Universidade de Aveiro Departamento de Matemática

2012

ILÍDIO MENDES MOREIRA

CRIPTOGRAFIA DE CHAVE PÚBLICA COM BASE EM CÓDIGOS

ILÍDIO MENDES MOREIRA

CRIPTOGRAFIA DE CHAVE PÚBLICA COM BASE EM CÓDIGOS

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações, realizada sob a orientação científica da Doutora *Maria Raquel Rocha Pinto*, professora Auxiliar do Departamento de Matemática da Universidade de Aveiro **e** do Doutor *Paulo José Fernandes Almeida,* professor Auxiliar do Departamento de Matemática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Helmuth Robert Malonek

professor Catedrático da Universidade de Aveiro

Prof. Doutor José Pedro Miranda Mourão Patrício

professor Auxiliar do Departamento de Matemática da Universidade do Minho

Profª. Doutora Maria Raquel Rocha Pinto (Orientadora)

professora Auxiliar do Departamento de Matemática da Universidade de Aveiro

Prof. Doutor Paulo José Fernandes Almeida (Orientador)

professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

Agradecimentos / acknowledgements

Em primeiro lugar, queria expressar um profundo agradecimento à minha família, pelo elevado espírito de sacrifício em me acompanhar nesta tarefa: minha esposa Dulcelina e meus filhos Jocieline, Luizeth e Dário.

Aos meus orientadores, a professora Doutora Maria Raquel Rocha Pinto e o professor Doutor Paulo José Fernandes Almeida, pela entrega profissional, pelas palavras sábias e sempre motivadoras e, por me terem dado a oportunidade de, com as suas experiências, crescer cientificamente.

A todos os professores, colegas, familiares e amigos, que directa ou indirectamente contribuíram para a consecução deste objectivo.

A Deus, pela vida, saúde e força para continuar.

palavra-chave

sistema criptográfico de McEliece, chave pública, código de Goppa, corpo finito, assinatura digital.

resumo

Este trabalho de dissertação foca o sistema criptográfico de McEliece. Este é um sistema criptográfico de chave pública que tira partido do facto do problema de descodificação de um código linear geral ser NP-completo. Mais especificamente, este sistema criptográfico usa um código de Goppa sobre um corpo finito como chave privada, para o qual existe um algoritmo de descodificação eficiente, e um código linear geral, derivado do código Goppa anterior, como chave pública.

Assim, neste trabalho, começa-se por analisar alguns resultados sobre corpos finitos, necessários ao longo desta dissertação. Posteriormente, estudam-se os códigos lineares sobre corpos finitos, em particular os códigos de Goppa, apresentando-se um algoritmo de descodificação para estes códigos. Em seguida, é apresentada uma descrição detalhada do sistema criptográfico de McEliece e são analisados alguns ataques a este sistema criptográfico. Por fim, é ainda analisada a sua aplicação na segurança de assinaturas digitais.

keywords

McEliece cryptosystem, public key, Goppa code, finite field, digital signature.

abstract

This thesis studies the public key McEliece cryptosystem, which is based on the fact that the problem of decoding a general linear code is NP-hard. More specifically, the McEliece cryptosystem uses a Goppa code over a finite field as private key (these codes have an efficient decoding algorithm), and a general linear code as public key.

We start by analysing some results about finite fields, which will be use later. We also study the theory of linear codes over finite fields and we analyse with detail the Goppa codes, presenting in particular a decoding algorithm for such codes. Next, we concentrate on the study of the McEliece cryptosystem and we describe some attacks against this cryptosystem. Finally, the application of this cryptosystem to digital signatures security is analysed.

Conteúdo

Lista de Tabelas

Capítulo 1

Introdução

Há cerca de trinta anos atrás, o surgimento da criptografia de chave pública impulsionou, por um lado, uma incrível revolução a nível de segurança nas comunicações electrónicas e, por outro, um aumento considerável de estudos e investigações à volta dos sistemas criptográficos. Tudo "começou" por volta do ano de 1976, quando Diffie e Hellman (ver [6]) apresentaram publicamente o primeiro sistema de troca de chaves cuja segurança é baseada na dificuldade de resolver os problemas de logaritmo discreto (DLP). Em 1977 o sistema de chave pública baseado no algoritmo **RSA**, fundamentado no problema de factorização de inteiros (FP), foi apresentado por Ron Rivest, Adi Shamir e Leonard Adleman (ver [29]). Em 1984 Taher **Elgamal** (ver [8]) descrevia o seu sistema criptográfico ancorado no sistema de troca de chaves de Diffie-Hellman. **Criptografia de curvas elípticas** (**ECC**) é proposta por volta de 1985, em trabalhos independentes, por Neal Koblitz (ver [16]) e Victor S. Miller (ver [24]).

McEliece [21], também em 1977, apresentou o sistema criptográfico de chave pública que leva o seu nome, baseado num problema da teoria de códigos, especificamente, no problema de descodificação de um código linear genérico, provado como pertencente à classe $\mathcal{NP}$- completo (ver [2]). O sistema por ele proposto usa como "trapdoor", o código de Goppa que possui algoritmos eficientes[1] tanto para codificar como para descodificar. Mas isto não foi suficiente para evitar ser pouco usado, actualmente, quando comparado com os três sistemas anteriormente referidos. Aqueles, com ênfase no RSA, constituíram desde cedo o alicerce para, praticamente, todos os protocolos de segurança e todas as

[1] Stefan Heyse em [13] aponta dados de implementação como sendo mais eficiente do que o RSA

implementações de criptografia de chave pública.

Em 1994, Peter W. Shor [32] dá a conhecer um algoritmo quântico para factorização de inteiros e cálculo de logaritmo discreto. Este algoritmo, teoricamente, quebra o sistema criptográfico RSA e de Elgamal e, consequentemente, a criptografia de curvas elípticas por estas duas últimas estarem suportados em problemas relacionados. Contrariamente, o sistema proposto por McEliece, com parâmetros bem escolhidos, resiste aos ataques quânticos (ver [7]). Isto faz dele um forte candidato à criptografia pós-quântica. Este facto, aliado à curiosidade científica sobre um assunto que ganha cada vez mais relevância, constitui um elemento motivador para o nosso estudo sobre "*criptografia de chave pública com base em códigos*" especificamente, o **sistema criptográfico de McEliece**.

Neste trabalho, propomos estudar o sistema criptográfico de McEliece e a variante de Niederreiter, os principais ataques e consequentes propostas de defesa. Como estes sistemas são baseado em códigos (lineares) correctores de erros, estudaremos a estrutura de um código linear binário e o processo de correcção de erros, em especial os códigos binários de Goppa que são utilizados no sistema criptográfico de McEliece.

Este trabalho está dividido em cinco capítulos. Nos preliminares, capítulo 2, abordamos conceitos que consideramos fundamentais, como estruturas algébricas e o algoritmo de Euclides, para melhor compreensão da estrutura de corpos finitos, $GF(p^m)$, bem como dos cálculos sobre os mesmos, apresentados no capítulo 3. No capítulo 4, debruçamo-nos sobre os códigos lineares - sua representação matricial, distância mínima de Hamming e formas de descodificação - especificando na secção 4.2, os códigos binários de Goppa e ilustrando um algoritmo de correcção de erros. No capítulo 5 descrevemos o sistema criptográfico de McEliece bem como uma das suas variantes, o sistema criptográfico de Niederreiter. Um exemplo de uso do sistema para cifrar e decifrar mensagens é descrito na secção 5.2.1. Neste exemplo, os outputs foram gerados a partir de uma implementação do sistema feita, por nós, no Maple e apresentada no apêndice A.1. A questão de segurança do sistema é abordada, no capítulo 6, com base na descrição de alguns ataques e na dificuldade de sucesso dos mesmos ou defesas correspondentes. Por fim, no capítulo 7, falamos sobre a construção de assinaturas digitais com base na variante de Niederreiter, proposta em [5].

Capítulo 2

Preliminares

Neste capítulo, pretendemos abordar, de forma breve, alguns conceitos que compreendemos serem essenciais para o desenvolvimento dos capítulos posteriores. De entre estes estão as estruturas algébricas - anéis e corpos - e propriedades relacionadas, anéis de polinómios, algoritmo de Euclides para polinómios como uma forma prática de determinação tanto do máximo divisor comum entre polinómios, como do inverso multiplicativo de um polinómio - ferramenta bastante útil para cálculos em corpos finitos - e classes de resíduos módulo um polinómio irredutível, com relevância na construção de corpos finitos sobre os quais definimos os códigos, neste trabalho.

