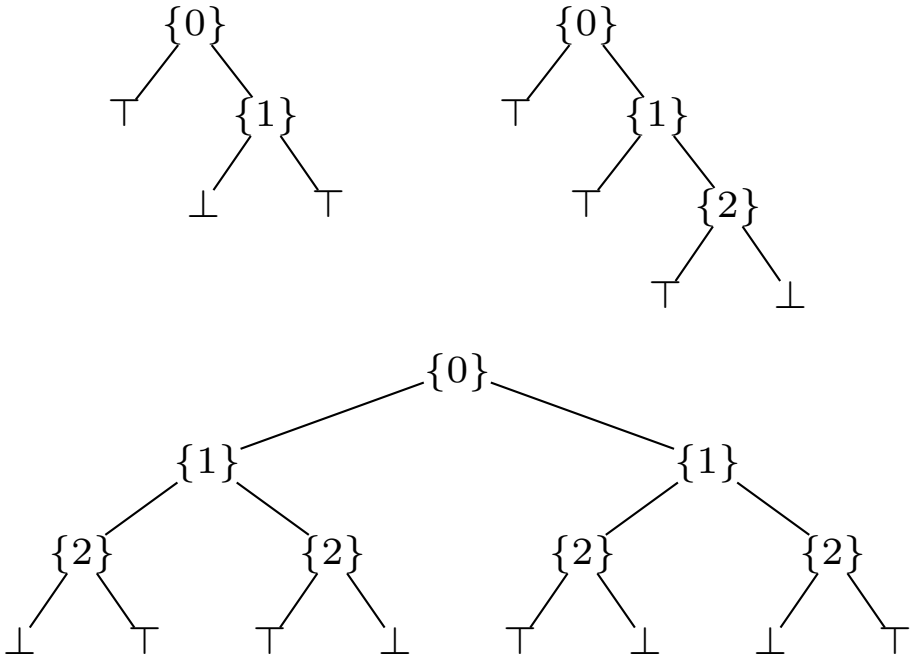
expandindo

$$(1 + (1 + x_0) \cdot x_1) \cdot (1 + (1 + x_0) \cdot (1 + x_1) \cdot (1 + x_2)) \cdot$$
$$\cdot (1 + (1 + x_0) + x_1 + (1 + x_2)) = 1$$

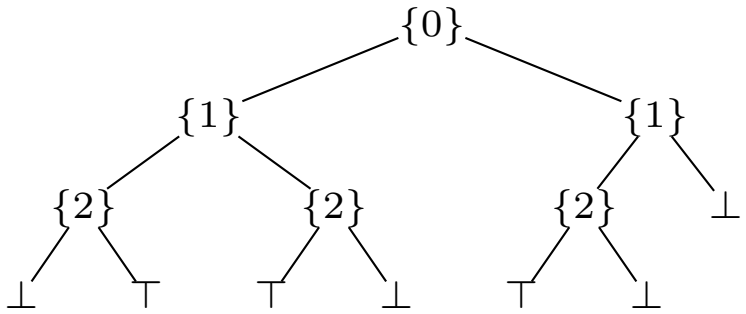
simplificando

$$(1 + x_1 + x_0 \cdot x_1) \cdot (1 + (1 + x_0) \cdot (1 + x_1) \cdot (1 + x_2)) \cdot (1 + x_0 + x_1 + x_2) = 1$$

Os espectros dos três factores nesta equação serão (abrev. $\top \equiv \{\emptyset\}$, $\bot \equiv \emptyset$),



Usando as regras no facto 157 o espectro resultante será



(64)

Esta árvore permite resolver os problemas (I), (II) e (III), neste caso, usando o resultado seguinte

158 FACTO

Seja $f : \mathrm{GF}(2) \to \mathrm{GF}(2)$ *uma função booleana e* α *o seu fraccionamento em* $\mathbb{U}_n$ *reduzido (não contém componentes da forma* $\langle u\ ,\ \beta\ ,\ \beta\rangle$ *) e construído segundo as regras do facto 157. Então:*

1. $\mathrm{supp}(f) = \emptyset$ *se e só se* $\alpha \equiv \bot$.
2. $x \in \mathrm{supp}(f)$ *se e só determina um caminho válido em* α *de acordo com as seguintes regras:*
   (a) *Numa folha* $\top$, x *determina o caminho válido vazio* ε *; não existe caminho válido numa folha* $\bot$.
   (b) *O caminho válido determinado por* x *em* $\langle u\ ,\ \alpha^+\ ,\ \alpha^-\rangle$ *(caso exista) é a sequência* $u^s\,\omega$, *em que* $s = +$, *se for* $x_u = 1$, *e* $s = -$, *se for* $x_u = 0$, *e* ω *é o caminho válido determinado por* x *em* α^s .

Notas: O que este resultado nos diz é que, basicamente, o fraccionamento α é uma representação do conjunto $\text{supp}(f)$ que é computacionalmente conveniente: as regras fornecem um algoritmo para determinar se o conjunto é ou não vazio e, caso não seja vazio, o valor lógico da relação $x \in \text{supp}(f)$.

Conhecido o fraccionamento α se se procurar soluções para os problemas básicos, teremos:

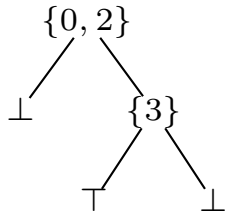
1. *Problema de tipo **I**:* saber se $\text{kwd}(f) = \emptyset$ reduz-se a saber se $\alpha = \bot$.
2. *Problema de tipo **II**:* se $\alpha \neq \bot$ encontrar um elemento arbitrário $x \in \text{kwd}(f)$ resolve-se gerando aleatoriamente um caminho válido em α. Isto significa criar um caminho que se inicie na raiz de α e siga, com igual probabilidade, por qualquer dos seus sub-fraccionamentos distintos de $\bot$.
3. *Problema de tipo **III**:* gerar todo o conjunto $\text{kwd}(f)$ é equivalente a gerar todos os caminhos válidos em α. Computacionalmente, devido ao resultado anterior, não existe distinção entre α e o suporte de f; por isso, gerar $\text{kwd}(f)$ é, essencialmente, construir o fraccionamento α.

É importante notar que diferentes valores de x podem determinar o mesmo caminho ω em α; isto acontece sempre que a árvore não está "cheia", i.e., quando o caminho não contém todos os possíveis índices $i \in 0..n-1$.

Tome-se, como exemplo, o seguinte fraccionamento em $\mathbb{U}_4$ (índices $0..3$) cujo o único caminho válido é

$$\{0,2\}^- \ \{3\}^+$$

Isto significa que qualquer x que pertença ao suporte da função tem de verificar $x_0 = x_2 = 0$ e $x_3 = 1$.



Note-se que o índice 1 não ocorre no caminho; isto significa que a componente x_1 não está fixa a nenhum dos valores 0 ou 1; representemos este facto pela relação $x_1 = ?$. O "pseudo- valor" $?$ é conhecido por *"do not care"* e, com esta notação, pode-se escrever o elemento x do suporte como $x = (0, ?, 0, 1)$.

Existe um outro "pseudo-valor" importante, representado pelo símbolo !, que indica que uma variável booleana está, simultaneamente, obrigada a ter um valor 0 e obrigada a ter um valor 1. Exemplo, $x = (!, 1, !, 1)$ indicaria que todas as componentes são fixas em 1 e, adicionamento, a primeira e terceira estão fixas em 0.

Numa máquina determinística típica isto é impossível e isso conduziria, naturalmente, a um resultado não-válido para uma computação que produzisse este pseudo-valor como resultado parcial. No entanto é possível ter modelos da computação onde um resultado ! seja perfeitamente válido; por exemplo, em modelos de computação quântica ou, genericamente, computação não-determinística.

Escrita de outro modo, a equação (63) é $\delta(w \oplus \mathbf{S}(z \oplus x)) = 1$, ou, equivalentemente,

$$\delta^{(w)}(\mathbf{S}^{(z)}(x)) = 1 \qquad \text{ou} \qquad [h^{(z)}]_{\bar{w}} = 1 \tag{65}$$

Podemos definir uma função boolena

$$\mathcal{Q}_{z,w}(x) \doteq \delta^{(w)}(\mathbf{S}^{(z)}(x)) \tag{66}$$

e a solução de um problema directo (procurar a chave x) como um ataque a $\mathcal{Q}_{z,w}$.

Uma computação tratável φ que produza uma solução para um problema directo do tipo **II** designa-se por **ataque directo** ao triplo $\langle \mathbf{S}, z, w \rangle$. A entropia de $\mathcal{Q}_{z,w}$ extendida aos ataques directos

$$\mathcal{H}(\mathcal{Q}_{z,w}) = \min_{\varphi} \mathcal{H}(\varphi) - \log_2 \left\{ \mathcal{Q}_{z,w} \right\}_{\varphi}$$

é a **entropia directa** de $\langle \mathbf{S}, z, w \rangle$.

Nota Recordemos que $\left\{ \mathcal{Q}_{z,w} \right\}_{\varphi}$ representa a probabilidade da computação φ produzir um resultado x tal que $\mathcal{Q}_{z,w}(x) = \delta(w \oplus \mathbf{S}(z \oplus x)) = 1$.

Tendo agora em atenção o facto (158) vemos que a complexidade computacional para encontrar essa solução x nas sua duas componentes (construir o fraccionamento α e um caminho qualquer nesse fraccionamento) tem uma complexidade computacional que é determinada, essencialmente, pela primeira componente.

A construção do fraccionamento, na pior das circunstâncias, pode ser exponencial com o número de *bits* e, por isso, dificilmente será considerada "tratável". Se, para um determinado triplo $\langle \mathbf{S}, z, w \rangle$ o cálculo do fraccionamento for tratável, então a probabilidade $\{\mathcal{Q}\}_{\varphi}$ é igual a 1 e a entropia reduz-se ao cálculo da menor entropia da computação φ que produz esse fraccionamento.

Deve-se ter em atenção que um ataque directo ao triplo $\langle \mathbf{S}, z, w \rangle$, com um par (z, w) particular, fornece muito pouca informação sobre $\mathbf{S}$; nomeadamente sobre a chave x se ela estiver a ser usada com outros pares *entrada-saída*. Por isso faz sentido definir uma forma mais realista de ataque.

Ataque 1 Seja μ um sub-conjunto de cardinalidade N do conjunto

$$\Omega \doteq \{ (z, w) \mid \mathbf{S}(z \oplus k) = w \} \tag{67}$$

Um **ataque directo** a $\mathbf{S}$ de dimensão N é uma computação tratável que, conhecidos $\mathbf{S}$ e μ, determina k.

Notas O ataque ao triplo $\langle \mathbf{S}, z, w \rangle$ fornece aleatoriamente uma solução x possível para k; se estiver em causa apenas um par (z, w) qualquer solução x serve. É uma ataque directo a $\mathbf{S}$ de dimensão 1.

Porém se este par for apenas um dos elementos de μ a solução x encontrada é apenas um dos valores possíveis para k. Sabe-se que k, solução do ataque directo, é um dos valores possíveis do suporte de $\mathcal{Q}_{z,w}$; mas não sabemos qual é. O valor x relaciona-se com k apenas pelo facto de serem ambos elementos desse suporte.

Pode-se construir uma função boolena, análoga a (66), que representa o facto de, para <u>todos</u> os pares $(z, w) \in \mu$, se verificar $\mathcal{Q}_{z,w}(x) = 1$

$$\begin{aligned} \mathcal{Q}_\mu(x) &\doteq \Pi_{(z,w)\in\mu}\, \mathcal{Q}_{z,w}(x) = \\ &= \Pi_{(z,w)\in\mu}\, \delta(w \oplus \mathbf{S}(z \oplus x)) \end{aligned} \tag{68}$$

Uma computação φ que encontre um x tal que $\mathcal{Q}_\mu(x) = 1$ não "acerta" necessariamente em k. A probabilidade de se ter $x = k$ é determinada pelo tamanho do suporte da função $\mathcal{Q}_\mu$; isto é, pelo *peso* dessa função.

Portanto a probabilidade de uma computação determinística ϕ, que tome o resultado de φ e o considere um resultado para k, é dado pela razão entre o número de possíveis k se tivéssemos toda a informação possível (isto é o peso da função $\mathcal{Q}_\Omega$) e o número de hipótese de resultados de φ dados pelo peso de $\mathcal{Q}_\mu$.

$$-\log_2 \left\{ \mathcal{Q}_\Omega : \mathcal{Q}_\mu \right\}_\phi \;=\; -\log_2 \mathrm{wt}(\mathcal{Q}_\Omega) + \log_2 \mathrm{wt}(\mathcal{Q}_\mu)$$

A entropia de ϕ é 0 porque é determinística; por isso, usando pode-se escrever a relação que dá a entropia do ataque directo à SBox $\mathbf{S}$

$$\mathcal{H}(\mathcal{Q}_\Omega) \;=\; \min_\mu \left(\mathcal{H}(\mathcal{Q}_\mu) + \log_2 \mathrm{wt}(\mathcal{Q}_\mu) - \log_2 \mathrm{wt}(\mathcal{Q}_\Omega) \right) \tag{69}$$

No caso particular em que só existe um k possível e a solução x para $\mathcal{Q}_\mu)(x) = 1$ pode ser encontrada deterministicamente por uma computação tratável (i.e. $\mathcal{H}(\mathcal{Q}_\mu) = 0$) ainda resta a componente $\log_2 \mathrm{wt}(\mathcal{Q}_\mu)$.

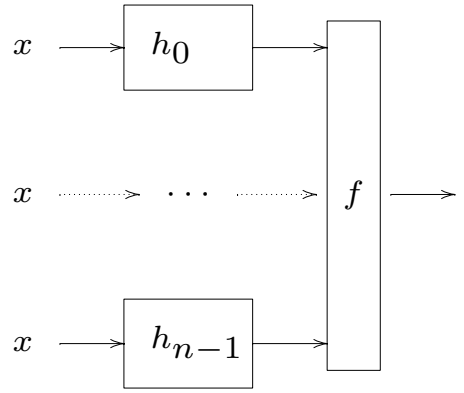
3.2 Composição de funções booleanas

Considere-se uma uma S-Box $k \times n$, $H : \mathrm{GF}(2)^k \rightarrow \mathrm{GF}(2)^n$. Pode-se escrever $H = (h_0, \ldots, h_{n-1})$ em que os vários h_i são funções boolenas de k argumentos.

$$h_i : \mathrm{GF}(2)^k \rightarrow \mathrm{GF}(2)$$

Seja $f : \mathrm{GF}(2)^n \rightarrow \mathrm{GF}(2)$ uma outra função boolena. É possível construir uma computação em que os resultados das n funções h_i são usados como argumentos de f; i.e, uma computação da forma

$$f(h_0(x), \ldots, h_{n-1}(x))$$



Fica definida uma nova função boolena (com k argumentos) a que chamamos **composição** de f e H e que representamos por

$$f \circ H \quad \text{ou} \quad H; f$$

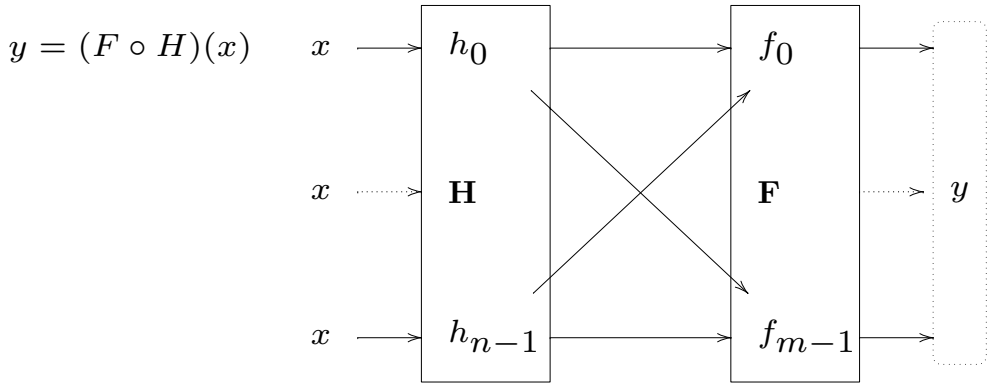
□

Considerando agora, não só uma função f, mas m funções deste tipo (formando uma S-Box de dimensão $n \times m$)

$$F = (f_0 \ , \ f_1 \ , \ \cdots \ , \ f_{m-1}) \qquad \text{com} \quad f_i : \mathrm{GF}(2)^n \rightarrow \mathrm{GF}(2)$$

então, através da composição de cada um dos f_i com H, constrói-se o vector de funções boolenas que define a **composição** das S-Boxes F e H.

$$F \circ H \doteq (f_0 \circ H \ , \ f_1 \circ H \ , \ \cdots \ , \ f_{m-1} \circ H)$$



Para exprimir o espectro destas composições é necessário aplicação a noção se selecção apresentada na definição 150 (página 184) a S-Boxes.

159 DEFINIÇÃO
Seja H uma S-Box $n \times m$; para $u \in \mathbb{U}_m$, as funções booleanas H_u e $[H]_u$ definem-se por

$$H_u = \Pi_{i \in u} \, h_i \qquad [H]_u = H_u \cdot \Pi_{i \notin u} \, (1 + h_i)$$

Deste modo, se f tem espectro $\mathcal{A}$, será $f(y) = \sum_{u \in \mathcal{A}} y_u$. Se for $y = H(x)$ teremos

$$(f \circ H)(x) \;=\; f(y) \;=\; \sum_{u \in \mathcal{A}} H_u(x)$$

Representemos por $\mathcal{B}_i$ o espectro da função boolena h_i. Então o espectro de $H_u = \Pi_{i \in u}\, h_i$ será $\left(\bigoplus_{i \in u} \mathcal{B}_i\right)$; usando as regras atrás definidas para produtos de espectros, este cálculo é polinomial em n.

O espectro de $(f \circ H)$ pode, agora, ser calculado como

$$\biguplus_{u \in \mathcal{A}} \left(\bigoplus_{i \in u} \mathcal{B}_i \right) \tag{70}$$

Funções Lineares

Infelizmente, no caso geral, a fórmula (70) tem pouco interesse computacional porque obriga a percorrer todo u no espectro de f que, em princípio, cresce exponencialmente com n.

No entanto, para algumas formas particulares de f, o espectro $\mathcal{A}$ assume formas que tornam simples o cálculo (70). Nomeadamente quando f é uma **função linear**.

Qualquer função linear $f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$ verifica

$$f(x \oplus y) = f(x) + f(y) \quad \text{para todo} \quad x, y \in \mathrm{GF}(2)^n$$

e o seu espectro é formado por conjuntos com um único índice.

Neste caso (70) resume-se a $\biguplus_{\{i\} \in \mathcal{A}} \mathcal{B}_i$ que pode ser calculado com complexidade polinomial.

Uma função linear importante é a **função traço** definida por

$$\mathrm{Tr}(x_0, x_1, \cdots, x_{n-1}) \doteq x_0 + x_1 + \cdots + x_{n-1} \tag{71}$$

cujo espectro é o conjunto $\{ \{i\} \mid i \in 0..n-1 \}$ formado por todos os índices singulares.

Esta função determina a **paridade** do vector $x = (x_0, \cdots, x_{n-1})$; i.e. a paridade do número de componmentes iguais a 1 no vector x.

Esta noção de paridade pode ser generalizada. Tomemos uma função linear f qualquer e o vector constante c^f que tem componentes a 1 exactamente para os índices onde f tem monómios; isto é,

$$(c^f)_u = 1 \qquad \text{se e só se} \quad u \in \mathrm{Sp}(f) \tag{72}$$

Designamos c^f por **máscara** ou **paridade** de f . Nessas circunstâncias

160 FACTO

Se f *é linear e tem máscara* c^f *, então* $f(x) = \mathsf{Tr}(x * c^f)$ *para todo* $x \in \mathsf{GF}(2)^n$.

Exemplo: se for $f(x) = x_0 + x_2 + x_7$, com $x \in \mathsf{GF}(2)^8$, então a máscara é

$$c^f = (1, 0, 1, 0, 0, 0, 0, 1)$$

Note-se que $x * c^f$ é o vector $(x_0, 0, x_2, 0, 0, 0, 0, x_7)$; o seu traço constrói precisanente o valor de $f(x)$.

3.3 Diferenças e Linearidade

Considere-se de novo uma S-Box $\mathbf{S}$ e o problema definido em (62) e (63) de determinar a chave x tal que $\mathbf{S}(z \oplus x) = w$.

Vamos supor que $\mathbf{S}$ era tal que existia uma solução, pelo menos, para o seguinte problema:

Diferenças críticas: encontrar $a, b \in \mathrm{GF}(2)^n$ tais que,

$$\mathbf{S}(a \oplus y) \oplus \mathbf{S}(y) = b \qquad \text{para todo} \quad y \in \mathrm{GF}(2)^n \tag{73}$$

Os elementos (a, b) chamam-se **diferenças críticas**.

Se for possível encontrar um ou mais pares de difenças críticas (a, b) então qualquer cifra assente na complexidade computacional de inverter **S** perde entropia.

Por exemplo, o ataque directo a $\mathbf{S}$ tem a seguinte variante:

Ataque 2 Criptoanálise Diferencial do Ataque Directo

1. Recolher pares (z_j, w_j) com $w_j = \mathbf{S}(z_j \oplus k)$; a chave k é a mesma em todos os pares e é a incógnita deste ataque.
2. Recolher pares de diferenças críticas (a_i, b_i) .
3. Encontrar x' tal que, para algum i e todos os j , se verifique $\mathbf{S}(z_j + x) = w_j \oplus b_i$.
4. Nestas circunstâncias tem-se $x \doteq a_i \oplus x'$ é uma solução eventual para k .

A justificação reside no facto de, sendo (a_i, b_i) uma par de diferenças críticas, verifica-se $\mathbf{S}(a_i \oplus (z_j \oplus x')) \oplus \mathbf{S}(z_j \oplus x') = b_i$.

Por construção tem-se, para todo j , $\mathbf{S}(z_j \oplus x') = w_j \oplus b_i$; escolhendo $x = x' \oplus a_i$ verifica-se, para todo j, $\mathbf{S}(z_j \oplus x) \oplus (w_j \oplus b_i) = b_i$; donde $\mathbf{S}(z_j \oplus x) = w_j$ para todo j ; isto significa que x é, eventualmente, k .

Note-se que:

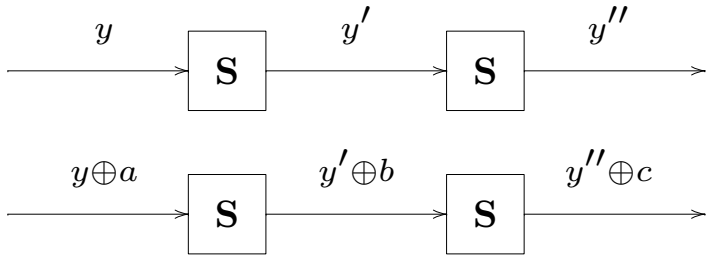
1. O passo (3) é, essencialmente, uma solução do problema inicial mas para valores diferente de *outupts*; é uma versão do problema inicial para pares $(z_j, w_j \oplus b_i)$.
   Aparentemente não se ganha nada com esta formalização dado que, como parte do problema inicial, temos de resolver um problema do mesmo tipo.
2. O ganho reside apenas no facto de ser possível adaptar o problema a uma escolha criteriosa de valores b_i caso seja possível calcular o respectivo a_i .

É possível resolver um ataque directo para diferentes conjuntos de pares $\mu_i = \left\{ (z_j, w_j \oplus b_i) \mid j \in 1.. \right\}$ e seleccionar a solução com menor entropia.

No entanto os ataques mais importantes resultam da aplicação das diferenças críticas à composição de S-Boxes.

Suponhamos um caso simples com a composição de $\mathbf{S}$ consigo próprio e vamos supor que (a, b) e (b, c) são dois pares de diferenças críticas.

Consider-se a seguinte situação em que a S-Box resultante é aplicada a duas *entradas* diferentes: y e $y \oplus a$.



(74)

Pelo propriedade das diferenças críticas, conhecidos os vários valores y, y' e y'' no primeiro caso, é muito simples determinar os valores respectivos quando a *entrada* muda de y para $y \oplus a$:

(i) O resultado intermédio muda de y' para $y' \oplus b$ porque o primeiro par de diferenças críticas me diz que $\mathbf{S}(y \oplus a) \oplus \mathbf{S}(y) = b$; logo, $\mathbf{S}(y \oplus a) = \mathbf{S}(y) \oplus b = y' \oplus b$.

(ii) O resultado final muda de y'' para $y'' \oplus c$ porque o par de diferenças críticas (b, c) diz-nos que $\mathbf{S}(y' \oplus b) \oplus \mathbf{S}(y') = c$.
A entrada do 2º $\mathbf{S}$ mudou de y' para $y' \oplus b$; a diferença crítica c repflecte-se, agora, na respectiva saída.

Por este facto se diz que

o par de diferenças críticas (a, b) **propaga** *a diferença* a, *na entrada, para a diferença* b *na saída,*

A propagação de diferenças diminui consideravelmente a entropia de uma concatenação de várias S-Boxes. O aumento de entropia que devia resultar dessa concatenção acaba por se não se manifestar dado que muita da complexidade se perde com a existência de diferenças críticas.

A propósito, o exemplo anterior demonstra o seguinte facto

161 FACTO
Se (a, b) *é um par de diferenças críticas para* $\mathbf{S}$ *e* (b, c) *é um par de diferenças críticas para* $\mathbf{R}$ *então* (a, c) *é um par de diferenças críticas para* $\mathbf{R} \circ \mathbf{S}$.

□

É possível exprimir diferenças críticas através de uma função booleana. Representemos por $\Delta^{\mathbf{S}}_{a,b}$ a função que, para todo o argumento y, verifica

$$\Delta^{\mathbf{S}}_{a,b}(y) \;=\; \delta(\mathbf{S}(a \oplus y) \oplus \mathbf{S}(y) \oplus b) \tag{75}$$

e representemos por

$$\rho_{\mathbf{S}}(a,b) \;\doteq\; 2^{-n} \cdot \mathsf{wt}(\Delta^{\mathbf{S}}_{a,b})$$

a razão entre número de argumentos y para os quais $\Delta^{\mathbf{S}}_{a,b}(y) = 1$ e o número total de argumentos possíveis.

Note-se que (a,b) é um par de diferenças críticas se e só se $\Delta^{\mathbf{S}}_{a,b}$ é a função constante 1. Ou seja, quando $\rho_{\mathbf{S}}(a,b) = 1$.

Porém pode acontecer que (a,b) não seja um par de diferenças críticas mas, ainda assim, exista a possibilidade de, para a maioria dos possíveis argumentos y, se verificar $\Delta^{\mathbf{S}}_{a,b}(y) = 1$; isto significa que será $\rho_{\mathbf{S}}(a,b) < 1$ mas, ainda assim, próximo do limite 1 .

Recordemos a noção de entropia; a notação $\left\{\Delta^{\mathbf{S}}_{a,b}\right\}_{\varphi}$ representa a probabilidade de a computação φ gerar um resultado y que verifique $\Delta^{\mathbf{S}}_{a,b}(y) = 1$.

Se (a,b) fosse um para de diferenças críticas esta probabilidade seria sempre 1 qualquer que seja a computação φ que produza resultados em $\mathsf{GF}(2)^n$.

É possível exprimir a entropia de $\Delta^{\mathbf{S}}_{a,b}$ relativo a φ.

$$\mathcal{H}(\Delta^{\mathbf{S}}_{a,b})_{\varphi} \doteq \mathcal{H}(\varphi) - \log_2 \left\{\Delta^{\mathbf{S}}_{a,b}\right\}_{\varphi}$$

Se φ for determinística será $\mathcal{H}(\varphi) = 0$. Se se comportar como um gerador aleatório, a probabilidade de "acertar" num elemento so suporte de $\Delta^{\mathbf{S}}_{a,b}$ é $\rho_{\mathbf{S}}(a,b)$. Nestas circunstâncias particulares (para um tal φ) será

$$\mathcal{H}(\Delta^{\mathbf{S}}_{a,b})_{\varphi} = -\log_2 \rho_{\mathbf{S}}(a,b) \qquad \text{ou} \tag{76}$$

$$\rho_{\mathbf{S}}(a,b) = 2^{-\mathcal{H}(\Delta^{\mathbf{S}}_{a,b})_{\varphi}}$$

Estas relações definem uma aproximação para a perda de entropia introduzida por um candidato a par de diferenças críticas quando não existe informação sobre a função de diferenças $\Delta^{\mathbf{S}}_{a,b}$ para além do seu suporte.

Facilmente se generaliza o facto 161 para "candidatos" a pares de diferenças críticas

162 FACTO

$$\rho_{\mathbf{R}\circ\mathbf{S}}(a,c) \geq \rho_{\mathbf{S}}(a,b) \cdot \rho_{\mathbf{R}}(b,c)$$

□

A **distância de Hamming** entre duas funções $f, g : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (representa-se por $d(f,g)$) é o peso de $f+g$; isto é, a $d(f,g)$ conta o número de argumentos nos quais as funções divergem. A **não-linearidade** de f é a menor das distâncias $d(f,g)$ quando g percorre todas as funções booleanas afins em $\mathrm{GF}(2)^n$.

A **correlação** entre f e g, é a função racional

$$\mathcal{C}(f,g) \doteq 1 - 2^{-(n-1)} \cdot d(f,g) \tag{77}$$

Nota:
A correlação tem um resultado compreendido entre -1 e 1; caso as funções sejam iguais temos $d(f,g)=0$ e $\mathcal{C}(f,g)=1$; se as funções forem totalmente distintas (i.e. $f=1+g$) então $d(f,g)=2^n$ e $\mathcal{C}(f,g)=-1$; se o número de argumentos x para os quais $f(x) \neq g(x)$ for metade do total, então $d(f,g)=2^{n-1}$ e $\mathcal{C}(f,g)=0$.

Convenção:
Vimos que cada função linear $\omega : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ é únicamente determinada por um elemento $c^\omega \in \mathrm{GF}(2)^n$ designado por máscara ou paridade ω. Se não surgirirem ambiguidades designaremos a função e a respectiva paridade pelo mesmo símbolo. Assim escrever $\omega \in \mathrm{GF}(2)^n$ e $\omega(x)$ são notações compatíveis: a primeira representa o vector paridade e a segunda faz referência à função linear que esse vector determina.

Alguns resultados intermédios

163 FACTO

Sejam ω, υ funções lineares e f uma outra função boolena qualquer. Então

(i) $\mathcal{C}(\omega, \upsilon) = \delta(\omega \oplus \upsilon)$

(ii) $\mathcal{C}(1 + f, \omega) = -\mathcal{C}(f, \omega)$

(iii) *Se* $g(x) = f(x) + \upsilon(x)$ *então* $\mathcal{C}(g, \omega) = \mathcal{C}(f, \omega + \upsilon)$

164 DEFINIÇÃO

Dada uma função boolena $f : \mathsf{GF}(2)^n \longrightarrow \mathsf{GF}(2)$ *e uma função linear* ω *, seja*

$$F(\omega) \doteq \mathcal{C}(f, \omega) \tag{78}$$

A função $F : \omega \mapsto F(w)$ *, fazendo* ω *percorrer todas as* 2^n *funções lineares realizáveis em* $\mathsf{GF}(2)^n$*, chama-se o* **espetro de Walsh** *de* f *.*

A transformação $\mathcal{W} : f \mapsto F$ *, que associa* f *ao seu espectro de Walsh, chama-se* **transformada de Walsh-Hadamard**.

165 FACTO

Para todo $f, g \in \mathsf{GF}(2)^n \longrightarrow \mathsf{GF}(2)$ *verifica-se*

$$\mathcal{C}(f, g) = 2^{-n} \cdot \sum_x (-1)^{f(x)+g(x)} \tag{79}$$

e, se $F \doteq \mathcal{W}(f)$, *verifica-se para todo* $x \in \mathsf{GF}(2)^n$

$$(-1)^{f(x)} = \sum_{\omega} F(\omega)\,(-1)^{\omega(x)} \tag{80}$$

em que a primeira soma se estende a todos $x \in \mathsf{GF}(2)^n$ *e a segunda soma a todas as funções lineares* $\omega \in \mathsf{GF}(2)^n$.

Se $G \doteq \mathcal{W}(g)$ *verifica-se* $\mathcal{W}(f+g) = F \otimes G$ *em que a* **convolução** *de espectros de Walsh é definida por*

$$(F \otimes G)(\omega) = \sum_{\upsilon} F(\omega \oplus \upsilon) \cdot G(\upsilon) \tag{81}$$

166 Definição
Seja $\mathbf{S}$ *uma SBox definida por um vector de funções boolenas* $(h_0, h_1, \cdots, h_{n-1})$. *Para quaisquer duas funções lineares* $\omega, \upsilon \in \mathsf{GF}(2)^n$ *seja*

$$\mathcal{C}^{\mathbf{S}}(\upsilon, \omega) \doteq \mathcal{C}(\upsilon \circ \mathbf{S}, \omega) \tag{82}$$

Vista como uma matriz $2^n \times 2^n$, *a função* $\mathcal{C}^{\mathbf{S}}$ *chama-se* **matriz de correlação** *de* $\mathbf{S}$.

167 Facto
Nas condições da definição anterior,

(i) $\mathcal{W}(\upsilon \circ \mathbf{S}) = \bigotimes_{\upsilon_i=1} \mathcal{W}(h_i)$.

(ii) *Se* f *é uma função boolena de espectro* F *e* $g = f \circ \mathbf{S}$ *então o seu espectro* G *pode ser calculado como*

$$G \;=\; F \times \mathcal{C}^{\mathbf{S}}$$

em que G *e* F *são vistos como vectores-linha de* 2^n *componentes e a operação* $\times$ *é vista como a multiplicação de matrizes.*

(iii) *Se* $\mathbf{R}$ *é uma outra S-Box então a matriz correlação da composição* $\mathbf{R} \circ \mathbf{S}$ *é o produto (no sentido da multiplicação de matrizes) das duas matrizes de correlação.*

$$\mathcal{C}^{\mathbf{R}\circ\mathbf{S}} \;=\; \mathcal{C}^{\mathbf{R}} \times \mathcal{C}^{\mathbf{S}}$$

3.4 Funções de argumento GF(2^n)

As funções $f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)$ (SBoxes $n \times n$) têm argumentos que são vistos como elementos do corpo de Galois de ordem n. Como o domínio da função é finito ela pode ser sempre descrita[36] por um polinómio de grau inferior a $2^n - 1$

$$f(x) \;=\; \sum_{i=0}^{2^n-1} a_i\, x^i \;, \qquad a_i \in \mathrm{GF}(2^n) \tag{83}$$

Tendo em atenção que, para todo $x \neq 0$, se verifica

$$x^{2^n-1} = 1 \;,\; x^{2^n-2} = x^{-1} \;,\; \cdots \;,\; x^{2^n-1-i} = x^{-i}$$

então é possível escrever a representação anterior do seguinte modo:

$$f(x) \;=\; \begin{cases} a_0 & , \quad \text{se} \quad x = 0 \\ b_0 + \sum_{i=1}^{N} a_i\, x^i + b_i\, x^{-i} & , \quad \text{se} \quad x \neq 0 \end{cases} \tag{84}$$

[36] Qualquer função $f : \mathbb{F}_q \longrightarrow \mathbb{F}_q$, que passa por N pontos, pode ser aproximada por um polinómio de Lagrange de grau $N - 1$; o número total de argumentos de f é q e, por isso, existem, quanto muito, q pontos que a determinam; logo qualquer aproximação (incluindo a própria função) tem grau igual ou inferior a $q - 1$.

com

$$N = 2^{n-1} - 1 \ , \quad b_0 = a_0 + a_{(2^n-1)} \ , \quad b_i = a_{(2^n-1-i)} \quad i > 0$$

EXEMPLO 25: Na cifra AES as operações sobre *bytes* são descritas por funções $f : \mathrm{GF}(2^8) \rightarrow \mathrm{GF}(2^8)$. O corpo de Galois é representado numa base polinomial do tipo 0 usando o polinómio característico $\mathbf{c} = y^8+y^4+y^3+y+1$.