2.1 Estruturas algébricas

Os conceitos apresentados nesta secção, têm [17] como suporte bibliográfico. Ali, também, podem ser lidas as demonstrações dos teoremas que aqui aparecem.

2.1.1 Anéis e Corpos

Definição 2.1 (Anel)**.** *Um anel* $(A, +, \cdot)$ *é formado por um conjunto munido de duas operações binárias que representamos por* "$+$" *e* "$\cdot$" *tais que:*

1. *A é* ***um grupo abeliano*** *em relação à operação "+".*
2. *A operação "·" é* ***associativa****.*
3. *"·" é* ***distributivo*** *em relação à "+".*

Podemos classificar os anéis de acordo com a seguinte definição:

Definição 2.2 (Classificação de anéis:)**.**

1. *Um anel A é dito* ***anel com identidade*** *se admite identidade multiplicativa, que é o mesmo que dizer:* $\exists\, e,\ \ ae = ea = a\ \ \forall a \in A$
2. *Um anel é* ***comutativo*** *se "·" é comutativo.*
3. *Um anel A é dito um* ***domínio de integridade*** *se é um anel comutativo com identidade,* $e \neq 0$*, no qual* $ab = 0 \ \Rightarrow\ a = 0$ *ou* $b = 0$*.*
4. *Um anel A é um* ***anel de divisão*** *se* $(A \setminus \{0\}, \cdot)$ *forma um grupo multiplicativo.*
5. *Um anel de divisão comutativo é um* ***corpo****.*

Definição 2.3 (Característica)**.** *Se A é um anel arbitrário e existe um inteiro positivo n tal que* $na = 0$ *para todo* $a \in A$*, então o mais pequeno n nesta condição chama-se* ***característica de*** *A que simbolizamos, aqui, por* $car(A)$ *e, diz-se que A tem característica n. Se tal inteiro positivo não existe, diz-se que A tem característica 0.*

Teorema 2.4. *Um anel A, de característica positiva tendo uma identidade e sem divisores de zero tem característica prima*

Teorema 2.5. *Seja A um anel comutativo de característica p - primo. Então,*

$$(a+b)^{p^m} = a^{p^m} + b^{p^m} \qquad e \qquad (a-b)^{p^m} = a^{p^m} - b^{p^m}, \qquad \text{para } a, b \in A \text{ e } m \in \mathbb{N}.$$

Definição 2.6 (Subanel)**.** *Um subconjunto S de um anel $A(+,\cdot)$ é um* ***subanel de*** *A desde que seja fechado em relação às operações "+" e "·" e, com elas, forme um anel.*

Definição 2.7 (Ideal)**.** *Um subconjunto I de um anel A é um* ***ideal*** *desde que seja um subanel de A e, para todos* $a \in I$ *e* $r \in A$*,* $ar \in I$ *e* $ra \in I$*.*

Definição 2.8 (Ideal principal)**.** *Seja A um anel comutativo. Um ideal I de A é dito* ***principal*** *se existe um* $a \in A$ *tal que* $I = (a)$*. Neste caso, I, também, designado de* ***ideal principal gerado por*** *a.*

Definição 2.9. *Um anel, A, é dito* ***domínio de ideais principais*** *se todo ideal de A é principal.*

Um ideal I do anel A define uma partição de A constituída de classes laterais disjuntas, a que chamamos **classes de resíduos módulo I**. A classe de resíduo de um elemento $a \in A$ módulo I consiste de todos os elementos de A que são da forma $a + c$ para algum $c \in I$ e será representada por $[a] = a + I$. Dois elementos a e b de A dizem-se **congruentes módulo** $I (a \equiv b \mod I)$ se pertencem à mesma classe de resíduo módulo I.

O conjunto das classes de resíduos de um anel A módulo um ideal I forma um anel sob as operações "$+$" e "$\cdot$".

Definição 2.10 (Anel quociente)**.** *O anel das classes de resíduos do anel A módulo o ideal I sob as operações "$+$" e "$\cdot$" é denominado* ***anel quociente*** *e representa-se por A/I.*

Exemplo 2.11. *O anel das classes de resíduos $\mathbb{Z}/(n)$ é constituído pelas classes laterais (classes de resíduos) de um inteiro, a, módulo um inteiro positivo n: $[0] = 0+(n)$, $[1] = 1+(n)$, $\ldots, [n-1] = n-1+(n)$. em particular, seja $n = 3$ então, $\mathbb{Z}/(3)$ é constituído pelos elementos $[0] = 0 + (3)$, $[1] = 1 + (3)$, $[2] = 2 + (3)$.*

Da definição 2.2, fica subentendido, nas alíneas 3), 4) e 5) o seguinte:

Definição 2.12 (Corpo)**.** *Um corpo é um domínio de integridade no qual, para todos os elementos $a \neq 0$, existe um elemento b tal que $a \cdot b = 1$. Este elemento, usualmente representado por $\frac{1}{a}$ ou a^{-1}, é o inverso multiplicativo de a.*

Um corpo diz-se finito se tem um número finito de elementos, e infinito caso contrário. Como consequência do teorema 2.4 vem que:

Teorema 2.13. *Um corpo finito tem característica prima.*

2.2 Anel de Polinómios

Definição 2.14. *Seja $f(x) = \sum_{i=0}^{n} a_i x^i$ um polinómio não nulo com $a_n \neq 0$ e coeficientes em um anel A. A n chamamos grau de f e simbolizamos por $gr(f) = n$. Por convenção*

$gr(0) = -\infty$ (grau de um polinómio nulo). Polinómios de grau ≤ 0 são denominados polinómios constantes.

Definição 2.15. *Seja A um anel comutativo. O conjunto $A[x] = \{\sum_{i=0}^{n} a_i x^i \mid a_i \in A\}$ é chamado **anel de polinómios sobre A** na variável x.*

Vamos assumir que, em $A[x]$, estejam definidas as operações de adição e multiplicação usual como seguem:
Dado dois polinómios $f(x) = \sum_{i=0}^{n} a_i x^i$ e $g(x) = \sum_{j=0}^{m} b_j x^j$,

$$\begin{aligned} f(x) + g(x) &= \sum_{i=0}^{n} (a_i + b_j) x^i, \quad \text{com} \quad i = j \ e \ n > m \\ f(x) \cdot g(x) &= \sum_{k=0}^{n+m} c_k x^k, \quad \text{onde} \quad c_k = \sum_{\substack{i+j=k \\ 0 \leq i \leq n,\, 0 \leq j \leq m}} a_i b_j \end{aligned}$$

Desta forma, temos o seguinte resultado:

Teorema 2.16. *Seja $f, g \in A[x]$ então,*

$$gr(f+g) \leq max\{gr(f),\ gr(g)\};$$

$$gr(f \cdot g) \leq gr(f) + gr(g).$$

Teorema 2.17. *Se A é um anel, então:*

1. *O anel de polinómio $A[x]$ é comutativo se e só se A o for.*
2. *$A[x]$ tem elemento unidade se e só se A o tiver.*
3. *$A[x]$ é um domínio de integridade se e só se A o for.*

Definição 2.18 (Domínio euclidiano)**.** *Seja A, um domínio de integridade. A é dito **domínio euclidiano** se existe uma função, λ, definida de $A \setminus \{0\}$ para $\mathbb{N}_0$ tal que, se $a, b \in A, b \neq 0$, existem $q, r \in A$ satisfazendo a seguinte propriedade:*

$$\begin{cases} a = bq + r \\ r = 0 \ ou \ \lambda(r) < \lambda(b) \end{cases}$$

O anel de polinómio, $A[x]$, é um domínio euclidiano. A função λ devolve o grau (gr) de cada elemento de $A[x]$.

É particularmente importante para nós assumirmos que os polinómios estejam definidos sobre uma estrutura de corpo. Representamos um corpo por $\mathbb{F}$ e o anel de polinómios com coeficiente em $\mathbb{F}$ por $\mathbb{F}[x]$.

Dizemos que o polinómio $g \in \mathbb{F}[x]$ divide o polinómio $f \in \mathbb{F}[x]$ se existe um polinómio $h \in \mathbb{F}[x]$ tal que $f = g \cdot h$.

O facto de $\mathbb{F}[x]$ permitir o algoritmo da divisão implica que todo Ideal $I \in \mathbb{F}[x]$ é principal. Logo $\mathbb{F}[x]$ é um **domínio de ideais principais** (ver [17, pág. 21]).

Daqui em diante, a menos de indicação em contrário, os polinómios estarão sempre definidos sobre um corpo finito, portanto, o anel de polinómio a considerar é sempre um domínio euclidiano, por conseguinte é um domínio de ideais principais.

2.2.1 Algoritmo de Euclides para polinómios

Algoritmo de Euclides é um dos mais antigos algoritmo que se conhece. O seu uso remonta ao século IV antes de Cristo e constitui, ainda hoje, numa importante ferramenta de computação. Este algoritmo está associado a diversos outros ligados à descodificação de códigos correctores de erros, nomeadamente, os códigos Reed-Solomon e os códigos de Goppa, razão pela qual propomos descrevê-lo aqui.

Teorema 2.19 (Algoritmo da divisão)**.** *Seja $f \neq 0$ um polinómio pertencente a $\mathbb{F}[x]$. Então, $\forall g \in \mathbb{F}[x]$, existem q e r pertencentes a $\mathbb{F}[x]$ tais que: $g = q \cdot f + r$, com $gr(r) < gr(f)$.*

Este algoritmo pode ser aplicado para encontrar o máximo divisor comum entre polinómios ou inteiros.

Máximo divisor comum aplicando o algoritmo de Euclides

Sejam f e g, dois polinómios em $\mathbb{F}[x]$. Representamos o máximo divisor comum entre f e g por $mdc(f,g)$.

A ideia do algoritmo de Euclides é reduzir o cálculo do máximo divisor comum entre par de polinómios, ao cálculo entre pares cada vez mais pequenos (de menor grau).

O algoritmo de Euclides gera uma sequência de restos estritamente decrescente em grau. $gr(r_1) < gr(r_2) < \ldots < gr(r_n)$, em que cada r_i é obtido a partir dos dois restos imediatamente anteriores: $r_i = r_{i-2} \mod r_{i-1}$ com $i \geq 0$.

Teorema 2.20 (Algoritmo de Euclides)**.** *Dados dois polinómios $f = r_0$ e $g = r_{-1}$ pertencentes a $\mathbb{F}[x]$ tais $gr(f) < gr(g)$. Fazendo divisões sucessivas obtém-se a seguinte sequência de equações:*

$$\begin{aligned} r_{-1} &= q_1 r_0 + r_1, & g(r_1) < g(r_0) \\ r_0 &= q_2 r_1 + r_2, & g(r_2) < g(r_1) \\ &\ldots \\ r_{i-2} &= q_i r_{i-1} + r_i, & g(r_i) < g(r_{i-1}) \\ r_{i-1} &= q_{i+1} r_i \end{aligned}$$

Assim, o último resto diferente de zero no processo de divisão inteira é o máximo divisor comum entre r_0 e r_{-1}.

Exemplo 2.21. *Determina o máximo divisor comum entre os polinómios $P_1 = x^4 + x^2 + x + 1$ e $P_2 = x^5 + x^3 + x + 1$ de $\mathbb{F}_2[x]$, onde $\mathbb{F}_2$ é o corpo constituído por $\{0,1\}$, de característica 2 (neste corpo, $1 + 1 = 0$).*

$$\begin{aligned} x^5 + x^3 + x + 1 &= x(x^4 + x^2 + x + 1) + x^2 + 1 \\ x^4 + x^2 + x + 1 &= (x^2 + 1)x^2 + x + 1 \\ x^2 + 1 &= (x + 1)(x + 1) + 0 \end{aligned}$$

Na última equação, o polinómio resto é igual a zero. Logo, o último resto diferente de zero,$(x + 1)$, é o polinómio máximo divisor comum entre P_1 e P_2.

Algoritmo estendido de Euclides

Teorema 2.22 (Algoritmo estendido de Euclides)**.** *Sejam r_{-1} e r_0, elementos de $\mathbb{F}[x]$ com $gr(r_0) \leq gr(r_{-1})$ e com máximo divisor comum h. Então, existem elementos U e V em $\mathbb{F}[x]$, tais que:*

$$U \times r_0 + V \times r_{-1} = h. \tag{2.1}$$

Demonstração. Repare que no teorema 2.20 estabelece-se que os sucessivos restos satisfazem o seguinte:

$$r_i = r_{i-2} - q_i r_{i-1}. \tag{2.2}$$

De (2.2) segue que todos os divisores comuns entre r_{i-1} e r_{i-2}, também dividem r_i e, todos os divisores comuns entre r_i e r_{i-1} também dividem r_{i-2}. Daí, para todo i, $mdc(r_i, r_{i-1}) = mdc(r_{i-1}, r_{i-2})$. Consequentemente:

$$r_i = mdc(r_i, r_{i-1}) = mdc(r_0, r_{-1}), \qquad r_{i+1} = 0. \tag{2.3}$$

Conforme a equação (2.2), cada resto é expresso como combinação linear dos dois restos imediatamente anteriores. Por indução mostra-se que o último resto é combinação linear dos dois primeiros restos, quais sejam: r_0 e r_{-1} ou seja $mdc(r_0, r_{-1}) = Ur_0 + Vr_{-1}$ para algum U e V. Com efeito:

$$\begin{aligned}
&\textit{se } i = 1, \qquad r_1 &&= r_{-1} - q_1 r_0. \\
&\textit{se } i = 2, \qquad r_2 &&= r_0 - q_2 r_1 \\
&&&= r_0 - q_2(r_{-1} - q_1 r_0) \\
&&&= (1 + q_1 q_2) r_0 - q_2 r_{-1}
\end{aligned}$$

Portanto, para $i = 1, 2$ os respectivos restos são expressos como combinação linear de r_0 e r_{-1}. Admitamos a hipótese de que quando $i \leq k$, $r_i = U_i r_0 + V_i r_{-1}$ e provemos que para $i = k + 1$, $r_{k+1} = U_{k+1} r_0 + V_{k+1} r_{-1}$:

$$\begin{aligned}
r_{k+1} &= -q_{k+1} r_k + r_{k-1} \qquad \textit{por } (2.2) \\
&= -q_{k+1}(U_k r_0 + V_k r_{-1}) + U_{k-1} r_0 + V_{k-1} r_{-1}, \qquad \text{por hipótese de indução} \\
&= \underbrace{(-q_{k+1} U_k + U_{k-1})}_{U_{k+1}} r_0 + \underbrace{(-q_{k+1} V_k + V_{k-1})}_{V_{k+1}} r_{-1}
\end{aligned}$$

□

Posto isso, fica estabelecido que:

$$r_i = U_i r_0 + V_i r_{-1}. \tag{2.4}$$

E mais, as sequências U_i *e* V_i são iterativamente calculadas:

$$U_i = -q_i U_{i-1} + U_{i-2} \qquad e \qquad V_i = -q_i V_{i-1} + V_{i-2} \; com \; i \geq 1.$$

De (2.4) deduz-se que $U_{-1} = 0, \quad V_{-1} = 1, \quad U_0 = 1 \quad e \quad V_0 = 0$, condições de partida de um processo prático para o cálculo do máximo divisor comum entre dois polinómios e do inverso multiplicativo de um polinómio.

Exemplo 2.23. *Determina* U, V *e* h *de modo a que* $h = mdc(P_1, P_2) = UP_1 + VP_2$ *com (*P_1 *e* P_2 *definidos no exemplo anterior).*

Em cada iteração, toma-se r_{i-2} *e* r_{i-1} *como o dividendo e o divisor respectivamente e, calcula-se o quociente e o novo resto. Os cálculos são mostrados na tabela 2.1 de onde se tem* $U = x^3 + 1$ *e* $V = x^2$.

i	$r_i = r_{i-2} \mod r_{i-1}$	q_i	U_i	V_i
-1	$x^5 + x^3 + x + 1$	-	0	1
0	$x^4 + x^2 + x + 1$	-	1	0
1	$x^2 + 1$	x	x	1
2	$x + 1$	x^2	$x^3 + 1$	x^2
3	0	$x + 1$	-	-

Tabela 2.1: Execução do algoritmo estendido de Euclides para os polinómios P_1 e P_2

De facto, $x + 1 = (x^4 + x^2 + x + 1)(x^3 + 1) + (x^5 + x^3 + x + 1)x^2$.