A definição do **AES** especifica uma função `ByteSub` que é a composição de duas funções $\mathrm{ByteSub} = \mathrm{Inv} \circ \mathrm{Afim}$. A primeira das quais é a função auto-inversa

$$\mathrm{Inv}(x) \ = \ \begin{cases} 0 & \text{se} \quad x = 0 \\ x^{-1} & \text{se} \quad x \neq 0 \end{cases} \tag{85}$$

A segunda função é uma transformação afim definida não sobre $\mathrm{GF}(2^8)$ mas sim sobre sobre $\mathrm{GF}(2)^8$; tem a forma

$$\mathrm{Afim}(x) \ = \ c \oplus (\omega_0(x), \cdots, \omega_{n-1}(x))$$

$$\text{com} \quad c, \omega_i \in \mathrm{GF}(2)^8$$

(os valores concretos das constantes c, ω_i constam na especificação oficial do AES).

É importante ter uma representação de ambas as funções da mesma forma. Por isso é indispensável ver como converter uma representação linear ou afim genérica em $\mathrm{GF}(2)^n$ numa função de domínio $\mathrm{GF}(2^n)$. É o que faremos em seguida.

Alguns casos especiais de funções de domínio $\mathrm{GF}(2^n)$ merecem referência especial:

1. Os **automorfismos** $f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)$ (funções injectivas que preservam a estrutura do corpo) fixam necessariamente os elementos de $\mathrm{GF}(2)$.

   Vendo $\mathrm{GF}(2^n)$ como uma extensão de $\mathrm{GF}(2)$, o facto 140 determina que f tem a forma $\sigma^k : x \mapsto x^{2^k}$, para algum $k \in 0..n-1$. Só as potências do morfismo de Frobenius $\sigma : x \mapsto x^2$ são automorfismos neste corpo. É possível escrever os σ^k de uma forma vectorial de modo a permitir o uso de uma álgebra matricial para representar transformações lineares.
   Para tal define-se a função $(\cdot)_\sigma : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)^n$ que transforma um elemento x do corpo de Galois no vector, sobre esse corpo, formado por todos $\sigma^k(x)$ (as imagens de x pelos vários automorfismos).

$$x_\sigma \doteq (x \ , \ \sigma(x) \ , \ \sigma^2(x) \ , \ \cdots \ , \ \sigma^{n-1}(x)) \tag{86}$$

2. As **funções boolenas** $f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)$ podem ser representadas pelas formas genéricas em (83) ou (84) tendo em atenção que o contradomínio $\mathrm{GF}(2)$ está imerso em $\mathrm{GF}(2^n)$.

EXEMPLO 26: a função *traço* (ver definição 141 – página 149), é definida pelo polinómio $\mathrm{tr}(x) = x + x^2 + x^4 + \cdots + x^{2^{n-1}}$.

3. As **funções lineares** $f : \mathrm{GF}(2^n) \to \mathcal{K}$, sendo $\mathcal{K}$ uma qualquer extensão de $\mathrm{GF}(2)$, são funções que preservam somas e multiplicações por escalares:

$$f(x+y) = f(x) + f(y) \quad , \quad f(a\,x) = a\,f(x)$$

para todos $x, y \in \mathrm{GF}(2^n)$ e $a \in \mathrm{GF}(2)$.

A função linear f pode sempre ser escrita como um polinómio

$$f(x) \;=\; \sum_{k=0}^{n-1} a_k\,\sigma^k(x) \qquad \text{com} \quad a_k \in \mathcal{K} \tag{87}$$

Representanto por $\mathbf{a}$ o vector $(a_0, a_1, \cdots, a_{n-1})$, o polinómio (87) pode ser escrito como o produto escalar[37] de dois vectores

$$f(x) \;=\; \mathbf{a} \cdot x_\sigma \tag{88}$$

[37] Se $u, v \in X^n$ o produto escalar $u \cdot v$ é $u^t\,v = v^t\,u$, em que $(\cdot)^t$ representa a transposição de matrizes.

EXEMPLO 27: Como caso particular temos construções da forma $\mathrm{tr}(x \cdot y)$; pela definição verifica-se imediatamente que, para todos $x, y \in \mathrm{GF}(2^n)$

$$\mathrm{tr}(x\,y) \;=\; x_\sigma \cdot y_\sigma$$

Voltando à representação em (88), seja $y = f(x) = \mathbf{a} \cdot x_\sigma$. Então y_σ pode ser calculado simplesmente como

$$y_\sigma \;=\; \mathbf{A}\, x_\sigma$$

sendo $\mathbf{A}$ a matriz cujo elemento genérico é

$$\mathbf{A}_{ij} = (a_k)^{2^i} \qquad \text{com} \quad k = j - i \pmod{n-1} \tag{89}$$

4. As **funções bilineares** são funções de dois argumentos

$$f : \mathrm{GF}(2^n) \times \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)$$

que são lineares em cada um dos seus argumentos.
Isto é, para todo $x, y, z \in \mathrm{GF}(2^n)$ e $a \in \mathrm{GF}(2)$

$$f(x, y+z) = f(x,y) + f(x,z) \quad , \quad f(x+z, y) = f(x,y) + f(z,y)$$

$$f(x, a \cdot y) = a \cdot f(x,y) = f(a \cdot x, y)$$

A forma mais geral para estas funções é dada por

$$f(x) \;=\; x_\sigma \cdot (\mathbf{A}\, y_\sigma) \tag{90}$$

sendo $\mathbf{A} \in \mathrm{GF}(2^n)^{n\times n}$ uma matriz com elementos em $\mathrm{GF}(2^n)$.

EXEMPLO 28:

Exemplos de formas bilineares em $\mathrm{GF}(2^n)$, com $n > 3$

$$f(x,y) = x \cdot y \quad , \quad f(x,y) = x^2 \cdot y^4 + x^4 \cdot y^2 + c \cdot x^8 \cdot y^8$$

5. As **funções quadráticas** $g : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)$ têm a forma $g(x) = f(x,x)$ sendo $f(\cdot,\cdot)$ uma função bilinear. Isto é,

$$g(x) \;=\; x_\sigma \cdot \mathbf{A}\, x_\sigma \tag{91}$$

para uma matriz apropriada $\mathbf{A} \in \mathrm{GF}(2^n)^{n\times n}$.

EXEMPLO 29:

A função $g(x) = x^3$ é uma função quadrática já que pode ser escrita na forma $f(x,x)$ sendo $f(x,y) = x\, y^2$ que é, claramente, uma forma bilinear.

Pelo mesmo motivo qualquer monómio x^k é uma função quadrática se se puder escrever $k = 2^i + 2^j \pmod{2^n - 1}$ para i, j apropriados.

6. Uma base $\mathcal{B}$ de $\mathrm{GF}(2^n)$ determina sempre n **projecções**; i.e., funções booleanas $p_\mu : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)$, uma para cada elemento $\mu \in \mathcal{B}$, de tal forma que qualquer $x \in \mathrm{GF}(2^n)$ pode ser reconstruído a partir dos elementos μ e dos valores $p_\mu(x)$

$$x \;=\; \sum_{\mu \in \mathcal{B}} p_\mu(x)\, \mu \tag{92}$$

Representemos por $(\cdot)^{\tilde{}} : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)^n$ a função que associa x ao vector das suas projecções. Vectorialmente, (92) é

$$x \;=\; \mathcal{B} \cdot x^{\tilde{}} \tag{93}$$

Notas Algumas propriedades das projecções que derivam directamente de (92)

(i) Como a representação de x em $\mathcal{B}$ é única, as projecções são também únicas.

(ii) A projecção em $\mu \in \mathcal{B}$ de um outro elemento da base $\upsilon \in \mathcal{B}$ tem de ser zero porque, por definição de base, não é possível representar υ como uma soma de outros elementos da mesma base. Por isso

$$p_\mu(\upsilon) \;=\; \delta(\mu + \upsilon) \qquad \forall\, \mu, \upsilon \in \mathcal{B}$$

(iii) São sempre funções lineares que preservam a multiplicação escalar; i.e, verificam

$$p_\mu(0) = 0 \quad , \quad p_\mu(x+y) = p_\mu(x) + p_\mu(y) \quad , \quad p_\mu(a\,x) = a\,p_\mu(x)$$

para todos $x, y \in \mathrm{GF}(2^n)$ e $a \in \mathrm{GF}(2)$.

(iv) Para todo $x \in \mathrm{GF}(2^n)$ existe y tal que $p_\mu(x) \neq p_\mu(y)$; isto é, as projecções são funções sobrejectivas.

Se compararmos as duas últimas propriedades com as da função traço (página 149) vemos que elas coincidem; por isso é natural que existam semelhanças entre as duas noções. De facto é relativamente simples provar o seguinte resultado,

168 FACTO
Para qualquer elemento $\mu \in \mathcal{B}$ de uma base de $\mathrm{GF}(2^n)$ existe um único elemento $\mu' \in \mathrm{GF}(2^n)$, designado por **elemento dual** *de μ, tal que, para todo $x \in \mathrm{GF}(2^n)$,*

$$p_\mu(x) \;=\; \mathrm{tr}(\mu' x) \;=\; (\mu')_\sigma \cdot x_\sigma$$

O conjunto $\mathcal{B}' \doteq \{\, \mu' \mid \mu \in \mathcal{B} \,\}$ de todos os elementos duais forma uma nova base de $\mathrm{GF}(2^n)$ que será designada por **base dual** *de $\mathcal{B}$.*

Sugestão de prova: Construa-se a matriz $n \times n$ de elementos $\mathbf{T}_{\mu\upsilon} \doteq \mathrm{tr}(\mu\,\upsilon)$; as colunas da sua inversa $\mathbf{T}^{-1}$ determinam os elementos da base dual.

□

Ordenando os elementos de $\mathcal{B}$ num vector, e preservando a mesma ordem na base dual $\mathcal{B}'$, a prova do facto

anterior mostra que estes vactores estão relacionados por

$$\mathcal{B}' \;=\; \mathbf{T}^{-1}\,\mathcal{B} \tag{94}$$

em que $\mathbf{T}$ designa a matriz dos traços cruzados: $\mathbf{T}_{\mu\upsilon} = \mathrm{tr}(\mu\,\upsilon)$.

EXEMPLO 30: Consideremos de novo $\mathrm{GF}(2^4)$ com uma base polinomial do tipo 0

$$\mathcal{B} = \left\{1\ ,\ \beta\ ,\ \beta^2\ ,\ \beta^3\right\}$$

Para calcular a matriz dos traços $\mathbf{T}_{ij} \doteq \mathrm{tr}(\beta^i \cdot \beta^j) = \mathrm{tr}(\beta^{i+j})$ basta calcular a lista dos traços das potências de β para expoentes entre 0 e 6.

Usando o polinómio característico $\mathbf{c}(X) = X^4 + X + 1$ facilmente se constrói a tabela

i	0	1	2	3	4	5	6
$\mathrm{tr}(\beta^i)$	0	0	0	1	0	0	1

A matriz $\mathbf{T}$ e a sua inversa $\mathbf{T}^{-1}$ são

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Portanto a base dual é formada pelos elementos (pela ordem dos respectivos duais em $\mathcal{B}$)

$$\mathcal{B}' = \left\{ 1+\beta^3 \ , \ \beta^2 \ , \ \beta \ , \ 1 \right\}$$

e as projecções respectivas serão

$$p_1(x) = \mathsf{tr}(x) + \mathsf{tr}(x\,\beta^3) \qquad , \qquad p_\beta(x) = \mathsf{tr}(x\,\beta^2)$$
$$p_{\beta 2}(x) = \mathsf{tr}(x\,\beta) \qquad , \qquad p_{\beta 3}(x) = \mathsf{tr}(x)$$

o que permite concluir a identidade

$$x = \mathsf{tr}(x) + \mathsf{tr}(x\,\beta^3) + \mathsf{tr}(x\,\beta^2)\,\beta + \mathsf{tr}(x\,\beta)\,\beta^2 + \mathsf{tr}(x)\,\beta^3$$

□

A noção de elemento dual pode ser extendida a qualquer $x \in \mathrm{GF}(2^n)$. O **elemento dual** de x na base $\mathcal{B}$, representado por x', é elemento de $\mathrm{GF}(2^n)$ que tem exactamente as mesmas componentes de x mas na base dual $\mathcal{B}'$; i.e. para todo $\mu \in \mathcal{B}$

$$p_\mu(x) = \mathrm{tr}(\mu' x) = \mathrm{tr}(\mu x') = p_{\mu'}(x') \tag{95}$$

Tomando como implícita a base $\mathcal{B}$, o operador $(\cdot)^\sim$ associa cada elemento de $\mathrm{GF}(2^n)$ ao vector das suas projecções nessa base. Então verifica-se

$$x'^\sim = \mathbf{T}^{-1} x^\sim \quad , \quad x' = \mathcal{B} \cdot x'^\sim = \mathcal{B}' \cdot x^\sim \tag{96}$$

□

O operador $(\cdot)_\sigma$ pode ser extendido para vectores de elementos $\mathrm{GF}(2^n)$. Dado $A \in \mathrm{GF}(2^n)^n$ a matriz $A_\sigma \in \mathrm{GF}(2^n)^{n\times n}$ tem, por linha de ordem i, o vector $(A_i)_\sigma$; isto é $(A_\sigma)_{ik} = \sigma^k(A_i)$.

Nestas circunstâncias

169 FACTO

Se $\mathcal{B}, \mathcal{B}'$ são um par de bases duais então:

1. $(\mathcal{B}')_\sigma = \mathbf{T}^{-1}\,\mathcal{B}_\sigma$

2. $\tilde{x} = (\mathcal{B}')_\sigma \, x_\sigma \quad e \quad x_\sigma = (\mathcal{B}_\sigma)^t \, \tilde{x}$
3. $(\mathcal{B}_\sigma)^t \, \mathcal{B}_\sigma = \mathbf{T} \quad e \quad (\mathcal{B}_\sigma)^t \, (\mathcal{B}')_\sigma = \mathbf{I}.$
4. $\mathcal{B}_\sigma \, (x')_\sigma = (\mathcal{B}')_\sigma \, x_\sigma.$

Prova O primeiro resultado é consequência da relação $\mathcal{B}' = \mathbf{T}^{-1} \, \mathcal{B}$, da linearidade dos morfismos σ^k e do facto de fixarem todos os elementos da matrix $\mathbf{T}^{-1}$.

O segundo resultado é consequência do anterior e do facto 168. Os últimos resultados são as definições da matriz dos traços cruzados $\mathbf{T}$, da base dual e elemento dual.

□

Todas estas noções são essenciais quando se pretende estudar funções booleanas que requerem "mudança de representação"; isto ocorre quando uma função é formada pela composição de uma sequência de funções que manipulam as palavras de *bits* alternadamente na representação $\mathrm{GF}(2^n)$ e na representação $\mathrm{GF}(2)^n$.

É o caso da SBox do AES (introduzida na exemplo 25) que é determinada pela composição da função $x \mapsto x^{-1}$ em $\mathrm{GF}(2^8)$ seguida de uma transformação afim em $\mathrm{GF}(2)^8$.

O problema essencial é o seguinte:

Dada uma base de representação $\mathcal{B}$ *em* $\mathsf{GF}(2^n)$, *dada uma função* $f^{\tilde{}}$ *de domínio* $\mathsf{GF}(2)^n$, *construir a função* f *de domínio* $\mathsf{GF}(2^n)$ *que verifica* $f^{\tilde{}}(x^{\tilde{}}) = f(x)^{\tilde{}}$ *para todo* x.
Ou, inversamente, dada a função f, *construir* $f^{\tilde{}}$.

Note-se que: $f^{\tilde{}}(x^{\tilde{}})$ denota a função de palavras de bits $f^{\tilde{}}$ aplicada ao vector de bits que representa x (visto como elemento do corpo de Galois); $f(x)^{\tilde{}}$ é o vector de bits que representa o resultado $f(x)$; a relação entre as duas funções diz-nos que estes dois vectores são iguais.

Vamos procurar resolver este problema para algumas formas particulares de funções $f^{\tilde{}} : \mathsf{GF}(2)^n \to \mathsf{GF}(2)^n$ ou $f^{\tilde{}} : \mathsf{GF}(2)^n \to \mathsf{GF}(2)$.

$f^{\tilde{}}(x^{\tilde{}}) = x^{\tilde{}} \oplus c^{\tilde{}}$ com $c \in \mathsf{GF}(2^n)$.

Esta é a função *branqueamento* ("whitening") que ocorre quase sempre nos andares das cifras. A função de domínio $\mathsf{GF}(2^n)$ equivalente é

$$f(x) \;=\; x + c$$

$f^{\tilde{}}(x^{\tilde{}}) = c^{\tilde{}}(x^{\tilde{}}) = \mathsf{Tr}(c^{\tilde{}} * x^{\tilde{}})$ com $c \in \mathsf{GF}(2^n)$

Forma genérica da função booleana linear; a função de domínio $\mathsf{GF}(2^n)$ equivalente é

$$f(x) \;=\; \mathsf{tr}(c'\,x) \;=\; (c')_\sigma \cdot x_\sigma$$

Prova: Simples utilização das identidades no facto 169

$$f^{\tilde{}}(x^{\tilde{}}) = c^{\tilde{}} \cdot x^{\tilde{}} =$$

$$(\mathcal{B}_\sigma\,(c')_\sigma) \cdot (\mathcal{B}')_\sigma\, x_\sigma = (((\mathcal{B}')_\sigma)^t \mathcal{B}_\sigma\,(c')_\sigma) \cdot x_\sigma = (c')_\sigma \cdot x_\sigma$$

$f^{\tilde{}}(x^{\tilde{}}) = \mathbf{H}\, x^{\tilde{}}$ com $\mathbf{H} \in \mathrm{GF}(2)^{n\times n}$.

Esta é a forma genérica da SBox linear onde cada *bit* à saída é uma combinação linear dos *bits* de entrada.
A função de domínio $\mathrm{GF}(2^n)$ equivalente é o polinómio

$$f(x) = \quad \mathbf{h} \cdot x_\sigma \quad = \quad \sum_{k=0}^{n-1} \mathbf{h}_k\, x^{2^k} \tag{97}$$

em que $\mathbf{h} \in \mathrm{GF}(2^n)^n$ é o vector

$$\mathbf{h} = \quad (\mathcal{B}')_\sigma^t\, \mathbf{H}^t\, \mathcal{B}$$

Prova:

$$f(x) = \mathcal{B} \cdot f^{\sim}(x^{\sim}) =$$

$$\mathcal{B} \cdot (\mathbf{H}\,(\mathcal{B}')_{\sigma}\,x_{\sigma}) = ((\mathcal{B}')^{t}_{\sigma} \cdot \mathbf{H}^{t}\,\mathcal{B}) \cdot x_{\sigma}$$

É importante ter-se em conta que (97), apesar da forma aparentemente complexa, é uma função linear. porque tem a forma prevista em (87).

Pode-se interpretar $\mathbf{H}^{t}\,\mathcal{B}$ como o vector determinado pelas colunas de $\mathbf{H}$ vendo cada coluna como os coeficientes de um elemento de GF(2^{n}) na base $\mathcal{B}$.

Este vector (e, consequentemente, a matriz $\mathbf{H}$) pode ser recuperado de $\mathbf{h}$ por

$$\mathbf{H}^{t}\,\mathcal{B} \;=\; \mathcal{B}_{\sigma}\,\mathbf{h}$$

Isto permite fazer a conversão em sentido inverso: dada a função linear de domínio GF(2^{n}), obter a matriz que determina a função equivalente de domínio GF$(2)^{n}$.

A relação entre $\mathbf{H}$ e $\mathbf{h}$ pode ainde ser vista de outr forma; note-se que se podia escrever

$$\mathbf{h} \cdot x_{\sigma} = \; (\mathbf{H}^{t}\,\mathcal{B}) \cdot (\mathcal{B}')_{\sigma}\,x_{\sigma} = (\mathbf{H}^{t}\,\mathcal{B}) \cdot x^{\sim}$$

Finalmente pode-se ver

$$\mathbf{h} \;=\; (\mathbf{H}\,\mathcal{B}')^{t}_{\sigma}\,\mathcal{B}$$

$\mathbf{H}\,\mathcal{B}'$ representa ver as linhas da matriz como vectores representados na base $\mathcal{B}'$; isto é, se ω_{i} representar o elemento de GF(2^{n}) representado pela linha de ordem i da matriz **H**, então $\mathbf{H}\,\mathcal{B}' = (\omega'_{0}, \cdots, \omega'_{n-1})$ é o vector dos elementos duais.

EXEMPLO 31: Regressando à cifra AES e ao exemplo 25, a transformação afim $g\tilde{}(y\tilde{}) = \mathbf{b} \oplus \mathbf{H}\, y\tilde{}$ é definida, na especificação da cifra, por

$$\mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

ou, representando cada *byte* em base hexadecimal e a marix como um vector de colunas,

$$\mathbf{b} = 63 \qquad \mathbf{H} = (\texttt{8F}\ ,\ \texttt{C7}\ ,\ \texttt{E3}\ ,\ \texttt{F1}\ ,\ \texttt{F8}\ ,\ \texttt{7C}\ ,\ \texttt{3E}\ ,\ \texttt{1F})$$

A especificação do AES define o polinómio caractarístico

$$\mathbf{c}[X] = 1 + X + X^3 + X^4 + X^8$$

Para uma base polinomial do tipo 0 gerada por este polinómio, e usando a mesma metodologia de representação,

temos a matriz dos traços cruzados, a sua inversa e $\mathcal{B}_\sigma$

$$\mathbf{T} = (05\ ,\ 0A\ ,\ 15\ ,\ 2B\ ,\ 57\ ,\ AE\ ,\ 5C\ ,\ B9)$$

$$\mathbf{T}^{-1} = (94\ ,\ 0D\ ,\ 1A\ ,\ A0\ ,\ 65\ ,\ CA\ ,\ 25\ ,\ 2A)$$

$$\mathcal{B}_\sigma = \begin{bmatrix} 01 & 01 & 01 & 01 & 01 & 01 & 01 & 01 \\ 02 & 04 & 10 & 1B & 5E & E4 & 4D & FA \\ 04 & 10 & 1B & 5E & E4 & 4D & FA & 02 \\ 08 & 40 & AB & B3 & E8 & 1D & 4A & EF \\ 10 & 1B & 5E & E4 & 4D & FA & 02 & 04 \\ 20 & 6C & 97 & 94 & 91 & 80 & 9A & C5 \\ 40 & AB & B3 & E8 & 1D & 4A & EF & 08 \\ 80 & 9A & C5 & 20 & 6C & 97 & 94 & 91 \end{bmatrix}$$

Os elementos de $\mathcal{B}_\sigma$ são polinómios; nesta representação são apresentados os seus coeficientes como vectores de bits, começando no grau mais elevado. Por exemplo $E4 = 11100100$ denota o polinómio $X^7 + X^6 + X^5 + X^2$.

Com estas três matrizes é possível computar todos os elementos necessários; por exemplo,

$$(\mathcal{B}')_\sigma = \mathbf{T}^{-1}\,\mathcal{B}_\sigma$$

que é a componente essencial para calcular $\mathbf{h}$ a partir de $\mathbf{H}$.

Feitas as contas conclui-se

$$\mathbf{h} = (\texttt{05}\ ,\ \texttt{09}\ ,\ \texttt{F9}\ ,\ \texttt{25}\ ,\ \texttt{F4}\ ,\ \texttt{01}\ ,\ \texttt{B5}\ ,\ \texttt{8F})$$

Conclui-se portanto que a transformação afim do AES é

$$\begin{aligned} g(y) = \quad & \texttt{63} + \texttt{05} \cdot y + \texttt{09} \cdot y^2 + \texttt{F9} \cdot y^4 + \texttt{25} \cdot y^8 + \\ & + \texttt{F4} \cdot y^{16} + \texttt{01} \cdot y^{32} + \texttt{B5} \cdot y^{64} + \texttt{8F} \cdot y^{128} \end{aligned}$$

A SBox do AES resulta da composição dessa função ao resultado da transformação

$$x \mapsto x^{254}$$

que mapeia 0 em si próprio e qualquer $x \neq 0$ em x^{-1}.

A transformação final abtém-se então substituindo, em $g(y)$, a variável y por x^{254} e reduzindo os expoentes módulo 255 (uma vez que $x^{255} = 1$ se $x \neq 0$).

Por exemplo

$$y^{128} \quad \mapsto \quad (x^{254})^{128} \quad \mapsto \quad x^{(254*128 \mod 255)} \quad \mapsto \quad x^{127}$$

Genericamente y^k reduz-se a x^{255-k}. O resultado final é

$$f(x) = \mathtt{63} + \mathtt{05} \cdot x^{254} + \mathtt{09} \cdot x^{253} + \mathtt{F9} \cdot x^{251} + \mathtt{25} \cdot x^{247} + \\ + \mathtt{F4} \cdot x^{239} + \mathtt{01} \cdot x^{223} + \mathtt{B5} \cdot x^{191} + \mathtt{8F} \cdot x^{127} \tag{98}$$

□

Finalmente vamos examinar a representação das **funções bilineares**

$$f(x, y) = x_\sigma \cdot \mathbf{A}\, y_\sigma \tag{99}$$

ou equivalentemente, com $\boldsymbol{\alpha} \doteq \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t$

$$f(x, y) = x^{\tilde{}} \cdot \boldsymbol{\alpha}\, y^{\tilde{}} \tag{100}$$

Usando as identidades $x_\sigma = \mathcal{B}_\sigma^t\, x^{\tilde{}}$ e $\boldsymbol{\alpha} = \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t$ tem-se

$$f(x, y) = (\mathcal{B}_\sigma^t\, x^{\tilde{}}) \cdot \mathbf{A}\, \mathcal{B}_\sigma^t\, y^{\tilde{}} = x^{\tilde{}} \cdot \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t\, y^{\tilde{}} = x^{\tilde{}} \cdot \boldsymbol{\alpha}\, y^{\tilde{}}$$

Seja $\boldsymbol{\alpha}_k$ a matriz $\mathrm{GF}(2)^{n\times n}$ que selecciona a componente k de cada um dos elementos de $\boldsymbol{\alpha}$; seja z_k a componente correspondente de $f(x,y)$; então

$$z_k = x^{\tilde{}} \cdot \boldsymbol{\alpha}_k \, y^{\tilde{}}$$

Desta forma é possível calcular as componentes individuais de $f(x,y)$; expandindo obtém-se uma função booleana com monómios da forma $x_i \, y_j$.

$$z_k = \sum_{ij} \boldsymbol{\alpha}_k^{ij} \, x_i \, y_j$$

EXEMPLO 32: Um exemplo de tal função, em $\mathrm{GF}(2)^4$ seria definida pelo sistema de equações

$$\begin{bmatrix} z_0 = & x_0\,y_0 + x_1\,y_0 + x_1\,y_1 \\ z_1 = & x_1\,y_0 + x_1\,y_2 + x_2\,y_0 \\ z_2 = & x_3\,y_3 \\ z_3 = & x_2\,y_0 + x_0\,y_2 \end{bmatrix}$$

As matrizes $\boldsymbol{\alpha}_k$ são

$$\boldsymbol{\alpha}_0 = \begin{bmatrix} 1000 \\ 1100 \\ 0000 \\ 0000 \end{bmatrix} \qquad \boldsymbol{\alpha}_1 = \begin{bmatrix} 0000 \\ 1010 \\ 1000 \\ 0000 \end{bmatrix} \qquad \boldsymbol{\alpha}_2 = \begin{bmatrix} 0000 \\ 0000 \\ 0000 \\ 0001 \end{bmatrix} \qquad \boldsymbol{\alpha}_3 = \begin{bmatrix} 0010 \\ 0000 \\ 1000 \\ 0000 \end{bmatrix}$$

A matriz $\boldsymbol{\alpha}$ congrega estas componentes individuais em vectores de *bits*

$$\boldsymbol{\alpha} = \begin{bmatrix} 1000 & 0000 & 0001 & 0000 \\ 1100 & 1000 & 0100 & 0000 \\ 0101 & 0000 & 0000 & 0000 \\ 0000 & 0000 & 0000 & 0010 \end{bmatrix}$$

Conhecida a matriz $\boldsymbol{\alpha}$, é possível recuperar a matriz $\mathbf{A}$,

$$\boldsymbol{\alpha} = \mathcal{B}_\sigma \, \mathbf{A} \, \mathcal{B}_\sigma^t \quad \Longrightarrow \quad \mathbf{A} = (\mathcal{B}')_\sigma^t \, \boldsymbol{\alpha} \, (\mathcal{B}')_\sigma \quad \text{com} \quad (\mathcal{B}')_\sigma = \mathbf{T}^{-1} \, \mathcal{B}_\sigma$$

Tomemos o polinómio característico $\mathbf{c}[X] = X^4 + X + 1$ e a base polinomial de tipo 0 apresentada no exemplo 30 (página 235).

A base dual calculada nesse exemplo foi

$$\mathcal{B}' = \left\{ 1 + \beta^3 \ , \ \beta^2 \ , \ \beta \ , \ 1 \right\}$$

As matrizes $(\mathcal{B}')_\sigma$ e $\mathbf{A}$ que daí resultam são

$$(\mathcal{B}')_\sigma = \begin{bmatrix} 9 & 4 & 2 & 1 \\ D & 3 & 4 & 1 \\ E & 5 & 3 & 1 \\ B & 2 & 5 & 1 \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} A & 5 & B & A \\ 3 & D & D & B \\ 2 & D & 7 & 1 \\ 5 & 5 & 7 & D \end{bmatrix}$$

As **funções quadráticas** têm uma representação que deriva directamente da representação das funções bilineares: se for $z = g(x) = x_\sigma \cdot \mathbf{A} x_\sigma$ então teremos

$$z \;=\; \tilde{x} \cdot \boldsymbol{\alpha}\, \tilde{x}$$

com $\boldsymbol{\alpha} = \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t$. Deste modo, a componente z_k do resultado é

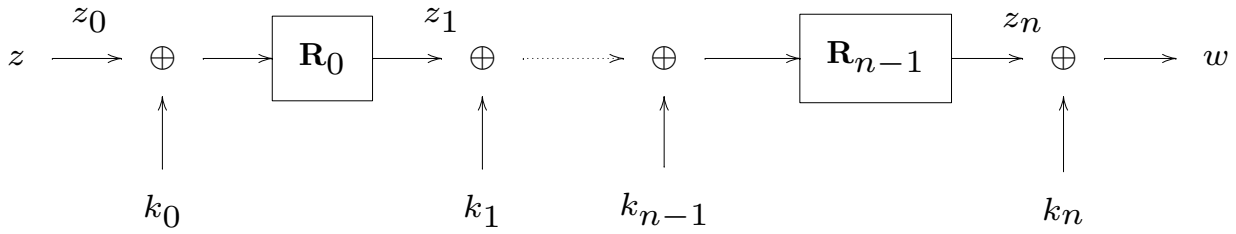
$$z_k \;=\; \sum_{ij} \boldsymbol{\alpha}_k^{ij}\, x_i\, x_j \tag{101}$$

em que $\boldsymbol{\alpha}_k^{ij}$ denota a componente de ordem k do elemento de índices i, j da matriz $\boldsymbol{\alpha}$.

O valor boolenao $\boldsymbol{\alpha}_k^{ij}$ determinam se o monómio $x_i\, x_j$ ocorre ou não na função boolena que calcula z_k. Assim, em termos de representações em $\mathrm{GF}(2)^n$, as formas quadráticas produzem (como seria de esperar) funções boolenas de grau 2; todos os monómios são da forma $x_i\, x_j$.

3.5 Criptoanálise Algébrica

As principais cifras simétricas[38] modernas têm a seguinte estrutura



(102)

A função **cifrar** ($w = f(k, z)$) processa-se de acordo com

1. As componentes essenciais são n SBoxes $\mathbf{R}_i$ invertíveis (os "**rounds**") e um **programador de chaves** (não representado na figura) que expande a **chave de controlo** k em $n+1$ **chaves de "round"** $k_0, k_1, .., k_n$.
2. Existe um **estado** z_i , inicializado com a entrada z , e que é recorrentemente alterado por duas transformações: uma soma $\oplus$ com a chave k_i e a aplicação de $\mathbf{R}_i$. Após o "round" final, a saída w obtém-se somando uma última chave k_n ao estado.
3. A relação de recorrência, fazendo variar $i \in 0..n-1$, é

$$z_0 = z \quad , \quad z_{i+1} = \mathbf{R}_i(z_i \oplus k_i) \qquad w = z_n \oplus k_n \tag{103}$$

[38] Cifras que usam a mesma chave para cifrar e decifrar.

A operação **decifrar** ($z = f^{-1}(k, w)$) tem a mesma estrutura com as alterações:

1. A novas chaves de "round" k'_i são as iniciais mas são apresentadas pela ordem inversa; isto é, $k'_i = k_{n-i}$
2. As SBoxs $\mathbf{R}_i$ são substitídas elas suas inversas algébricas e são apresentadas pela ordem inversa; isto é, $\mathbf{R}'_i = \mathbf{R}^{-1}_{n-i-1}$
3. O estado, representado por w_i , calcula-se fazendo variar $i \in 0..n-1$

$$w_0 = w \qquad w_{i+1} = \mathbf{R}'_i(w_i \oplus k'_i) \qquad z = w_n \oplus k'_n$$

Para ver que realmente estas duas funções são a inversa uma da outra basta notar que se preserva o invariante

$$w_i \oplus k'_i = z_{n-i} \qquad \text{ou} \qquad z_i \oplus k_i = w_{n-i}$$

Presupõe-se que uma mesma chave k é usada para cifrar um grande número de mensagens[39]. Presupõe-se também que são conhecidos alguns (poucos) pares mensagem+criptograma $\langle z_i, w_i \rangle$ gerados com essa chave.

Um ataque é uma algoritmo tratável que, a partir desta informação, descobre a chave k ou, equivalentemente, um algoritmo tratável que, a partir de um w arbitrário (distincto dos w_i conhecidos), descobre o elemento z

[39] Se a chave k fosse usada apenas uma vez então a cifra mais segura é simplesmente $w = z \oplus k$; esta é a chamada **segurança perfeita** de Shanon.

que cifrado com k reconstrói w. Numa cifra ideal a sua entropia está limitada pelo tamanho da chave em *bits*. Qualquer quebra em relação a este valor máximo é um ataque.

A segurança da cifra depende crucialmente do "design" dos "rounds" $\mathbf{R}_i$ e, normalmente, tem-se em vista dois objectivos principais:

- Não deve existir propagação de diferenças e, por isso, cada $\mathbf{R}_i$ deve manifestar um elevado grau de não-linearidade[40].