Corolário 2.24. *Se* r_{i-1} *e* r_i *são primos entre si então, existem elementos* U *e* V *tais que*

$$Ur_{-1} + Vr_0 = 1.$$

Repare que, para $r_i = 1$ tem-se $Ur_0 + Vr_{-1} = 1; \Rightarrow \; Ur_0 = 1 \mod r_{-1}$ ou seja $U = r_0^{-1} \mod r_{-1}$.

O exemplo que segue ilustra o cálculo do inverso multiplicativo de um polinómio, f, módulo outro, g.

Exemplo 2.25. *Dados os polinómios* $f(x) = x^5 + x^4 + x^3 + x + 1$ *e* $g(x) = x^4 + x^3 + 1$ *de coeficientes em* $\mathbb{F}_2$. *Vamos verificar que o* $mdc(f, g) = 1$ *e exibir* U *e* V *tal que* $gU + fV = 1$. *Segue a execução do algoritmo na tabela 2.2. Como se pode ver, o último*

i	$r_i = r_{i-2} \mod r_{i-1}$	q_i	U_i	V_i
-1	$x^5 + x^4 + x^3 + x + 1$	-	0	1
0	$x^4 + x^3 + 1$	-	1	0
1	$x^3 + 1$	x	x	1
2	x	$x + 1$	$x^2 + x + 1$	$x + 1$
3	1	x^2	$x^4 + x^3 + x^2 + x$	$x^3 + x^2 + 1$
4	0	x	-	-

Tabela 2.2: Execução do algoritmo estendido de Euclides

resto diferente de zero (linha 3 da tabela) é igual a 1, o que significa que $mdc(f, g) = 1$. *Os polinómios* $U_3 = x^4 + x^3 + x^2 + x$ *e* $V_3 = x^3 + x^2 + 1$ *são, respectivamente,* U *e* V *procurados. E mais, o polinómio* $V_3 = x^3 + x^2 + 1$ *é o inverso multiplicativo de* f *módulo* g *pois:*

$$(x^5 + x^4 + x^3 + x + 1) \times (x^3 + x^2 + 1) = 1 \mod (x^4 + x^3 + 1).$$

Como $\mathbb{F}[x]$ é um domínio de ideais principais, os elementos (polinómios) primos podem ser vistos como **polinómios irredutíveis** sobre $\mathbb{F}$ e vice-versa (ver [15, pág. 10]).

Definição 2.26. *Um polinómio* $f \in \mathbb{F}[x]$, *com* $gr(f) = n > 0$, *é irredutível sobre* $\mathbb{F}$ *se, neste corpo, não é representável como produto de factores de polinómios não constantes de grau inferior a* n.

Observe-se que a irredutibilidade de um polinómio depende fortemente do corpo em consideração. Assim, o polinómio $x^2 + 1 \in \mathbb{R}[x]$ é irredutível em $\mathbb{R}$, corpo de números reais, mas $x^2 + 1 = (x - i)(x + i)$ é redutível sobre o corpo de números complexos.

Os polinómios irredutíveis são muito importantes para a estrutura do anel, $\mathbb{F}[x]$, na medida em que todo $f \in \mathbb{F}[x]$ pode ser escrito de uma forma única, como produto de polinómios mónicos irredutíveis. A demonstração deste resultado, pode ser encontrada em [17, pág. 23 e 24].

No que refere aos anéis de polinómios, tem lugar o seguinte resultado:

Teorema 2.27. *Se $f \in \mathbb{F}[x]$ um polinómio irredutível sobre $\mathbb{F}$, então o anel das classes de resíduos, $\mathbb{F}[x]/(f)$, constituído por todos os possíveis resto da divisão dos polinómios de $\mathbb{F}[x]$ por f, é um corpo.*

Demonstração. Seja f um polinómio de grau n. $\mathbb{F}[x]/(f)$ é o conjunto de todos os polinómios de grau não superior a $n-1$ e com coeficientes em $\mathbb{F}$.

1. é fácil verificar que $\mathbb{F}[x]/(f)$ é um domínio de integridade. Com efeito,

   i) $\forall g \in \mathbb{F}[x]/(f)$, $g \cdot \mathbf{1} = g$, onde $\mathbf{1}$ é o polinómio unidade.

   ii) $\forall g \; e \; g' \in \mathbb{F}[x]/(f)$, $g \cdot g' = g' \cdot g$

   iii) $\forall g \; e \; g' \in \mathbb{F}[x]/(f)$, $g \cdot g' = 0 \;\Rightarrow\; g = 0 \vee g' = 0$. Com efeito, em $\mathbb{F}[x]/(f)$, $g \cdot g' = 0$ significa que $f|(g \cdot g')$ ou seja, $f|g$ ou $f|g'$ que é o mesmo que dizer, $g \equiv 0 \mod f$ ou $g' \equiv 0 \mod f$.

2. Como f é irredutível, a existência de *inversa multiplicativa* de qualquer elemento não nulo de $\mathbb{F}[x]/(f)$ é consequência do algoritmo de Euclides (estendido) 2.22.

Assim, tendo em conta a definição 2.12, por 1. e 2. $\mathbb{F}[x]/(f)$ é um corpo. □

Este resultado nos dá pistas em como construir corpos, como anéis de classes residuais módulo um polinómio irredutível, a ser abordado no capítulo que segue.

Capítulo 3

Corpos Finitos ou de Galois

Como já foi definido em secção anterior, um corpo finito pressupõe a existência de um conjunto finito de elementos no qual estão definidas as operações de adição, multiplicação e, é possível obter inversa de cada elemento. Os exemplos mais intuitivos são os corpos primos, $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ ou, também, denotado corpos de Galois com p elementos - $GF(p)$.

Exemplo 3.1. *O conjunto* $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ *relativamente às operações de adição e multiplicação modulo 5, definidas a seguir, é um corpo -* $GF(5)$*:*

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\cdot$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Tabela 3.1: tabela de adição e multiplicação mod 5

Um exemplo muito importante no âmbito deste trabalho é o corpo $GF(2)$. Vejamos, a seguir, a tabela de adição e multiplicação para este corpo:

Exemplo 3.2. $GF(2) = \{0, 1\}$

A aritmética é feita módulo 2. A adição é equivalente ao "ou" exclusivo e a multiplicação equivalente ao "$\wedge$*" lógico.*

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Muitas vezes é conveniente a construção de corpos maiores mas que conservem a simplicidade das operações definidas em $GF(2)$. No que segue, vamos analisar a construção de corpos como extensões do corpo $GF(2)$.

3.1 Extensão de Corpos Finitos - $GF(p^m)$

É fácil ver que, se $\mathbb{K}$ é um sub-corpo de $\mathbb{F}$, então $\mathbb{F}$ é um espaço vectorial sobre $\mathbb{K}$. Em particular, seja $\mathbb{F}_q$, com $q = p^m$, um espaço vectorial sobre $\mathbb{F}_p$ e, seja m a dimensão deste espaço vectorial, então tem-se que $\mathbb{F}_q = \{\sum_{i=1}^{m} a_i b_i \mid a_i \in \mathbb{F}_p\}$, onde $\{b_1, b_2, \ldots, b_m\}$ é uma base de $\mathbb{F}_q$ sobre $\mathbb{F}_p$. Portanto, $\mathbb{F}_q$ é uma combinação linear dos elementos desta base. Sendo assim, estabelece-se que:

Teorema 3.3. *[17] Todo o corpo finito $GF(q)$, tem p^m elementos, onde p, primo, é a característica do corpo e m é um inteiro maior ou igual a 1.*

$GF(2)$, $GF(3)$, $GF(5)$, $GF(2^2)$, $GF(2^3)$, $GF(2^4)$,$GF(3^2)$ etc. são alguns exemplos de corpos finitos.

Diz-se que o corpo $GF(p^m)$ é uma extensão do corpo $GF(p)$.

No que segue, vamos tentar ilustrar como construir uma extensão $GF(p^m)$ de $GF(p)$ como anel das classes de resto módulo um polinómio mónico e irredutível.

Como já foi provado, no teorema 2.27, $GF(p)[x]/(f(x))$ é um corpo. Se o grau de $f(x)$ é igual a m, então os elementos de $GF(p)[x]/(f(x))$ são polinómios de grau no máximo $m-1$.

$$GF(p)[x]/(f(x)) = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_0 \mid a_i \in GF(p)\}$$

Como os m coeficientes, a_i, pertencem a $GF(p)$, eles assumem p possíveis valores. Assim, o corpo $GF(p)[x]/(f(x))$ tem p^m elementos.