  Mais precisamente deve existir uma boa **mistura** entre a chave e a entrada: não deve ser possível "separar", à saída, os efeios individuais de cada um destes items.
- Não devem existir correlações entre visões particulares (paridades) da entrada e da saída. A existência de tais correlações implicaria a existência de relações priveligiadas entre alguns *bits* da entrada com alguns *bits* da saída, o que equivale a uma quebra na entropia da cifra.

  Deve existir uma boa **difusão** da influência de qualquer *bit* da entrada por toda a saída.

Idealmente os objectivos de "boa mistura" e "boa difusão" deveriam ser assegurados por um "design" único das SBoxes. No entanto, se atendermos que é necessário assegurar também boas propriedades computacionais numa função não-linear invertivel, este "design único" torna-se muito difícil.

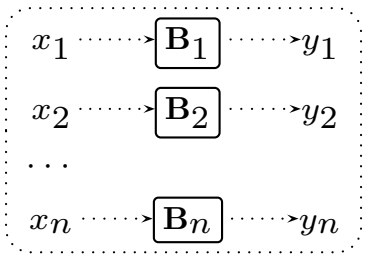
[40] Como cada "round" tem de ser uma função algebricamente invertível (para ser possível decifrar) se fosse linear as técnicas usuais de álgebra linear permitiriam um ataque trivial.

Por isso é usual decompor as SBox em componentes cada uma das quais com um objectivo próprio. Por exemplo é usual escolher uma componente linear com propriedades óptimas em termos de difusão mas muito má (por ser linear) em termos de mistura. Outro tipo de componente são as "bricklayer functions" (que veremos em seguida) que têm boas propriedades em termos de mistura e eficiência computacional mas que apresentam elevadas correlações e, por isso, são más em termos de difusão.

Funções "bricklayer"

A entrada x e a saída y são vectores de $n \times s$ *bits* agrupados em n blocos de s *bits*.

Cada bloco é transformado independentemente dos restantes por uma $s \times s$-SBox $\mathbf{B}_i$. A SBox global $\mathbf{B}$ (de tamanho $(n\,s) \times (n\,s)$) obtém-se fraccionando a entrada x em blocos x_i, aplicando a SBox $\mathbf{B}_i$ ao bloco x_i e agregando os resultados parciais y_i num único vector y.



$$y = \mathbf{B}(x) \qquad y, x \in (\mathbb{B}^s)^n$$
$$y = (y_1, \ldots, y_n) \qquad x = (x_1, \ldots, x_n)$$
$$y_i = \mathbf{B}_i(x_i) \qquad x_i, y_i \in \mathbb{B}^s$$

Se todas as SBoxes $\mathbf{B}_i$ forem invertíveis, também $\mathbf{B}$ é invertível. Adicionalmente $\mathbf{B}^{-1}$ é também uma função "bricklayer" determinada pelas n inversas $\mathbf{B}_i^{-1}$.

A vantagem destas funções reside na sua eficiência computacional; isto deriva do facto de lidar com SBox de uma dimensão s que é muito menor do que a simensão $(n\, s)$ exigida para a SBox global.

Para além de ausências de diferenças e correlações (ligadas aos objectivos de "boa mistura" e "boa difusão"), surge um outro tipo de objectivo que deriva da existência de uma outra forma de ataque.

Considere-se, de novo, as relações em (103) que traduzem as relações essenciais entre os vários items de informação que caracterizam a cifra. Delas resulta um sistema de equações com incógnitas z_i e k_i conhecida a entrada z e o criptograma w.

$$\begin{cases} z_0 & = z \\ z_n \oplus k_n & = w \\ z_{i+1} \oplus \mathbf{R}_i(z_i \oplus k_i) & = 0 \qquad i \in 0..n-1 \end{cases} \tag{104}$$

Uma tentativa de resolver estas equações esbarra com a forma complicada das SBoxes $\mathbf{R}_i$. No entanto é geralmente possível inserir estas equações numa estrutura algébrica adequada (num corpo finito, específicamente) de tal modo que as equações possam ser escritas

$$\begin{cases} z_0 & = z \\ z_n + k_n & = w \\ \mathbf{F}_i(z_{i+1}, z_i + k_i) & = 0 \qquad i \in 0..n-1 \end{cases} \tag{105}$$

em que as funções $\mathbf{F}_i(\cdot,\cdot)$ têm uma estrutura algebricamente muito mais símples do que os $\mathbf{R}_i(\cdot)$.

EXEMPLO 33: O exemplo paradigmático é SBox AES determinada por $y = x^{254}$ ou $y = x^{-1}$ (quando $x \neq 0$) que, explicitamente, é uma função não-linear bastante complexa mas que, implicitamente, se escreve como uma forma bilinear

$$x \cdot y = 1 \quad , \; x \neq 0$$

A forma indicada em (99) e (100) (página 245) para as funções bileneares permite-nos ver esta forma gerada pela matrizes

$$\mathbf{A} = \begin{bmatrix} 10000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \end{bmatrix} \qquad \boldsymbol{\alpha} = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 \\ 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B \\ 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 \\ 08 & 10 & 20 & 40 & 80 & 1B & 36 & 6C \\ 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 \\ 20 & 40 & 80 & 1B & 36 & 6C & D8 & AB \\ 40 & 80 & 1B & 36 & 6C & D8 & AB & 4D \\ 80 & 1B & 36 & 6C & D8 & AB & 4D & 9A \end{bmatrix}$$

Uma vez mais, o elementos genérico a em ambas as matrizes é descrito pela palavra de bits $a^{\tilde{}}$ em notação hexadecimal.

Recordemos, nesta representação da forma bilinear $x \cdot y$, a equação $1 = x \cdot y$ se descreve como

$$1 \;=\; \sum_{i,j} \boldsymbol{\alpha}_{ij} \cdot \; x_i \, y_j$$

É possível olhar para esta representação de uma outra forma: considera-mos, por momentos, os pares $x_i \, y_j$ como uma única incógnita e a matriz $\boldsymbol{\alpha}$ como um vector de coeficientes. Vamos chamar X ao vector destas "incógnitas duplas" e continuamos a representar por $\boldsymbol{\alpha}$ a forma linearizada da matriz.

Com $64 = 8 \times 8$ incógnitas booleanas a equação é escrita

$$1 \;=\; \boldsymbol{\alpha} \cdot X \tag{106}$$

Esta equação base dá origem e outras; nomeadamente

$$x = x^2 \, y, \quad x^2 = x^4 \, y^2, \quad x^4 = x^8 \, y^4, \quad \cdots x^{64} = x^{128} \, y^{64}$$

e a versão dual para y

$$y = x \, y^2, \quad y^2 = x^2 \, y^4, \quad \cdots \quad y^{64} = x^{64} \, y^{128}$$

Cada uma destas duas sequências é gerada a partir da equação base ($x = x^2 \cdot y$ ou $y = x \cdot y^2$) por aplicação sucessiva da transformação linear $\sigma : z \mapsto z^2$ a ambos os lados da equação; as equações resultantes são, portanto, linearmente dependentes.

De facto σ gera transformações lineares nos vectores $x\tilde{}$ e $y\tilde{}$ que permitem ir transformando uma equação na seguinte.

Portanto temos três equações em GF(2^8) (das quais resultam 3×8 equaçoes em GF(2)) que podem ou não ser linearmente independentes e que derivam de

$$x \cdot y = 1 \quad (x \neq 0) \qquad x^2\, y + x = 0 \qquad x\, y^2 + y = 0$$

Existem ainda outras formas bilineares; por exemplo, multiplicando ambos os lados da 2ª equação por x^2 e ambos os lados da 3ª equação por y^2 constrói-se

$$x^4\, y + x^3 = 0 \qquad x\, y^4 + y^3 = 0$$

As formas bilineares em todas estas 5 equações são por matrizes¡ $\mathbf{A}$ apropriadas que vão dar origem a matrizes $\boldsymbol{\alpha}$.

Os restnates constituintes dos lados esquerdos das equações (para além das formas bilineares) são monímios x , y , que são formas lineares, ou os monómios x^3 , y^3 , que são formas quadráticas.

As formas bilineares têm exactamente a mesma forma que (106) e dão origem a somas de monómios do tipo $x_i\, y_j$ calculadas por $x\tilde{} \cdot \boldsymbol{\alpha}\, y\tilde{} = \sum_{ij} \boldsymbol{\alpha}_{ij}\, x_i\, y_j$.

Para a forma $x^2\, y$ tem-se a matriz $\mathbf{A}'$ com $\mathbf{A}'_{10} = 1$ e $\mathbf{A}'_{ij} = 0$ para os restantes índices.

A forma $x\,y^2$ é determinada por uma matriz $(\mathbf{A}')^t$ que é transposta da anterior.

A forma $x^4\,y$ é definida pela matriz $\mathbf{A}''$ com $\mathbf{A}''_{20} = 1$ e $\mathbf{A}'_{ij} = 0$ para os restantes índices. A forma $x\,y^4$ é definida pela transposta desta matriz.

As matrizes $\boldsymbol{\alpha}$ correspondentes serão

$$\boldsymbol{\alpha}' = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 \\ 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 \\ 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 \\ 40 & 80 & 1B & 36 & 6C & D8 & AB & 4D \\ 1B & 36 & 6C & D8 & AB & 4D & 9A & 2F \\ 6C & D8 & AB & 4D & 9A & 2F & 5E & BC \\ AB & 4D & 9A & 2F & 5E & BC & 63 & C6 \\ 9A & 2F & 5E & BC & 63 & C6 & 97 & 35 \end{bmatrix}$$

$$\boldsymbol{\alpha}'' = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 \\ 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 \\ 1B & 36 & 6C & D8 & AB & 4D & 9A & 2F \\ AB & 4D & 9A & 2F & 5E & BC & 63 & C6 \\ 5E & BC & 63 & C6 & 97 & 35 & 6A & D4 \\ 97 & 35 & 6A & D4 & B3 & 7D & FA & EF \\ B3 & 7D & FA & EF & C5 & 91 & 39 & 72 \\ C5 & 91 & 39 & 72 & E4 & D3 & BD & 61 \end{bmatrix}$$

Juntanto todas estes vectors $\boldsymbol{\alpha}$ como linhas de uma matriz constrói-se uma nova matriz com $64 = 8 \times 8$ colunas

e $5 \times 8 = 40$ linhas. Daqui resulta um sistema de equações "aparentemente linear" nas incógnitas X

$$\begin{cases} 1 & = \boldsymbol{\alpha} \cdot X \\ x & = \boldsymbol{\alpha}' \cdot X \\ y & = (\boldsymbol{\alpha}')^t \cdot X \\ x^3 & = \boldsymbol{\alpha}'' \cdot X \\ y^3 & = (\boldsymbol{\alpha}'')^t \cdot X \end{cases} \quad \Longrightarrow \quad \begin{bmatrix} 1 \\ x \\ y \\ x^3 \\ y^3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}' \\ (\boldsymbol{\alpha}')^t \\ \boldsymbol{\alpha}'' \\ (\boldsymbol{\alpha}'')^t \end{bmatrix} X$$

Não se trata realmente de um sistema de equações lineares porque os $x\tilde{}, y\tilde{}$ aparecem por si no lado esquerdo das equações ao mesmo tempo que aparecem nas incógnitas X .

A questão essencial está no número de equações que são linearmente independentes e sobre a forma como estes sistemas podem ser resolvidos.

No exemplo anterior resultaram 5 formas bilineares

$$x \cdot y \qquad x^2 \cdot y \qquad x^4 \cdot y \qquad x \cdot y^2 \qquad x \cdot y^4$$

para as quais foi possível construir equações lineares no binómios $x_i y_j$. Levantam-se imediatamente duas questões:

1. As equações resultantes são linearmente independentes?

2. Será possível acrescentar outras formas (por exemplo, $x^8 \cdot y$) que gerem equações linearmente independentes das existentes?

4.Criptografia Simétrica

Neste capítulo procuraremos dar uma visão introdutória da família de técnicas criptográficas cujo uso requer o conhecimento, partilhado por todos os legítimos intervenientes, de uma única chave. Esta família de técnicas designa-se, genericamente, por **criptografia simétrica**.

Dada a limitação no número de chaves, a gama de técnicas que é possível definir é bastante limitada; essencialmente são cifras, funções de "hash" e combinações destes dois tipos base. Em contrapartida estas técnicas fornecem:

. **Elevada eficiência computacional**
Um parâmetro fundamental para aferição de uma técnica criptográfica é o seu **"throughput"**; isto é, a quantidade de informação que consegue processar numa unidade de tempo. As técnicas simétricas são as que possuem o maior "throughput" tendo capacidade para processar, em tempo real, sinais de áudio e vídeo. Nomeadamente permitem implementações em "hardware" dedicado, o que é factor essencial para aumentar o "throughput". contribui fortemente para aumentar a eficiência computacional.

. **Elevada eficiência de segurança**
A incerteza quando à viabilidade de um ataque com sucesso, expresso em bits, mede o *grau de segurança* de uma

técnica criptográfica. Nas técnicas simétricas a incerteza está sempre limitada pelo comprimento da chave[41] mas pode, no entanto, ser menor; a diferença entre o tamanho da chave e o grau de segurança da técnica, mede a *eficiência* de segurança da técnica.

As técnicas simétricas têm uma elevada eficiência de segurança já que (pelo menos, nas boa técnicas) o grau de segurança está muito próximo do tamanho da chave; por exemplo, mesma na sua versão mais limitada, o AES usa chaves de 128 bits e tem um grau de segurança muito próximo desse valor. Em contraste uma cifra assimétrica como o RSA precisa de uma chave de 1024 bits para ter um grau de segurança da ordem dos 80 bits.

. **Elevada flexibilidade e robustez**

É frequente que vários contextos de processamento de informação imponham restrições e condicionalismos de segurança específicos a uma mesma técnica criptográfica. As técnicas simétricas têm a capacidade de se adaptar a estas condições através de modos de funcionamento específicos. Também têm a flexibilidade de se metamorfosear adaptando-se às necessidades do contexto; por exemplo, as mesmas primitivas criptográficas básicas podem ser usadas para cifras de blocos, cifras sequenciais e funções de "hash".

Outro aspecto essencial é a sua robustez a ataques; quando existe a hipótese de um ataque é possível, quase sempre, efectuar ligeiras modificações à técnica básica de modo a o tornar inviável. A evolução do DES para o 3DES, que estendeu efectivamente o tempo de vida útil do DES para 30 anos, foi uma resposta simples ao

[41] Existe sempre um ataque trivial, o chamado **ataque por força bruta**, que consiste em percorrer todo o espaço de chaves até que uma delas satisfaça uma determinada condição; por exemplo, verificar se o criptograma obtido com a chave teste coincide com um determinado valor observado.

aparecimento das técnicas de criptoanálise; do mesmo modo, quando foi detectado um ataque ao SHA logo no início da sua existência, uma ligeira alteração conduziu ao SHA-1 que é, já há 15 anos, a função de "hash" mais popular.

Por estas razões as técnicas simétricas são os alicerces de todo o edifício de segurança num sistema de informação. Especificações de segurança complexas e subtis exigem, normalmente, técnicas assimétricas; no entanto, dado que as técnicas assimétricas, devido à sua elevada complexidade computacional, conseguem processar apenas pequenas quantidades de informação (chaves, "hashs", etc.), o tratamento de elevadas quantidades de dados exige sempre uma combinação de uma técnica assimétrica com uma técnica simétrica capaz de processar o volume de dados.

4.1 Nomenclatura de Cifras

Permutação

Uma permutação (também designada por **substituição simples**) é uma função bijectiva de domínio finito; isto é, uma função $f : \mathbb{B}^n \longrightarrow \mathbb{B}^n$ que seja injectiva.

As permutações são a construção invertível mais simples e estão na base das cifras ditas **determinísticas**: a permutação, aplicada duas vezes, ao mesmo "input" produz sempre o mesmo "output".

Transformação holomórfica

Uma função $h : \mathbb{N} \longrightarrow \mathbb{N}$ tal que a família de conjuntos $\left\{h^{-1}(n)\right\}_{n\in\mathbb{N}}$ é uma cobertura disjunta de $\mathbb{N}$: isto é, os conjuntos são disjuntos dois a dois e a sua união cobre todo $\mathbb{N}$.

A transformação é usada para construção das cifras homomóficas: um algoritmo probabilístico que, sob "input" x, produz um $y \in h^{-1}(x)$. A função h é usada para decifrar o criptograma y.

Cifras por blocos

São primitivas criptográficas representáveis por funções boolenas $f : \mathbb{B}^t \times \mathbb{B}^n \longrightarrow \mathbb{B}^n$ tais que, para cada $k \in \mathbb{B}^t$, a função $f(k, \cdot) : \mathbb{B}^n \longrightarrow \mathbb{B}^n$ é uma permutação. Os inteiros n e t são os comprimentos, respectivamente, do **bloco** e da **chave**.

Cifras iteradas

São cifras por blocos em que a função f é construída por composição de um número finito p de cifras por blocos $r_1, r_2, ..., r_p : \mathbb{B}^t \times \mathbb{B}^n \longrightarrow \mathbb{B}^n$, designadas por **"rounds"**, da seguinte forma:

1. Existe uma função $g : \mathbb{B}^t \longrightarrow \left(\mathbb{B}^t\right)^p$ chamada **programador de chaves** que, a cada chave $k \in \mathbb{B}^t$, associa um vector com p chaves $g(k) = (k_1, k_2, ..., k_p)$ em $\mathbb{B}^t$.
2. Para uma determinada **chave** k e **texto** x, o **criptograma** correspondente $y = f(x, k)$ é calculado através da sequência
$$y_0 = x \quad , \quad y_i = r_i(k_i, y_{i-1}) \quad i = 1, ..., t \quad , \quad y = y_t$$
em que $k_i = g(k)_i$.

Desta forma a função f é definida por uma iteração de funções $f_0, f_1, \cdots, f_t$

$$f_0(x, k) = x \qquad , \qquad f_i(x, k) = r_i(f_{i-1}(x, k), g(k)_i) \quad i = 1, \cdots, t \qquad , \qquad f = f_t$$

Numa cifra iterada o acto de decifrar é também iterado: dado o criptograma y, a sequência y_i é reconstruída iterativamente do fim para o princípio,

$$y_t = y \qquad , \qquad y_{i-1} = r_i^{-1}(y_i, k_i) \quad i = t, ..., 1 \qquad , \qquad x = y_0$$

Cifras sequenciais ("stream ciphers"):

São primitivas criptográficas descritas por funcionais $f : \mathbb{B}^t \times \mathbb{B}^\infty \longrightarrow \mathbb{B}^\infty$ tais que:

(i) Para toda a chave k e todo i, o valor $f(k,u) \restriction i$ depende apenas de $u \restriction i$; isto é, se for $u \restriction i = v \restriction i$, então será $f(k,u) \restriction i = f(k,v) \restriction i$.

(ii) Para toda a chave k e todo i, a função $x \in \mathbb{B}^i \mapsto f(k, \uparrow x) \restriction i$ é uma permutação.

Este conceito de cifra sequencial efectua um processamento sequencial de mensagens *orientado ao bit*. Em cada estado constrói-se o bit de ordem i do criptograma a partir dos bits do texto de ordem $j \leq i$.

Nomeadamente, uma forma comum de cifra sequencial orientada ao bit, é descrita por uma função f da forma

$$f(k,u) \;=\; s(k) \oplus u \tag{107}$$

sendo $s : \mathbb{B}^m \longrightarrow \mathbb{B}^\infty$ um **gerador pseudo-aleatório** de bits.

Neste caso o bit de ordem i do criptograma depende apenas do bit com a mesma ordem i do texto. Cifras com esta propriedade designam-se por **mono-alfabéticas**. Cifras onde o bit de ordem i depende efectivamente de todos os bits do texto de ordem $j \leq i$ e, eventualmente, de alguns de ordem $j > i$, designam-se por **poli-alfabéticas**.

Cifras sequenciais por blocos

Por vezes é necessário generalizar o conceito de cifra sequencial para o de uma cifra sequencial *orientada ao bloco*. Neste modelo tanto o texto como o criptograma são vistos como sequências infinitas de blocos de tamanho fixo n; o cálculo do bloco de ordem i do criptograma recorre ao conhecimento de todos os blocos de ordem $j \leq i$ do texto.

Formalmente, uma cifra sequencial orientada a blocos de tamanho n, com chaves de tamanho m, é uma funcional $f : \mathbb{B}^t \times \mathbb{B}^* \longrightarrow \mathbb{B}^*$ tal que:

(i) Para toda a chave k e todo i múltiplo de n, se for $u \restriction i = v \restriction i$, então $f(k,u) \restriction i = f(k,v) \restriction i$.

(ii) Para toda a chave k e todo i múltiplo de n, a função $x \in \mathbb{B}^i \mapsto f(k, \uparrow x) \restriction i$ é uma permutação.

Essencialmente, em relação à definição inicial de cifra sequencial, nesta definição substitui-se o quantificador "*todo* i" por "*todo* i *múltiplo de* n". Isto implica que um bit do criptograma de ordem i arbitrária pode depender de bits do texto que têm ordems superiores.

Uma cifra sequencial mono-alfabética orientada ao bloco de tamanho n, pode ser definida à custa de uma qualquer cifra de blocos $h : \mathbb{B}^t \times \mathbb{B}^n \longrightarrow \mathbb{B}^n$ e de um gerador de chaves $s : \mathbb{B}^t \longrightarrow (\mathbb{B}^t)^\infty$.

Para tal texto e o criptograma são sequências de blocos e define-se a funcional $f : \mathbb{B}^t \times (\mathbb{B}^n)^\infty \longrightarrow (\mathbb{B}^n)^\infty$ por

$$f(k\,,\,v) \quad = \quad h(s_0, x_0)\; h(s_1, x_1)\; \cdots\; h(s_i, x_i)\; \cdots \tag{108}$$

em que $v = x_0\, x_1 \cdots x_i \cdots$ é a sequência de blocos de "input" e $s(k) = s_0\, s_1 \cdots s_i \cdots$ é a sequência de chaves geradas a partir de k. Ou seja, o bloco de ordem i no criptograma é determinado por

$$y_i \quad = \quad f(k, v)_i \quad = \quad h(s_i, x_i) \tag{109}$$

Para construir uma cifra poli-alfabética necessita-se de uma estrutura um pouco mais complexa. Uma forma frequente, chamada **cifra com "trail" l** usa o mesmo gerador de chaves $s : \mathbb{B}^t \to (\mathbb{B}^t)^\infty$ da cifra mono-alfabética, mas introduz uma nova função $H : \mathbb{B}^* \times \mathbb{B}^n \to \mathbb{B}^n$ que é invertível no 2º argumento; isto é, para todo y, $H(y, \cdot)$ é uma permutação.

Então a sequência $\{y_i\}$ de blocos do criptograma é gerada por

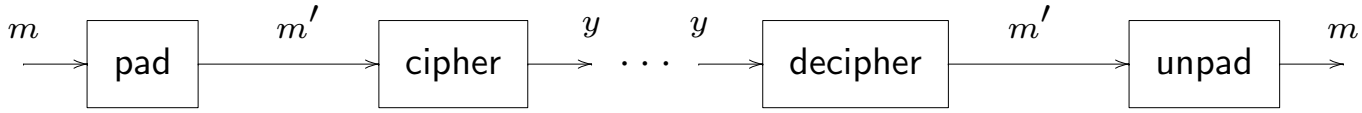
$$y_i \;=\; h(s_i, u_i) \qquad \text{com} \qquad u_i \;=\; H(y_{i-1} \| \cdots \| y_{i-l}\,,\; x_i) \tag{110}$$

Os l blocos do criptograma, anteriores a i, funcionam como chave que, com H, cifra x_i. O valor resultante u_i é, posterormente, cifrado (usado h) com a chave s_i gerada a partir de k.

Este tipo de construção levanta a questão de se saber como calcular os primeiro l blocos do criptograma. Note-se que, para calcular y_0, por exemplo, precisariamos de conhecer um "pseudo"-bloco y_{-l}. O problema resolve-se introduzindo l constantes que, independentemente do texto v, funcionam como blocos $y_{-l} \cdots y_{-1}$. Estas constantes são designadas por **vectores iniciais (iv)**.

"Padding"

Quando uma mensagem m, de comprimento arbitrário, está destinada a ser cifrada por uma cifra de blocos de tamanho fixo n, há necessidade de embeber a mensagem numa outra mensagem m' cujo comprimento é um múltiplo exacto do tamanho do bloco. Se o tamanho da nova mensagem m' for $t \cdot n$, então pode-se cifrar m' invocando t vezes a cifra de bloco n.



Este esquema exige uma função injectiva para embeber m em m' e, no final, uma função que recupere m sem ambiguidades. O mecanismo de "padding" exige duas funções:

1. Uma função injectiva pad : $\mathbb{B}^* \longrightarrow \mathbb{B}^*$ tal que, para todo $u \in \mathbb{B}^*$, $|\text{pad}(u)| = t \cdot n$, para algum $t > |u|/n$.
2. Uma função parcial unpad : $\mathbb{B}^* \rightharpoonup \mathbb{B}^*$ tal que $\text{unpad}(\text{pad}(u)) = u$, para todo u.

Modos de Cifras por blocos

Considere-se o problema de cifrar bit-strings infinitas usando uma cifra de blocos $h : \mathbb{B}^t \times \mathbb{B}^n \longrightarrow \mathbb{B}^n$ com chaves de tamanho k e blocos de tamanho n.

A forma mais simples de construir uma tal cifra, consiste em ver o "input" e o "output" como strings infinitas de blocos. e usar as construções (109) e (110) escolhendo formas particulares do gerador de chaves s e do "trail" H.

Formalmente queremos construir, usando h, uma funcional $f : \mathbb{B}^t \times (\mathbb{B}^n)^\infty \to (\mathbb{B}^n)^\infty$ que se comporte como uma cifra sequencial orientada a um bloco de tamanho n.

Numa cifra da forma (108) e (109) considere-se um gerador de chaves que se limita a repetir a chave k *ad infinitum*; isto é $s(k) = k^\infty$. Neste caso diz-se que a cifra h está em modo **electronic code book (ecb)**. No modo **ecb**, o bloco de orden i no criptograma é determinado por

$$y_i = h(k, x_i)$$

Em alternativa pode-se usar uma construção da forma (110), com o mesmo gerador de chaves $s(k) = k^\infty$, e um "trail" de tamanho 1, dado por $H(y_{i-1}, x_i) = y_{i-1} \oplus x_i$. Nesta construção diz-se que a cifra h está em modo **cipher-block chaining (cbc)**; bloco y_i, no modo **cbc**, é calculado por

$$y_i = h(k, x_i \oplus y_{i-1})$$

Uma abordagem alternativa consiste em ver o "input" e o "output" como "streams" de blocos com um tamanho m que é menor do que o tamnaho n do bloco usado por h. Os blocos y_i são gerados pela dupla recorrência

$$z_i = H(y_{i-1} \| z_{i-1}) \quad , \quad s_i = h(k, z_i) \restriction m \quad , \quad y_i = x_i \oplus s_i$$

em que $H : \mathbb{B}^* \to \mathbb{B}^n$ é uma qualquer função de "hash". Modos baseados nesta estrutura chamam-se **"feedback modes"**.

4.2 Arquitectura de Cifras por Blocos

Na maioria das cifras por blocos iteradas, os diversos "rounds", com eventual excepção do primeiro e do último, são iguais. São também formados por algumas componentes "standard". Dado que uma característica essencial das cifras por blocos é a sua eficiência computacional, essas componentes têm de ser de implementação muito simples.

O desafio essencial de qualquer arquitectura de cifra é o de construir um edifício extremamente seguro a partir de componentes muito simples e muito eficientes. Individualmente, cada uma das componentes pode não ser particularmente segura, mas a sua combinação produz a segurança pretedida.

A seguir indicam-se algumas das componentes básicas que ocorrem na construção de cifras.

Branqueamento ("**whitening**")

Sabe-se que, se $k \in \mathbb{B}^n$ for, nalgum sentido do termo, aleatório então todo $x \oplus k$ é também aleatório. Por isso a cifra $f(k, x) = x \oplus k$ dá segurança perfeita para ataques sem texto conhecido; não é possível conhecer x a partir de $y = x \oplus k$ sem conhecer a chave k.

No entanto, num ataque de texto conhecido (onde tanto x como y são conhecidos) determinar a chave k é trivial; basta fazer $k = y \oplus x$.

A cifra $f(k,x) \doteq x \oplus k$ é extremamente eficiente já que o `xor` de duas palavras de bits é directamente implementado num ciclo de máquina.

Funções Lineares

O "branqueamento" $x \oplus k$ determina, para cada k, uma função linear. Como sabemos, uma função $f : \mathbb{B}^n \longrightarrow \mathbb{B}^n$ é linear se verifica, para todo $x, y \in \mathbb{B}^n$ e $b \in \mathbb{B}$

$$f(x \oplus y) \;=\; f(x) \oplus f(y) \qquad , \qquad f(b\,x) \;=\; b\,f(x)$$

Funções lineares têm muitas representações; nomeadamente em $\mathrm{GF}(2)$ a função f é representada por uma matriz de bits $F \in \mathrm{GF}(2)^{n\times n}$; nessa representação tem-se $f(x) = F\,x$ usando as operações de álgebra matricial no corpo $\mathrm{GF}(2)$.

Também nesta representação, se a função $f(\cdot)$ for injectiva, então a matriz F tem uma inversa F^{-1} que é a representação matricial da função inversa f^{-1}

Normalmente uma função linear f é de implementação mais eficiente do que uma não linear e algumas formas particulares de funções lineares (como o "branqueamento") são mesmo extremamente eficientes.

Uma cifra $f(k,x)$, em que $f(k,\cdot)$ fosse linear para algum k, é vulnerável a um ataque muito simples. Sopunhamos que se conhecem vários criptogramas $y_i = f(k,x_i)$ e os respectivos textos x_i; seja $x = a_1\,x_1 \oplus a_2\,x_2 \cdots \oplus a_l\,x_l$,

com $a_i \in \mathrm{GF}(2)$, uma qualquer combinação linear dos diversos x_i. Então, mesmo sem conhecer a chave k, é possível calcular $y = f(k, x)$ usando a mesma combinação linear nos criptogramas; isto é, $y = a_1\, y_1 \oplus a_2\, y_2 \cdots \oplus a_l\, y_l$.

Nomeadamente, se os x_i formarem uma base do espaço vectorial $\mathrm{GF}(2)^n$, então o conhecimento de n pares (x_i, y_i) permite construir o criptograma de todos os 2^n possíveis pares (texto,criptograma), sem necessidade de conhecer a chave.

No entanto as funções lineares têm uma grande importância como componente de cifras porque uma construção da forma $y = f(x)$, sendo f linear, permite preservar a aleatoriedade de x em y.

□

Alguns exemplos de funções lineares de implementação muito eficiente.

Linear Feedback Shift Register (LFSR)

Sejam x_i, y_i os bits de ordem i dos vectores $x, y \in \mathrm{GF}(2)^n$; defina-se

$$y_0 = a_0\, x_0 + a_1\, x_1 + \cdots + a_{n-1}\, x_{n-1} \quad , \quad y_i = x_{i-1} \quad \text{para} \quad i = 1, \cdots, n-1$$

Esta transformação traduz-se no produto $y = F\, x$ em que a matriz F verifica $F_{1,i} = a_i$, $F_{i,i-1} = 1$ e $F_{i,j} = 0$ para os restantes índices. O vector $(a_0, \cdots, a_{n-1}) \in \mathrm{GF}(2)^n$ desina-se por **vector característico** do LFSR.

LFSR são uma componente muito importante de cifras sequenciais e têm a vantagem de ser implementadas em poucos ciclos máquina e directamente implementáveis em *hardware*.

Foram, durante muito tempo, usadas como componente de geradores de bit-strings pseudo-aleatórios. Um exemplo simples usa um LFSR definido por uma matriz F, do tipo atrás descrito, e a função traço $\mathrm{tr}(\cdot)$.

Gera-se uma sequência de bits b_i a partir da recorrência linear

$$x_0 \;=\; k \qquad , \qquad x_i \;=\; F\,x_{i-1} \quad \text{para } i = 1, \cdots \qquad , \quad b_i = \mathrm{tr}(x_i)$$

Esta construção pode-se generalizar usando t LFSR´s definidos por matrizes $F_1, \cdot, F_t$ e uma função booleana não-linear $\sigma : \mathrm{GF}(2)^t \longrightarrow \mathrm{GF}(2)$. Cada LFSR actua sobre um estado próprio e os traços dos diversos estados são combinados pela função σ.

A sequência de bits b_i é gerado pela recorrência

$$\begin{cases} x_{0,j} \;=\; k_j & j = 1, \cdots, t \\ x_{i,j} \;=\; F_j\, x_{i-1,j} & j = 1, \cdots, t \qquad i = 1, \cdots \\ b_i \;=\; \sigma(\mathrm{tr}(x_{i,1}), \cdots, \mathrm{tr}(x_{i,t})) \end{cases}$$

Transformações Pseudo-Hadamard (pHT)

Pode-se ver palavras $x, y \in \mathbb{B}^{2n}$ como pares de inteiros $(x_1, x_2), (y_1, y_2) \in \mathbb{Z}_n \times \mathbb{Z}_n$.

A transformação PHT define-se por $(y_1, y_2) = (2\,x_1 + x_2, x_1 + x_2)$ em que a adição $+$ e a multiplicação são feitas em $\mathbb{Z}_n$.

Assim $\begin{bmatrix} y_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ e esta transformação é, normalmente, efectuada num único ciclo máquina.

□

Vamos agora examinar funções não-lineares $f\mathbb{B}^n \longrightarrow \mathbb{B}^n$.

"Substitution boxes"

Funções não lineares genéricas $f : \mathbb{B}^n \longrightarrow \mathbb{B}^m$ são, normalmente, designadas por **substituion boxes** (ou, simplesmente, **S-boxes**).

Para um mesmo tamanho de bloco n, uma S-box $n \times n$ é muito mais difícil de implementar que uma função linear com a mesma dimensão. Por isso S-boxes são normalmente usadas com pequenos valores de n; enquanto que é relativamente simples implementar uma transformação linear para um bloco de 128 bits ou superior, as S-boxes não ultrapassam os 16 bits. Aliás, quase sempre as S-boxes são de 8 ou menos bits.

Quando n é pequeno é fácil implementar uma função f genérica como uma tabela em memória ROM; constrói-se uma memória com n linhas para o endereço de célula e cada célula contém m bits. Usando x como endereço de acesso à memória, e supondo que a célula de endereço x contém o valor $f(x)$, então, com um simples acesso à memória, calcula-se $f(x)$.

Obviamente que, atendendo à pequena capacidade de memória dos dispositivos que têm de executar estas cifras, esta estratégia só é viável se n for pequeno; por exemplo, uma S-box 8×8 requer `256 bytes` de memória; se for 16×16 requer `128 Kbytes`; se for 24×24 requer `48 Mbytes`; etc.

□

Quando são realmente necessárias funções não-lineares com elevados valores de n, então as únicas implementações possíveis são aquelas que aproveitam a forma particular destas funções.

Funções "bricklayer"

Considere-se palavras $x, y \in \mathbb{B}^n$ e considere-se um valor l que é divide n; seja $p = n/l$. Pode-se representar cada um dos elementos de $\mathbb{B}^n$ como um vector com p componentes em $\mathbb{B}^l$; isto é, usa-se o isomorfismos $\mathbb{B}^n \sim \left(\mathbb{B}^l\right)^p$.