Prosseguimos, a partir daqui, com um exemplo em $GF(2)$ afim de percebermos, sem muitos formalismos matemáticos, a construção do corpo $GF(2^m)$ - uma extensão de $GF(2)$.

Exemplo 3.4. *Seja $f(x) = x^3 + x + 1$ um polinómio irredutível sobre $GF(2)$. O corpo*

$$GF(2)[x]/(f(x)) = \{\sum_{i=0}^{2} a_i x^i \mid a_i \in GF(2)\}$$

tem 2^3 elementos, que são: $\{0, 1, x, x^2, x+1, x^2+x, x^2+x+1, x^2+1\}$.
Repare que todos os elementos não nulos deste corpo podem ser gerados por x (potências de x) sob a condição, $f(x) = 0$. Como cada elemento de $GF(2)[x]/(f(x))$ é uma classe de resto módulo $f(x)$, em circunstância, $0 = [0]$, $1 = [1]$, $x = [x]$ e assim por diante, estabelecemos que $[x] = \alpha$, então $f(\alpha) = 0$ e o corpo pode ser reescrito assim:

$$GF(2)[x]/(f(x)) = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$$

A tabela 3.2 apresenta diferentes formas de representar os elementos do corpo finito $GF(2^3)$. Tais representações são muito úteis para a aritmética em corpos finitos e, são usadas na descrição dos elementos que definem os códigos correctores de erros, em estudo neste trabalho (cap. 4).

Forma de representação		
potência	polinómio	binária
0	0	000
α^0	1	100
α	α	010
α^2	α^2	001
α^3	$1 + \alpha$	110
α^4	$\alpha + \alpha^2$	011
α^5	$1 + \alpha + \alpha^2$	111
α^6	$1 + \alpha^2$	101

Tabela 3.2: Corpo GF(8)

Da mesma forma podem ser construídas outras extensões de $GF(2)$, como o $GF(2^5)$, $GF(2^6)$ e outras que podem ser consultadas em [17, pág.370].

A representação ilustrada no exemplo 3.4 pode ser vista sob a perspectiva das consequências do seguinte resultado:

Teorema 3.5. *Seja $GF(q)$, $q = p^m$, um corpo finito, o grupo multiplicativo $(GF(q) \setminus \{0\}, \cdot)$ é cíclico.*

Demonstração. Pela definição de corpo finito, $(GF(q) \setminus \{0\}, \cdot)$ é um grupo abeliano finito. Chamemos k_M, a maior ordem de um elemento de $GF(q) \setminus \{0\}$. Da teoria de grupo, afirma-se que em qualquer grupo abeliano finito, a ordem de qualquer elemento divide a ordem máxima e, sendo assim, todo elemento $\beta \in GF(q) \setminus \{0\}$ satisfaz $\beta^{k_M} = 1$. Isto significa que todos os elementos de $GF(q) \setminus \{0\}$ são raízes do polinómio $X^{k_M} - 1$.

Sendo q o número de elemento de $GF(q)$, temos:

1. Pelo teorema fundamental de álgebra, o número de raízes de um polinómio é, no máximo, igual ao seu grau e, $x^{k_M} - 1$ tem $q - 1$ raízes em $GF(q)$ implicando que $q - 1 \leq k_M$;
2. Como k_M é a ordem de um elemento de $GF(q) \setminus \{0\}$ que é um grupo de ordem $q - 1$, então $k_M | (q - 1)$, logo $k_M \leq (q - 1)$.

Pelos itens 1. e 2., tem-se que $k_M = q - 1$. Portanto, existem elementos de $GF(q) \setminus \{0\}$ com ordem $q - 1$, o que significa que $(GF(q) \setminus \{0\}, \cdot)$ é cíclico. □

Corolário 3.6. *Seja $GF(q)$, um corpo finito com $q = p^m$:*

1. *Existe sempre um elemento, α, de ordem $p^m - 1$ a que chamamos **elemento primitivo** de $GF(q)$;*
2. *Os elementos de $GF(q)$ são as raízes do polinómio $x^{p^m} - x$.*

Desta forma, sendo α um elemento primitivo de $GF(p^m)$, então pode-se escrever que:

$$GF(p^m) = \{0\} \cup \{\alpha^i \mid 0 \leq i \leq p^m - 2\}$$

Fica a ideia de que os elementos que constituem o corpo finito dependem apenas da característica do corpo e do grau do polinómio - são todos zeros de $x^{p^m} - x$.

De facto, dado os inteiros p e m, **existe um único corpo finito** $GF(p^m)$ no sentido em que todos os que existirem sejam isomorfos.

Teorema 3.7. *Se α é um elemento primitivo de $GF(p^m)$, então para inteiros, i, α^i é também elemento primitivo de $GF(p^m)$ se e só se $mdc(i, p^m - 1) = 1$.*

Demonstração.

$\Rightarrow$ Seja α e α^i elementos primitivos de $GF(p^m)$ para i, inteiros. Vamos mostrar que $mdc(i, p^m - 1) = 1$.

Com efeito, Vamos supor que $mdc(i, p^m - 1) = d \neq 1$. Então, podemos escrever $i = i_0 d$ e $p^m - 1 = p_0 d$, com i_0 e p_0 inteiros. Assim é possível ter:

$$\begin{aligned} (\alpha^i)^{p_0} &= \alpha^{i_0 p_0 d} \\ &= (\alpha^{p^m - 1})^{i_0} = 1 \end{aligned}$$

Isto quer dizer que a $Ord(\alpha^i)$ divide p_0, com $p_0 < p^m - 1$, o que contraria a hipótese. Logo, $mdc(i, p^m - 1) = 1$.

$\Leftarrow$ Seja α um elemento primitivo de $GF(p^m)$ e supomos i, inteiro tal que $mdc(i, p^m - 1) = 1$. Vamos mostrar que α^i é, também, um elemento primitivo.

Com efeito, como α tem ordem $p^m - 1$, então $\alpha^i \in GF(p^m)$ tem ordem k, tal que $k|(p^m - 1)$. Por outro lado, pela definição de ordem, $(\alpha^i)^k = 1 = \alpha^{ik}$ implica que $(p^m - 1)|ik$. Mas, como $mdc(i, p^m - 1) = 1$, então $(p^m - 1)|k$. Logo, k, a ordem de α^i, é igual a $p^m - 1$.

□

Do resultado 3.7 segue que existem $\phi(p^m - 1)$ elementos primitivos em $GF(p^m)$, onde $\phi(n)$ denota a função de Euler. Esta função devolve o número de inteiros entre 1 e n que são primos relativos com n.

É conveniente o conhecimento sobre algumas relações entre os elementos primitivos ou geradores do grupo multiplicativo, os polinómios irredutíveis sobre um corpo e os demais elementos do corpo por forma a munirmos de ferramentas necessárias para o trabalho em corpos finitos

3.2 Polinómios mínimos

Definição 3.8. ***Polinómio mínimo,*** *$M_\beta(x)$, de um elemento $\beta \in GF(p^m) \setminus \{0\}$ é um polinómio mónico de menor grau do qual β é zero.*

Propriedades 3.9. *[20][Polinómios mínimos]*

1. *$M_\beta(x)$ é irredutível.*
2. *se $f(\beta) = 0$, então $M_\beta(x)$ divide $f(x)$ para todo $f(x)$ de $GF(p)[x]$.*
3. *$M_\beta(x)$ divide $x^{p^m} - x$.*
4. *O grau de $M_\beta(x)$ é $\leq m$.*
5. *O polinómio mínimo de um elemento primitivo de $GF(p^m)$ tem grau m.*
6. *β e β^p têm o mesmo polinómio mínimo.*

A última propriedade permite dispor os elementos do corpo em classes distintas de elementos que partilham o mesmo polinómio mínimo. Veja que a multiplicação por p agrupa os inteiros modulo p^m-1 em classes de resto modulo p^m-1 - classes ciclotómicas.

Exemplo 3.10. *Seja $p = 2$, $m = 4$. Os inteiros módulo 15 são dispostos nos seguintes classes ciclotómicas:*

$$\begin{aligned} C_0 &= \{0\} \\ C_1 &= \{1,\ 1 \cdot 2 = 2,\ 2 \cdot 2 = 4,\ 4 \cdot 2 = 8,\ 8 \cdot 2 = 16 \equiv 1\} \\ C_3 &= \{3, 6, 12, 9\} \\ C_5 &= \{5, 10\} \\ C_7 &= \{7, 14, 13, 11\} \end{aligned}$$

Assim, se α é um elemento primitivo de $GF(2^4)$, então,

$$GF(2^4) = \{\underbrace{0}_{L_0}, \underbrace{1}_{L_1}, \underbrace{\alpha, \alpha^2, \alpha^4, \alpha^8}_{L_\alpha}, \underbrace{\alpha^3, \alpha^6, \alpha^{12}, \alpha^9}_{L_{\alpha^3}}, \underbrace{\alpha^5, \alpha^{10}}_{L_{\alpha^5}}, \underbrace{\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}}_{L_{\alpha^7}}\}$$

Os elementos $\alpha, \alpha^2, \alpha^4$ e α^8 têm o mesmo polinómio mínimo. Como o número de zeros determina o grau do polinómio, este corpo tem três polinómios mínimos de grau 4 cujo

os zeros são elementos dos conjuntos L_α, L_{α^3} e L_{α^7} respectivamente. De entre eles, apenas dois, os que têm elementos primitivos como raízes, podem ser usados para gerar o corpo $GF(2^4)$.

Pelo teorema 3.7, os elementos primitivos de $GF(2^4)$ são os elementos que estão em L_α e em L_{α^7} e portanto, só os polinómios correspondentes podem gerar este corpo. São eles, x^4+x^3+1 e x^4+x+1.

Os polinómios mínimos cujas raízes são elementos primitivos do corpo, são denominados **polinómios primitivos**, e vão nos permitir definir os corpos finitos.

Definição 3.11 (Conjugado)**.** *Seja $GF(p^m)$ uma extensão de $GF(p)$ e seja $\alpha \in GF(p^m)$. Os elementos $\alpha, \alpha^p, \alpha^{p^3}, \ldots, \alpha^{p^{m-1}}$ são chamados **conjugados de** α relativamente a $GF(p)$. Ao conjunto destes elementos denomina-se **classe dos conjugados de** $GF(p^m)$ **sobre** $GF(p)$.*

No exemplo 3.10, o conjunto L_α é formado pelos conjugados de α relativamente a $GF(2)$.

Teorema 3.12. *[17, pág. 49] Os conjugados de $\alpha \in GF(p^m)$ relativamente a qualquer subcorpo de $GF(p^m)$ têm a mesma ordem no grupo $(GF(p^m) \setminus \{0\}, \cdot)$.*

Corolário 3.13. *Se α é um elemento primitivo de $GF(p^m)$ então, são primitivos todos os seus conjugados relativamente a qualquer sub-corpo de $GF(p^m)$.*

Em suma, sendo α um elemento primitivo de $GF(p^m)$ pode-se, por via de classes ciclotómicas[1] ou por via de conjugados, conhecer todos os polinómios mínimos deste corpo. E, combinando o resultado 3.7 torna possível identificar todos os polinómios primitivos.

3.3 Grupo de automorfismos de $GF(p^m)$

Existe uma relação íntima entre elementos conjugados e certos automorfismos de um corpo finito. Seja $GF(p^m)$ uma extensão de $GF(p)$. Uma transformação ρ definida de

[1] Observe que os elementos de cada classe constituem expoente das potência de α em cada classe de conjugado

$GF(p^m)$ sobre si mesmo que fixa todos os elementos de $GF(p)$ e preserva a adição e multiplicação é um automorfismo de $GF(p^m)$ sobre $GF(p)$. Simbolicamente:

$$\begin{aligned} \rho \;\; : \;\; & GF(p^m) \to GF(p^m) \\ & a \mapsto \rho(a) \\ & \text{tal que } \rho(b) = b \; se \; b \in GF(p) \end{aligned}$$

e, $\forall \alpha$ e β de $GF(p^m)$,

$$\begin{cases} \rho(\alpha + \beta) = \rho(\alpha) + \rho(\beta) \\ \rho(\alpha \cdot \beta) \;\; = \;\; \rho(\alpha) \cdot \rho(\beta) \end{cases} .$$

Teorema 3.14. *[17, pág. 49] Os diferentes automorfismos de $GF(p^m)$ sobre $GF(p)$ são, exactamente, as transformações $\rho_1, \rho_2, \ldots, \rho_{m-1}$ definidas por:*

$$\rho_j(\alpha) = \alpha^{p^j}, \qquad \forall \alpha \in GF(p^m) \qquad e \qquad 0 \leq j \leq m-1.$$

Claro está que os conjugados de $\alpha \in GF(p^m)$ relativamente a $GF(p)$ são obtidos aplicando todos os automorfismos de $GF(p^m)$ sobre $GF(p)$ ao elemento α. Seja α um elemento primitivo de $GF(p^m)$, então:

$$\rho_0(\alpha) = \alpha^{p^0}, \qquad \rho_1(\alpha) = \alpha^{p^1}, \qquad \rho_2(\alpha) = \alpha^{p^2}, \qquad \ldots \qquad \rho_{m-1}(\alpha) = \alpha^{p^{m-1}}.$$

Os automorfismos de $GF(p^m)$ sobre $GF(p)$ formam um grupo em relação à composição usual de transformações a que chamamos **grupo de automorfismos de** $GF(p^m)$. Pelo teorema 3.14, tal grupo é cíclico de ordem m e gerado por ρ_1. Em particular, se $p = 2$ temos que $\rho_1(\alpha) = \alpha^2$ é o **automorfismo de Frobenius**, e o grupo gerado por ρ_1 é designado por grupo dos automorfismos de Frobenius.

O grupo dos automorfismos de Frobenius é referenciado mais à frente (em 6.2) associado aos códigos binários de Goppa com polinómio gerador binário e como estando contido ao grupo de automorfismos deste código (ver [19]). A sua acção deixa invariante um conjunto de palavras de códigos que, por sua vez, definem um sub-código (sub-código idempotente) que joga um papel importante na determinação de código equivalente ao que é públicado e num consequente ataque estrutural ao sistema criptográfico de McEliece.

3.4 Cálculos básicos sobre $GF(p^m)$

Pretendemos, aqui, exemplificar alguns cálculos básicos sobre o corpo finito $GF(p^m)$ por forma a facilitar a leitura e compreensão da secção 4.2 na qual definimos o código de Goppa e damos exemplo de matriz de paridade com elementos de $GF(2^m)$.

A **adição** é feita com facilidade usando a representação vectorial (binária)[2] ou polinomial dos elementos do corpo. Com a representação binária, a adição coincide com a operação ***XOR*** e, é feita componente a componente.

A **multiplicação** pode ser feita com os elementos na forma polinomial sendo necessário, quase sempre, fazer a redução módulo o polinómio primitivo que gerou o corpo ou seja, quando o produto resultar em grau superior ao do polinómio primitivo, toma-se o resto da divisão por ele descartando o quociente. Todavia, é mais conveniente usar a representação exponencial. Neste caso, o expoente do produto é reduzido módulo $p^m - 1$.

A **divisão** pode ser percebida como a multiplicação pelo inverso multiplicativo. $\frac{a}{b} = a \times \frac{1}{b}$. Todo elemento $b \neq 0$ do corpo admite inverso, portanto, é preciso saber calculá-lo e, assim perfazer a divisão.

O **inverso multiplicativo** pode ser calculado via algoritmo extendido de Euclides para polinómios visto na secção 2.2.1. Mas também, podemos tirar proveito da ordem do corpo $(p^m - 1)$. Todo elemento do corpo pode ser expresso como potência do elemento gerador do grupo multiplicativo. Assim, encontrar o inverso de um elemento α^l, consiste em determinar um elemento I tal que $\alpha^l \times I = \alpha^{p^m-1} = 1$ ou seja $I = \alpha^{p^m-1-l}$. Convém reduzir, sempre, o expoente módulo $p^m - 1$.