Considere-se agora p S-boxes $f_1, \cdots, f_p \colon \mathbb{B}^l \to \mathbb{B}^l$. Então, pode-se definir uma função não linear por

$$f \;:\; x_1 \| x_2 \| \cdots \| x_p \quad \mapsto \quad f_1(x_1) \| f_2(x_2) \| \cdots \| f_p(x_p) \tag{111}$$

Neste tipo de função as S-boxes têm dimensão $l \times l$ e podem ser razoavelmente implementáveis. Cada uma destas transformações actua apenas sob uma das componentes do "input" x e produz a componente, com a mesma ordem, no "output" y. O "output" é construído bloco a bloco e não é de estranhar que este tipo de funções se designe por "bricklayer".

Suponhemos que f é esta função "bricklayer" e que um atacante conhece x e y e procura determinar k tal que $y = f(x \oplus k)$. Normalmente, para uma função não-linear bem escolhida, este problema teria complexidade exponencial com n. Porém, para uma função "bricklayer", o problema resume-se a p problemas diferentes de tamanho l. Ou seja, passamos de uma complexidade $O((2^l)^p)$ para uma complexidade $O(p\,2^l)$.

Por este motivo, funções "bricklayer" não podem ser usadas isoladamente como "rounds" de cifras. O seu "output" têm de ser pós-processado (normalmente por uma transformação linear) de modo que, no final, se tenha uma função $n \times n$ cujo comportamento se aproxime de uma função ideal.

Circuitos de Feistel

Uma dificuldade importante na escolha de funções não-lineares em "rounds" de cifras está no cumprimento do requisito de que os "rounds" têm de ser invertíveis. Construir e implementar eficientemente funções não-lineares invertíveis é um problema difícil.

Com o DES (que serviu de standard criptográfico no sistema financeiro durante quase 30 anos) surgiu uma construção que tomou o nome do principal autor do DES: os *circuitos de Feistel*.

Vamos tomar uma função $h\colon \mathbb{B}^n \longrightarrow \mathbb{B}^n$, não-linear mas não necessaramente invertível. Vamos contruir uma função $f\colon \mathbb{B}^{2n} \longrightarrow \mathbb{B}^{2n}$ do seguinte modo

$$f \quad : \quad (x_1, x_2) \quad \mapsto \quad (x_2 \oplus h(x_1) \ , \ x_1) \tag{112}$$

A 1ª componente do "input" é transferida para a 2ª componente do "output" sem qualquer transformação; a 2ª componente do "input" é misturado com $h(x_1)$ para produzir a 1ª componente do "output".

Das equações

$$y_1 \ = \ x_2 \oplus h(x_1) \qquad , \qquad y_2 \ = \ x_1$$

concluímos

$$x_1 \ = \ y_2 \qquad , \qquad x_2 \ = \ y_1 \oplus h(y_2)$$

Desta forma a inversa de f é a função

$$f^{-1} \quad : \quad (y_1, y_2) \quad \mapsto \quad (y_2 \ , \ y_1 \oplus h(y_2)) \tag{113}$$

que é uma função análoga a f. De facto, com a ajuda da função "troca" definda por $t\colon (x, y) \mapsto (y, x)$, vemos que

$$f^{-1} \quad = \quad t \circ f \circ t \tag{114}$$

4.3 Rijndael-AES

Em 2/10/00 o *National Institute of Standards and Technology* (NIST) do *US Department of Commerce* recomendoa adopção do algoritmo designado por **Rijndael**como **Advanced Encryption Standard** (AES).

O AES foi adoptado em 2002 como *standard* para criptografia simétrica e substituiu *standard* designado por DES (Data Encryption Standard) que, na sua versão inicial ou na versão modificada designada como TRIPLEDES, tem vindo a ser usado desde 1977 na segurança de todo o sector financeiro bem como em inúmeras outras situações.

□

Alguns pontos fundametais sobre a arquitectura do AES

1. O Rijndael é uma cifra iterada com blocos de tamanho variável ($N_b \times 32$ bits, com $N_b = 4, 6$ ou 8) e chaves de tamanho variável ($N_k \times 32$ bits, com $N_k = 4, 6$ ou 8). O algoritmo tem 10 iterações (*rounds*) quando $N_b = N_k = 4$, tem 14 iterações quando $N_b = 8$ ou $N_k = 8$ e tem 12 iterações nos restantes casos.

2. O Rijndael **não é** um circuito de Feistel. É uma cifra orientada ao byte em que cada iteração (*round*) é uma substituição composta por três transformações invertíveis designadas por **camadas** (*layers*):
   - **Camada não linear (*non-linear layer*)**
     aplicação paralela de S-*boxes* destinada a evitar correlações cruzadas,

- **Camada linear (*linear mixing layer*)**
  garante a difusão dos bits transformados,
- **Mistura de chaves (*key addition layer*)**
  mistura das sub-chaves por simples operações **XOR**

3. Antes da primeira iteração é aplicada uma mistura de chaves prévia. A motivação para este tipo de operação (conhecido por *whitening*) reside no facto que qualquer transformação invertível que ocorra antes da primeira mistura de chaves pode ser retirada sem afectar as propriedades de segurança da cifra; é, portanto, redundante.

   De modo a melhorar a fase de decifragem a camada linear da última iteração é diferente da dos restantes.

Para facilitar a apresentação a descrição do algoritmo será aqui feita para o caso mais simples: blocos e chaves de 128 bits ($N_b = N_k = 4$) e 10 iterações.

Round do Rijndael

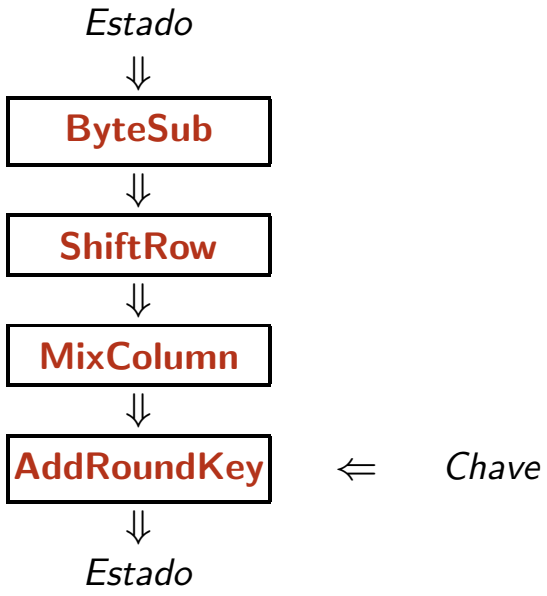
A definição de *round* é uma transformação sobre um **estado** de 128 bits organizado como uma matriz 4×4 de bytes. Cada *round* usa uma única sub-chave de 128 bits também organizada como uma matriz 4×4 de bytes. Ambas matrizes são sequencialmente organizadas "em colunas": i.e. a posição (i, j) da matriz corresponde à posição $i + 4 \times j$ num vector uni-dimensional.

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

Estado

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

Chave



Cada uma dos primeiros 9 *rounds* do Rijndael é uma sequência de 4 transformações sobre o estado acima representadas. O último *round* é análogo mas não tem a transformação **MixColumn**. Adicionalmente, antes do 1º *round* há uma transformação **AddRoundKey**.

Globalmente a sequência de transformações é

$$\begin{array}{l}\texttt{AddRoundKey} \Rightarrow \\ (\ \texttt{ByteSub} \Rightarrow \texttt{ShiftRow} \Rightarrow \texttt{MixColumn} \Rightarrow \texttt{AddRoundKey} \Rightarrow\ ) \quad \times 9 \\ \texttt{ByteSub} \Rightarrow \texttt{ShiftRow} \Rightarrow \texttt{AddRoundKey}\end{array}$$

As duas transformações ByteSub e ShiftRow formam a *Non Linear Layer*. A transformação MixColumn forma a *Linear Mixing Layer* enquanto que AddRoundKey implementa a *Key Addition Layer*.

Representação dos *bytes*

Cada byte (organizado do bit mais significativo para o menos significativo) $b_7\, b_6\, \cdots\, b_2\, b_1\, b_0$ é interpretado como um polinómio em Y do 7º grau

$$b_7 \times Y^7 + b_6 \times Y^6 + \cdots + b_2 \times Y^2 + b_1 \times Y + b_0$$

São definidas operações de soma e multiplicação de bytes do seguinte modo:

Soma $A \oplus B$ calcula-se fazendo o **XOR** bit-a-bit dos respectivos coeficientes.

Multiplicação $A \otimes B$ calcula-se multiplicando os polinómios e reduzindo o resultado módulo $Y^8+Y^4+Y^3+Y+1$.

EXEMPLO

$$\begin{aligned}\texttt{ff} \otimes \texttt{03} &= (Y^7+Y^6+Y^5+Y^4+Y^3+Y^2+Y+1) \otimes (Y+1) = \\ &= (Y^8+1) \pmod{Y^8+Y^4+Y^3+Y+1} = Y^4+Y^3+Y = \\ &= \texttt{1a}\end{aligned}$$

As operações de multiplicação e similares (como o cálculo da inversa) podem ser realizadas usando uma *tabela de logaritmos*: cada byte $b \neq 0$, pode ser escrito na forma

$$b = g^e \pmod{Y^8+Y^4+Y^3+Y+1} \qquad \text{com} \quad g = \texttt{ff}$$

para um único expoente $e \in \{0 \cdots 254\}$ determinado pela tabela 4 apresentada na página 292. Para $e > 254$, tem-se $g^e \equiv g^{e \bmod 255}$.

EXEMPLO

Vamos usar a tabela para calcular o produto $\texttt{ff} \otimes \texttt{03}$ e o inverso $\texttt{aa}^{-1}$.

Vemos que $\texttt{ff} = g^1$ e $\texttt{03} = g^{73}$. Logo $\texttt{ff} \otimes \texttt{03} = g^1 \otimes g^{73} = g^{74} = \texttt{1a}$.

Vemos que $\texttt{aa} = g^{223}$; logo $\texttt{aa}^{-1} = g^{-223 \pmod{255}}$. Como $-223 \equiv (255-223) \equiv 32 \pmod{255}$ o inverso será $\texttt{aa}^{-1} = g^{32} = \texttt{12}$.

Uma tabela de logaritmos e a sua inversa podem ser facilmente armazenadas numa S-Box (que se resume a um vector de 256 bytes). A implementação de operações de multiplicação ou calculo de inversas resumem-se a algumas **XOR** de bytes e procuras na tabela; deste modo são computacionalmente muito eficientes.

Transformação `ByteSub`

Actua em paralelo e de modo uniforme sobre cada um dos bytes da matriz do estado transformando-o do modo seguinte

$$b(Y) \quad \Longrightarrow \quad u(Y) \oplus v(Y) \times b^{-1}(Y) \pmod{Y^8+1}$$

$$\text{com} \quad u \equiv \texttt{c6} \quad \text{e} \quad v \equiv \texttt{f1}$$

Notas

1. A característica não linear da cifra é basicamente implementada na transformação $b \Longrightarrow b^{-1}$.
2. A multiplicação desta inversa por v usa um polinómio (Y^8+1) diferente do usado nas operações algébricas gerais sobre os bytes (aumentando a não-linearidade intrínseca). v é escolhido de modo que $v^{-1} \pmod{Y^8+1}$ exista.
3. A transformação pode ser implementada eficientemente numa S-Box 8×8 invertível. Na tabela 4 (página 291) é apresentada essa S-Box. A transformação inversa implementa-se usando esta mesma S-Box mas em sentido oposto.

4. A constante u é escolhida de modo que a S-Box não tenha pontos fixos (pontos b onde $S(b) = b$) nem anti-pontos fixos (pontos b onde $S(b) = \bar{b}$).

Transformação `ShiftRow`

As linhas da matriz que representa o estado sofrem uma rotação circular: a primeira não é rodada, a segunda roda uma posição, a terceira roda duas posições e a última roda 3 posições

$$\begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,3} & a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,0} \\ \hline \end{array}$$

A transformação inversa efectua-se rodando as linhas o mesmo número de posições mas em sentido inverso.

Transformação `MixColumn`

Nesta transformação do estado cada coluna da matriz que representa o estado é interpretada como um polinómio em X do 3º grau

$$\mathbf{a}_j(X) \equiv a_{0,j} + a_{1,j} \times X + a_{2,j} \times X^2 + a_{3,j} \times X^3$$

Nestes polinómios a soma é efectuada componente a componente e a multiplicação é efectuada módulo (X^4+1). As operações básicas são a multiplicação de cada $\mathbf{a}(X)$ por uma matriz $\mathbf{C}$ ou pela sua inversa $\mathbf{C}^{-1}$ dadas por

$$\mathbf{C}\ \mathbf{a}_j \equiv \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \otimes \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \qquad \mathbf{C}^{-1}\ \mathbf{a}_j \equiv \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \otimes \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

A operação **MixColumn** multiplica por **C** cada uma das 4 colunas do estado

$$\begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a_2} & \mathbf{a}_3 \end{bmatrix} \implies \begin{bmatrix} \mathbf{C}\,\mathbf{a}_0 & \mathbf{C}\,\mathbf{a}_1 & \mathbf{C}\,\mathbf{a}_2 & \mathbf{C}\,\mathbf{a}_3 \end{bmatrix}$$

A operação inversa de **MixColumn** é análoga mas usa a matriz $\mathbf{C}^{-1}$.

Transformação `AddRoundKey`

É efectuado um **XOR** byte-a-byte do estado e da sub-chave

$$\begin{array}{|cccc|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \implies \begin{array}{|cccc|} \hline a_{0,0} \oplus k_{0,0} & a_{0,1} \oplus k_{0,1} & a_{0,2} \oplus k_{0,2} & a_{0,3} \oplus k_{0,3} \\ a_{1,0} \oplus k_{1,0} & a_{1,1} \oplus k_{1,1} & a_{1,2} \oplus k_{1,2} & a_{1,3} \oplus k_{1,3} \\ a_{2,0} \oplus k_{2,0} & a_{2,1} \oplus k_{2,1} & a_{2,2} \oplus k_{2,2} & a_{2,3} \oplus k_{2,3} \\ a_{3,0} \oplus k_{3,0} & a_{3,1} \oplus k_{3,1} & a_{3,2} \oplus k_{3,2} & a_{3,3} \oplus k_{3,3} \\ \hline \end{array}$$

A transformação AddRoundKey coincide com a sua inversa.

A cifra inversa

Como cada transformação de um *round* é invertível, a sua inversa constrói-se por aplicação das inversas das diversas componentes pela ordem inversa. Notando que $\texttt{AddRoundKey}^{-1} = \texttt{AddRoundKey}$ a transformação será

$$\texttt{AddRoundKey} \Rightarrow \texttt{MixColumn}^{-1} \Rightarrow \texttt{ShiftRow}^{-1} \Rightarrow \texttt{ByteSub}^{-1}$$

O primeiro *round* não tem a transformação $\texttt{MixColumn}^{-1}$ e o último *round* é seguido de uma aplicação extra de AddRoundKey. As sub-chaves são usadas pela ordem inversa da usada na cifragem.

Note-se que

1. A ordem relativa das transformações ByteSub e ShiftRow é indiferente porque ByteSub do mesmo modo para qualquer byte e ShiftRow apenas muda a ordem relativa dos bytes.
2. A ordem relativa de MixColumn e AddRoundKey é indiferente porque MixColumn é uma transformação linear, AddRoundKey é uma soma e as transformações lineares $f(\cdot)$ verificam $f(a+k) = f(a) + f(k)$.

Se notar-mos que o 1º *round* não tem a transformação $\texttt{MixColumn}^{-1}$ vemos então (usando estas duas transposições)

que a decifragem é dada pela sequência exactamente análoga à da cifragem

$$\begin{array}{l} \texttt{AddRoundKey} \Rightarrow \\ (\ \texttt{ByteSub}^{-1} \Rightarrow \texttt{ShiftRow}^{-1} \Rightarrow \texttt{MixColumn}^{-1} \Rightarrow \texttt{AddRoundKey} \Rightarrow\ ) \quad \times 9 \\ \texttt{ByteSub}^{-1} \Rightarrow \texttt{ShiftRow}^{-1} \Rightarrow \texttt{AddRoundKey} \end{array}$$

4.4 Programador de Chaves

O programador de chaves gera, a partir da chave principal, tantas sub-chaves (com o mesmo tamanho da chave principal) quantos os *rounds* mais 1.

Note-se que cada *round* contém uma transformação **AddRoundKey** e, adicionalmente, antes do primeiro round existe uma operação **AddRoundKey** extra.

Vamos ilustrar a geração das sub-chaves para o caso em que $N_b = N_k = 4$ e são usados 10 *rounds*. Os princípios gerais são os seguintes:

- A `RoundKey` é um vector unidimensional de palavras de 32 bits (4 bytes) e de tamanho 4×11 . Sequencialmente são usadas 4 palavras deste vector para formar cada uma das sub-chaves.
- A chave principal (designada `CipherKey`) é expandida para a `RoundKey` segundo o algoritmo seguinte.
  1. O vector `RoundKey` é inicializado com 11 cópias do vector `CipherKey`.
  2. Para $i = 4$ até 44 a palavra na posição i é substituída pelo **XOR** da palavra na posição $i - 4$ e pela palavra w calculada da seguinte forma.
     (a) Se i não é múltiplo de 4 usa-se para w a palavra na posição $i - 1$,
     (b) Se i é múltiplo de 4 e se for $k = i/4$ e s a palavra na posição $i - 1$, então

$$w = \texttt{ByteSub4}(\texttt{rot4}(s)) \oplus \texttt{cons}(k)$$

em que:

i. $\texttt{cons}(k)$ é uma palavra constante

$$(Y^k, 00, 00, 00)$$

em que o 1º byte é representado pelo polinómio Y^k e os restantes três bytes são 00.

ii. $\texttt{rot4}(s)$ roda os 4 bytes de s uma posição:

$$(b_0, b_1, b_2, b_3) \Rightarrow (b_1, b_2, b_3, b_0)$$

iii. $\texttt{ByteSub4}(\cdot)$ aplica a S-Box ByteSub a cada um dos 4 bytes da palavra.