Exemplo 3.15. *Seja o corpo finito $GF(2^4)$ definido pelo polinómio primitivo $f(x) = x^4 + x + 1$ e cujo os elementos constam da tabela 3.3:*

a) *Calcula $\alpha^8 + \alpha^{13}$:*

$$\begin{aligned} \alpha^8 + \alpha^{13} &= \alpha^2 + 1 + \alpha^3 + \alpha^2 + 1 \\ &= \alpha^3 \end{aligned}$$

[2]binária, no nosso caso, porque só nos interessa o corpo $GF(2^m)$

Formas de representação					
exponencial	polinomial	binária	exponencial	polinomial	binária
0	0	0000	α^7	$1+\alpha+\alpha^3$	1101
$\alpha^0=\alpha^{15}=1$	1	1000	α^8	$1+\alpha^2$	1010
α	α	0100	α^9	$\alpha+\alpha^3$	0101
α^2	α^2	0010	α^{10}	$1+\alpha+\alpha^2$	1110
α^3	α^3	0001	α^{11}	$\alpha+\alpha^2+\alpha^3$	0111
α^4	$1+\alpha$	1100	α^{12}	$1+\alpha+\alpha^2+\alpha^3$	1111
α^5	$\alpha^2+\alpha$	0110	α^{13}	$1+\alpha^2+\alpha^3$	1011
α^6	$\alpha^3+\alpha^2$	0011	α^{14}	$1+\alpha^3$	1001

Tabela 3.3: Corpo GF(16)

ou na forma binária:

$$\begin{aligned} \alpha^8+\alpha^{13} \quad &: \quad 1010 \\ &+ \quad \underline{1011} \\ &\quad\;\; 0001=\alpha^3 \end{aligned}$$

b) *Calcula* $\alpha^8\times\alpha^{13}$.

$$\begin{aligned} \alpha^8\times\alpha^{13} &= \alpha^{21 \mod 15)} \\ &= \alpha^6 \end{aligned}$$

ou,

$$\begin{aligned} \alpha^8\times\alpha^{13} &= (\alpha^2+1)(\alpha^3+\alpha^2+1) \\ &= \alpha^5+\alpha^4+\alpha^3+1 \\ &= \alpha^3+\alpha^2=\alpha^6 \end{aligned}$$

Este resultado é alcançado como o resto da divisão de $\alpha^5+\alpha^4+\alpha^3+1$ *por* $f(\alpha)$ *ou, substituindo* α^5 *e* α^4 *por expressões da tabela 3.3.*

c) *Calcula* $(\alpha^8)^{-1}$

$$\begin{aligned} (\alpha^8)^{-1} &= \alpha^{15-8} \\ &= \alpha^7 \end{aligned}$$

d) *Calcula* $\frac{\alpha^{11}}{\alpha^8}$*: como* $\frac{1}{\alpha^8} = \alpha^7$*, então:*

$$\begin{aligned} \frac{\alpha^{11}}{\alpha^8} &= \alpha^{11} \times \alpha^7 \\ &= \alpha^{18 \mod 15} \\ &= \alpha^3 \end{aligned}$$

Capítulo 4

Códigos

A ideia intuitiva subjacente ao uso e construção de códigos pode ser vista no código de uso mais frequente entre nós os humanos. A língua em que nos comunicamos. A portuguesa, por exemplo, é constituída por um alfabeto de 26 letras. As palavras não são mais do que uma sequência de letras e, consequentemente, a língua é composta por palavras. É evidente que o conjunto das palavras que constituem a língua, não contém todas as palavras possíveis formada pelas letras do alfabeto. É claro o reconhecimento de algumas palavras como fazendo parte da língua e outras não.

O processo de comunicação nem sempre se dá a 100%. A transmissão de mensagem entre emissor e receptor muitas vezes ocorre com erros que dificultam a compreensão da mesma e, em certos casos até, ocasionam pedidos de retransmissão. Vamos supor que recebemos uma mensagem na qual constam palavras como *códrgos* ou *qaco*. É imediato a percepção de que tais palavras não fazem parte da língua portuguesa, têm um erro. Além do mais, quanto à primeira palavra, intuímos que, na mensagem, a palavra correcta devia ser *códigos* por esta ser a "palavra mais próxima de" *códrgo* na língua. Quanto à palavra *qaco*, percebemos que há várias hipóteses de correcção. As palavras *caco*, *paco*, *taco*, *saco*, etc. são todas igualmente próximas da palavra transmitida.

Fica estabelecido, ainda que de forma intuitiva, que para construção de códigos é preciso um conjunto finito de símbolos cujas sequências definem palavras que, conforme veremos mais à frente, denominamos de *palavras de código*, ou não, e dotá-lo de certas propriedades que permitem a detecção e correcção de erros. Estamos a falar de códigos correctores de erros primeiramente introduzidos por Hamming em [12] onde ele descreve

um código, que veio a ter o seu nome, que detecta dois erros e corrige um se for único.

Neste capítulo tentaremos definir de forma geral os códigos lineares como espaços vectoriais sobre corpos finitos e imediatamente restringir aos códigos binários. Procuraremos formalizar o conceito intuitivo de "palavra mais próxima de" que se concretiza no conceito de *distância mínima de Hamming* com base na qual é desenvolvido o processo de correcção de erros. A representação matricial dos códigos lineares binários e consequente processo de codificação/descodificação é apresentada. Como exemplo de concretização, descreveremos o *código binário de Goppa* que é utilizado na construção do sistema criptográfico de McEliece a ser apresentado no capítulo 5.

4.1 Códigos Lineares

Seja $GF(q)$ um corpo finito e $GF(q)^n$ o espaço vectorial de todos os n-uplos. O código C (n,M) sobre $GF(q)$, é um subconjunto de $GF(q)^n$ com M elementos. Se $M = q^k$, então C é denominado código - $[n, k]$. No caso de $q = 2$, o código é dito **binário**.

Normalmente, representamos os vectores $(a_1, a_2, \ldots, a_n)$ pertencentes a $GF(q)^n$ na forma $a_1a_2 \ldots a_n$ e, denominamos os vectores pertencente ao código C por **palavras de código**. Outra forma de especificar uma palavra de código é através da representação polinomial usada para códigos cíclicos[1].

Sem imposições ao nível de estrutura, a utilidade dos códigos é um pouco limitada. A **linearidade** é a mais útil estrutura adicional a impor (ver [14]). Assim, definimos um código linear de seguinte forma:

Definição 4.1. *Um código linear C, de comprimento n, é um subespaço linear de $GF(q)^n$, onde $GF(q)$ é um corpo finito com q elementos.*

Como C é um subespaço de $GF(q)^n$, então existe uma base $\{c_1, c_2, \ldots, c_k\}$, com $c_i \in GF(q)^n$ onde $k \leq n$ é a dimensão do subespaço, logo dimensão do código. C é designado de *código linear* - $[n, k]$ e qualquer palavra deste código pode ser expressa como combinação linear dos vectores dessa base. Sendo assim, um código linear, C, é representável matricialmente.

[1] pormenores sobre este assunto pode ser encontrado em [20, cap. 7]

Geralmente, utiliza-se duas formas para representar um código linear: através de uma **matriz geradora** ou através de uma **matriz de paridade**.

4.1.1 A Matriz Geradora, G, e matriz de paridade, H

.

Definição 4.2. *Seja C, um código linear - $[n, k]$. Uma matriz G, $k \times n$, cujas linhas formam uma base para o código C, chama-se* ***matriz geradora de*** *C. Em geral existem várias matrizes geradoras para o mesmo código no sentido de que, quaisquer k vectores de C linearmente independentes formam uma base.*

Uma matriz geradora determina um processo de codificação na medida em que transforma uma mensagem $u \in GF(q)^k$ numa palavra de código $x \in GF(q)^n$. Simbolicamente:

$$\begin{array}{rcl} \text{C: } GF(q)^k & \to & GF(q)^n \\ \text{u} & \mapsto & x = uG \end{array}$$

Assim, o conjunto de todas as palavras de código é:

$$C = \{x = uG \mid u \in GF(q)^k\} \tag{4.1}$$

Podemos impor, a menos de uma permutação de colunas, que um código linear - $[n, k]$ admite uma matriz geradora (e, consequentemente, todas) $G = [G_1|G_2]$, sendo G_1 uma matriz $k \times k$ e invertível.

Proposição 4.3. *Todo o código linear - $[n, k]$ admite uma única matriz geradora da forma $[I_k|A]$, onde I_k é matriz identidade $k \times k$ e A é uma matriz do tipo $k \times (n - k)$. Diz-se que tal matriz está na forma padrão.*

Demonstração. Com efeito, seja G uma matriz geradora de um código $C - [n, k]$ - subespaço linear do espaço vectorial $GF(q)^n$. Então, o espaço de linhas de G contem k vectores de $GF(q)^n$ linearmente independentes. Este espaço não altera com as operações elementares sobre as linhas. Sendo assim, é possível, por uma sequência de operações

elementares, obter $G' = [I_k|A]$. Portanto, tal matriz na forma padrão existe. É único: vamos supor que $G_1 = [I_k|A_1]$ e $G_2 = [I_k|A_2]$ são diferentes e geram o mesmo código. Assim $x = uG_1 = uG_2$ ou seja $u[I_k|A_1] = u[I_k|A_2]$ o que implica $A_1 = A_2$. □

As palavras de código produzidas por uma matriz geradora na forma padrão têm como primeiros k bits os da mensagem que codifica. Os bits restantes são denominados bits de paridade e possibilitam a recuperação da mensagem original.