Tabelas AES

S-Box da cifra Rijndael

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

Tabela de logaritmos de gerador $g = \texttt{ff}$ em GF(2^8) com polinómio carcaterístico $Y^8 + Y^4 + Y^3 + Y + 1$.

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

Na talela 4 a A entrada $(\texttt{d}_1, \texttt{d}_2)$ contém o expoente e tal que $b = g^e$ se escreve, em hexadecimal $\texttt{d}_1\texttt{d}_2$.

4.5 Outras Cifras por Blocos

Nos últimos anos, para além da cigra Rijndael adoptada como cifra AES, outras cifras mereceram atenção. Nomeadamente

Twofish cifra patrocinada por Bruce Schneier ao concurso AES e, durante bastante tempo, considerada como favorita. É uma cifra de construção que se pode classificar como "tradicional". Usa o conhecimento experimental para propor uma arquitectura de Feistel dupla.

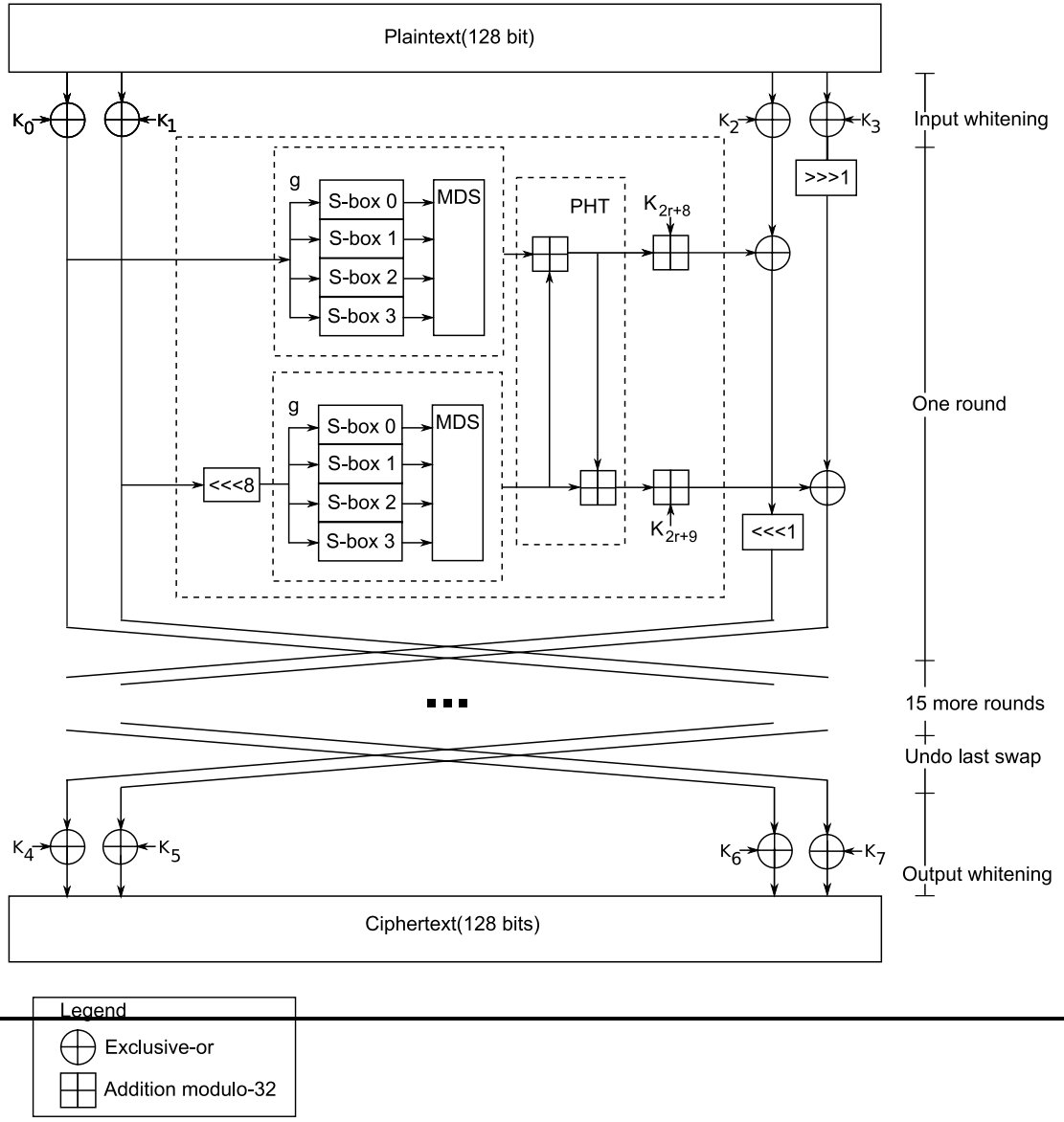
Kasumi Retirado do "ETSI 3rd Generation Partnership Project (3GPP) Tecnical Specification"

> *"The 3GPP Confidentiality and Integrity Algorithms f8 & f9 have been developed through the collaborative efforts of the European Telecommunications Standards Institute (ETSI), the Association of Radio Industries and Businesses (ARIB), the Telecommunications Technology Association (TTA), the T1 Committee. The f8 & f9 Algorithms Specifications may be used only for the development and operation of 3G Mobile Communications and services."*

Twofish é uma cifra que, na versão base, opera com blocos de 128 bits e chaves de 128 bits. Cada "round" é um circuito de Feistel duplo, cuja arquitectura geral está respresentada na figura seguinte.

MDS é uma transformação linear (matriz difusora) destinada a distribuir os "outputs" das S-Boxes.

PHT (*pseudo Hadamard Transformation*) é uma transformação de mistura dos dois circuitos de Feistel.



Kasumi

Na terminologia KASUMI, `f8` designa o algoritmo que implementa 8 "rounds" de cifra enquanto que `f9` é uma função de *hash* derivada de `f8`.

KASUMI é uma cifra de Feitsel com 8 "rounds" que usa blocos de 64 bits e chaves de 128 bits. A representação esquemática de um "round" Kasumi é feita nos seguintes diagramas.

4.6 Condições de Segurança em Cifras Simétricas

Segurança Orientada à Incerteza

A abordagem clássica à noção de segurança assenta nos conceitos de preservação de incerteza; essencialmente traduz a ideia de que a observação de um determinada valor do criptograma não dá qualquer informação sobre o eventual texto que lhe tenha dado origem.

Seja ψ uma função parcial recursiva e X, Y conjuntos finitos. Defina-se a relação $X \overset{\psi}{\hookrightarrow} Y$ por

$$X \overset{\psi}{\hookrightarrow} Y \quad = \quad \{\, x\|y \mid x \in X \wedge y \in Y \wedge (\exists k)\,[\,\psi(k\|x) \simeq y\,]\,\} \tag{115}$$

Tome-se um qualquer medida de incerteza ϑ definível em conjuntos finitos. Neste contexto

170 Noção
*A função ψ é (ϑ, X, Y) -**perfeitamente segura** se e só se $\vartheta(X \overset{\psi}{\hookrightarrow} Y) \geq \vartheta(Y) + \vartheta(X)$. A função é ϑ-**perfeitamente segura** se e só se for (ϑ, X, Y) -segura para todos conjuntos X, Y finitos.*

Exemplo 34:
Considere-se cifras por blocos $f : \mathbb{B}^t \times \mathbb{B}^n \to \mathbb{B}^n$. Temos $X = Y = \mathbb{B}^n$ e, usando ϑ como a incerteza de Hartley ($\vartheta(A) = \log_2[\![A]\!]$), será $\vartheta(X) = \vartheta(Y) = n$.

Vamos analisar dois exemplos de tais cifras vendo em que condições podem ou não ser ϑ-perfeitamente seguras.

Para simplificar a notação vamos designar por C_f a relação $\mathbb{B}^n \overset{f}{\hookrightarrow} \mathbb{B}^n$; f será ϑ, X, Y-perfeitamente segura se $\vartheta(C_f) = \vartheta(X) + \vartheta(Y) = 2\,n$.

1. **Cifra de Vernam** ("one-time pad") $f(k, x) = k \oplus x$.

$$C_f \quad \sim \quad \{\, (x, y) \in \mathbb{B}^n \times \mathbb{B}^n \mid \exists k \cdot y = x \oplus k \,\}$$

   Como, para todo $x, y \in \mathbb{B}^n$, existe sempre um k tal que $y = k \oplus x$ (nomeadamente $k = x \oplus y$), tem-se $C_f \sim \mathbb{B}^{2\,n}$. Portanto $\vartheta(C_f) = 2\,n$ e a cifra é perfeitamente segura.

2. **Cifra "perfeitamente insegura"** $f(k, x) \simeq g(x)$ para alguma permutação g.

   Tem-se $(\exists k)\,[f(k, x) \simeq y] \equiv [g(x) \simeq y]$; i.e. a "cifra" f não usa qualquer chave. Logo

$$C_f \sim \{\, (x, y) \in \mathbb{B}^n \times \mathbb{B}^n \mid \exists k \cdot f(k, x) \simeq y \,\} = \{\, (x, g(x)) \mid x \in \mathbb{B}^n \,\} \sim \mathbb{B}^n$$

   Portanto $\vartheta(C_f) = n < 2\,n$; como seria de esperar, f não é perfeitamente segura.

Nota

Estes dois casos descrevem situações extremas: segurança perfeita e insegurança perfeita. Note-se que, porque f é uma cifra, é uma função

total e, para cada x, existe sempre um valor y (pelo menos) no qual o predicado $(\exists k)\,[f(k\|x) \simeq y]$ é válido. Por isso, tem-se sempre $\vartheta(X \overset{f}{\hookrightarrow} Y) \geq \vartheta(X)$; por outro lado, como $(X \overset{f}{\hookrightarrow} Y) \subseteq X \times Y$, a incerteza de Hartley garante que $\vartheta(X \overset{f}{\hookrightarrow} Y) \leq \vartheta(X) + \vartheta(Y)$. Desta forma, em cifras por blocos, $\vartheta(X \overset{f}{\hookrightarrow} Y) - \vartheta(X)$ é sempre um valor compreendido entre 0 e $\vartheta(Y)$. Na primeira hipótese temos insegurança perfeita e na segunda temos segurança perfeita.

O exemplo anterior dá algumas pistas sobre o papel da dimensão do espaço e chaves. Começamos por ver o seguinte resultado

171 LEMA *Para todos conjuntos X, Y e toda a função parcial recursiva ψ tem-se*

$$X \overset{\psi}{\hookrightarrow} Y \;\sim \biguplus_{x \in X} \mathrm{rng}(\psi_x) \cap Y \tag{116}$$

em que $\uplus$ denota a união disjunta de conjuntos e $\{\psi_x\}$ é a sequência de funções parciais recursivas definidas por $\psi_x(k) = \psi(k\|x)$.

Esboço de prova
O lado direito em (115) pode-se reescrever $\{\, x\|y \mid x \in X \wedge y \in \mathrm{rng}(\psi_x) \,\} \cap Y$.

172 FACTO
Se ϑ é uma medida de incerteza monotónica[42] *e ψ é ϑ, X, Y-perfeitamente seguro, então* $\mathrm{rng}(\psi_x) \supseteq Y$ *, para*

[42] Isto é, $A \subseteq B$ implica $\vartheta(A) \leq \vartheta(B)$.

todo $x \in X$. Adicionalmente, se ϑ for a incerteza de Hartley, verifica-se $\vartheta(\mathrm{dom}(\psi_x)) \geq \vartheta(Y)$.

Prova A primeira asserção é consequência do lema 171 e da hipótese $\vartheta(X \overset{\psi}{\hookrightarrow} Y) \geq \vartheta(X) + \vartheta(Y)$. A segunda asserção resulta do facto de $[\![\mathrm{dom}(\psi_x)]\!] \geq [\![\mathrm{rng}(\psi_x)]\!]$, para todo ψ_x.

Como consequência

173 TEOREMA
Seja ϑ a incerteza de Hartley. Uma cifra por blocos $f : \mathbb{B}^t \times \mathbb{B}^n \to \mathbb{B}^n$ só é ϑ-perfeitamente segura se for $t \geq n$.

Prova
A cifra é modelada pela função parcial recursiva ψ que verifica $\psi(k\|x)\downarrow \Leftrightarrow |k| = t \wedge |x| = n$ e, ainda, $\psi(k\|x)\downarrow \Rightarrow \psi(k\|x) \simeq f(k, x)$. Dado que f é total, tem-se, para todo x, $\mathrm{dom}(\psi_x) \sim \mathbb{B}^t$ e $\vartheta(\mathrm{dom}(\psi_x)) = t$. O facto 172 diz-nos que f só pode ser perfeitamente segura quando for $t \geq n$.

□

Quando se determina a incerteza $\vartheta(A)$ de um conjunto A, estamos a colocar um limite superior ao esforço computacional para encontrar um elemento $a \in A$ que verifique uma **propriedade discriminadora** arbitrária ρ; por exemplo, uma propriedade que verifique $\rho(a) \simeq 1 \Leftrightarrow [a = x]$, para alguma x desconhecido.

A incerteza $\vartheta(A)$ deve traduzir a intuição de que

Os algorimos que decidem $(\exists a \in A) \cdot \rho(a)$ *têm, independentemente de* ρ*, um limite de complexidade* $O(2^{\vartheta(A)})$.

Por isso faz sentido associar a incerteza do conjunto vazio, $\vartheta(\emptyset)$, ao valor $-\infty$: a complexidade de procurar num conjunto vazio deve ser sempre $0 = 2^{-\infty}$.

Quando se escreve $\vartheta(X \overset{\psi}{\hookrightarrow} Y) - \vartheta(Y)$ estamos a caracterizar a complexidade de, dado um específico $y \in Y$, discriminar o $x \in X$ particular cujo criptograma é y.

EXEMPLO 35:
Sopunhamos que do conjunto dos textos X seleccionamos um subconjunto de X com apenas dois elementos; $\{x, x'\}$. Do conjunto de criptogramas Y, seleccionamos um só elemento y. Usando a incerteza de Hartley , tem-se $\vartheta(\{x, x'\}) = 1$ e $\vartheta(\{y\}) = 0$.

Ao determinar $\vartheta(\{x, x'\} \overset{\psi}{\hookrightarrow} \{y\})$, assume-se que são conhecidos os três valores x, x', y e procura-se seleccionar um de dois textos (x ou x') que tenha y como criptograma. Informalmente, $\vartheta(\{x, x'\} \overset{\psi}{\hookrightarrow} \{y\})$ mede a incerteza de seleccionar "aleatoriamente" um dos dois textos e de ser esse, de facto, o texto que originou o criptograma y.

Este exemplo motiva a análise de como se relaciona segurança perfeita com a capacidade de seleccionar, de entre dois textos, aquele que origina um criptograma arbitrário.

174 Teorema

A função ψ é ϑ, X, Y-perfeitamente segura se e só se, para todo $\{x, x'\}$, verifica-se

$$\vartheta(\{x, x'\} \overset{\psi}{\hookrightarrow} Y) - \vartheta(Y) \;\geq\; 1$$

Esboço de prova

Se for $\{x, x'\}$ um conjunto formado por dois textos distintos em X, tem-se

$$\{x, x'\} \overset{\psi}{\hookrightarrow} Y \quad \sim \quad \mathrm{rng}(\psi_x) \cap Y \;\uplus\; \mathrm{rng}(\psi_{x'}) \cap Y$$

Se suposermos que ψ é X, Y-perfeitamente segura, então o facto 172 garante-nos tanto $\mathrm{rng}(\psi_x)$ como $\mathrm{rng}(\psi_x)$ contêm Y; portanto $\{x, x'\} \overset{\psi}{\hookrightarrow} Y \;\sim\; Y \uplus Y$ e, por isso, $\vartheta(\{x, x'\} \overset{\psi}{\hookrightarrow} Y) - \vartheta(Y) \geq 1$.

Inversamente vamos supor que ψ não era ϑ, X, Y-perfeitamente seguro. Pelo lema 171 isto significa que existe pelo menos um $x \in X$ para o qual $\mathrm{rng}(\psi_x)$ é um subconjunto próprio de Y; ou seja existe um y tal que $\forall k \cdot \psi(k\|x) \neq y$. Escolha-se um outro x' tal que $\mathrm{rng}(\psi_{x'}) \supseteq Y$. Então, para este par de textos $\{x, x'\}$ tem-se $[\![\{x, x'\} \overset{\psi}{\hookrightarrow} Y]\!] < 2\,[\![Y]\!]$ e, consequentemente, $\vartheta(\{x, x'\} \overset{\psi}{\hookrightarrow} Y) - \vartheta(Y) < 1$.

Segurança Orientada à Aleatoriedade

A "perfeita segurança" pode não ser a forma perfeita de definir segurança.

Suponhamos, por exemplo, que ψ era tal que, para um valor x particular, se verificava $\psi(k\|x) \simeq k$. Pode acontecer esta condição se verifique mesmo que ψ seja "perfeitamente segura"; veja-se o *"one-time pad"* onde o texto $x = 0$ faz com que o criptograma $y = k \oplus x$ reproduza a chave. A intuição diz-nos que esta cifra não é segura se a mesma chave k for usada mais do que uma vez[43].

Por isso, uma noção de segurança que esteja de acordo com esta intuição, tem de considerar a possibilidade de um atacante escolher uma sequência de textos $\{x_n\}$ e observar a sequência de criptogramas correspondente $\{y_n\}$.

[43] Daí o qualificativo "one-time".