Exemplo 4.4. *Tomemos o exemplo de um código C-[n,k] sobre GF(2): para o código* $C - [7, 4]$ *de Hamming (ver [22]), uma matriz geradora possível é*

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right]$$

Este código transforma blocos com quatro bits de mensagem em palavras de código com sete bits. os três bits adicionais são de verificação. Seja $u = u_1u_2u_3u_4$ *a mensagem a codificar. A mensagem codificada seria* $x = u \times G = x_1x_2x_3x_4x_5x_6x_7$ *(com 7 bits), de onde se tem:* $x_1 = u_1$, $x_2 = u_2$, $x_3 = u_3$, $x_4 = u_4$ *e os restantes bits (de verificação)* x_5, x_6 *e* x_7 *são tais que:*

$$\begin{cases} x_5 \equiv x_2 + x_3 + x_4 \\ x_6 \equiv x_1 + x_3 + x_4 \mod 2 \\ x_7 \equiv x_1 + x_2 + x_4 \end{cases}$$

Definição 4.5 (Matriz de paridade-H)**.** *Matriz de paridade de um código linear C-[n,k], é uma matriz, H, de dimensão* $(n - k) \times n$*, de linhas linearmente independentes tais que:* $Hx^T = 0 \; \forall x \in C$.

Tal definição sugere-nos que um código linear C pode ser, alternativamente, definido da seguinte forma:

$$C = \{x \in GF(q)^n \mid Hx^T = 0\} \tag{4.2}$$

O facto de que cada linha de G é um vector de C, a equação (4.2) implica a seguinte relação entre as duas matrizes:

$$HG^T = 0 \tag{4.3}$$

Mais ainda, se combinarmos as equações (4.2) e (4.1) obteremos a forma padrão da matriz de paridade. Com efeito, considerando $G = [I_k|A]$, um codificador de C na forma padrão, tem-se que $Hx^T = 0 \ \forall x \in C \ \Rightarrow \ H(uG)^T = 0 \ \forall u \in GF(p)^k \ \Rightarrow$ $HG^T u^T = 0 \ \forall u \in GF(p)^k \ \Rightarrow \ HG^T = 0 \ \Rightarrow \ H \left[\begin{array}{c} I_k \\ \hline \\ A^T \end{array}\right] = 0$.

Concluimos assim que $H = [-A^T|I_{n-k}]$ é uma matriz de paridade de C que se diz estar na forma padrão.

Exemplo 4.6. *O código C do exemplo 4.4 teria uma matriz de paridade H tal que:*

$$H = \left[\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Para o corrente exemplo, a matriz $-A^T = A^T$ (porque está definida sobre $GF(2)$) é:

$$A^T = \left[\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{array}\right]$$

Na verdade, a matriz de paridade de um código C é matriz geradora de um outro código que denotamos de $C^\perp$ - código dual de C.

Definição 4.7. *[Código dual] Seja C um código linear sobre $GF(q)^n$. O código dual, $C^\perp$, de C é definido da seguinte forma:*

$$C^\perp = \{x \in GF(q)^n \mid x \cdot y = 0, \quad \forall y \in C\}$$

onde $x \cdot y$ denota o produto interno canónico entre x e y.

Definição 4.8. *[20, pág. 24 e 39][Códigos lineares equivalentes] Dois códigos lineares, C e C', são equivalentes se diferem apenas na ordem dos seus símbolos.*

Exemplo 4.9.

$$C = \left\{\begin{array}{l} 0000 \\ 0011 \\ 1100 \\ 1111 \end{array}\right. \qquad C' = \left\{\begin{array}{l} 0000 \\ 0101 \\ 1010 \\ 1111 \end{array}\right.$$

Uma definição mais formal de códigos lineares binários equivalentes é dada na subsecção 6.2.3.

4.1.2 Descodificação - código linear

Seja u uma mensagem codificada como $x = uG$, que é transmitida através de um determinado canal. Todavia, o que o descodificador recebe pode não ser exactamente x. Os canais de transmissão são afectados por ruído que pode distorcer as palavras de código adicionando-lhes vectores de erro, e, recebendo-se então, $y = x + e$ onde $e = e_1 e_2 \dots e_n$ pertencente a $GF(q)^n$. Cabe ao descodificador decidir, a partir de y, qual terá sido o x transmitido. Seria suficiente se o descodificador pudesse identificar o erro, e. Neste caso, bastava calcular $x = y - e$. Mas tal não é possível, o descodificador nunca tem a certeza de e, pelo que a estratégia é escolher o erro mais provável atendendo ao y recebido. Em última análise, esta estratégia passa por descodificar y como sendo a palavra de código mais próxima de x. Uma vez que os elementos u, x, y e e são vectores, a ideia do "próximo de" é alcançada com a introdução do conceito de distância.

O espaço vectorial $GF(q)^n$ pode ser considerado um espaço métrico se, sobre ele, definirmos a **distância de Hamming** entre dois vectores x e y.

Definição 4.10. *[Distância de Hamming] A distância de Hamming entre dois vectores x e y de $GF(q)^n$, é o número de elementos em que os dois vectores diferem. Simbolicamente:*

$$d_H(x, y) = |\{i : x_i \neq y_i,\ 1 \leq i \leq n\}| \tag{4.4}$$

onde $|Z|$ representa número de elementos de um conjunto Z.

Propriedades 4.11. *[14, pág. 8] A distância, d_H, satisfaz as seguintes propriedades:*

1. $d_H(x, y) \geq 0\ \forall x, y \in GF(q)^n$;
2. $d_H(x, y) = 0 \ \Leftrightarrow\ x = y$;
3. $d_H(x, y) = d_H(y, x)\ \forall x, y \in GF(q)^n$;
4. $d_H(x, z) \leq d_H(x, y) + d_H(y, z)\ \forall x, y, z \in GF(q)^n$.

Para definirmos, de entre várias palavras de código, uma, x, que seja mais próxima da palavra recebida, y, é necessário estabelecer a **distância mínima** de um código.

Definição 4.12. *[Distância mínima] Seja $C \subset GF(q)^n$ um código. A distância mínima de C é d tal que:*

$$d = min\{d_H(x, y)|\ x, y \in C,\ x \neq y\}.$$

A definição seguinte e a sua relação com o conceito de distância de Hamming facilita, enormemente, o cálculo de distância mínima de um código linear.

Definição 4.13. *[Peso de Hamming] Peso de Hamming de um vector* $x = x_1x_2 \dots x_n$ *é o número de* x_i *não nulos. Simbolicamente:*

$$wt(x) = |\{i : x_i \neq 0,\ 1 \leq i \leq n\}|.$$

Teorema 4.14. *Se* x *e* $y \in C$ *- código linear então,* $d_H(x, y) = wt(x - y)$.

Demonstração. Sendo C um código linear, então contém o vector nulo e $x - y$, para todos $x, y \in C$. Assim, $d_H(x, y) = d_H(x - y, 0) = wt(x - y)$. □

Do teorema 4.14, segue-se que:

$$d = min\ \{wt(z) \mid z \in C\ e\ z \neq 0\} \tag{4.5}$$

Observação 4.15. *A utilidade de um código reside na sua capacidade de corrigir erros. Tal capacidade é definida através da sua distância mínima. Se* $d = 2t + 1$, *o código corrige até* t *erros. Isto pode ser ilustrado da seguinte forma: na fig.:4.1-a) as esferas de raio* t *centradas nas palavras de código,* c_1 *e* c_2, *não se intersectam. Desta forma, qualquer palavra com erro em até* t *posições, seja ela* c', *estará no interior ou sobre a fronteira de uma determinada esfera, o que permite, sem ambiguidade, descodificá-la como a palavra de código sobre a qual a esfera está centrada, neste caso* c_1. *A fig.:4.1-b, é um caso em que* $d < 2t + 1$. *Nesta situação, apesar de* c' *possuir menos do que* t *erros, o descodificador tem dificuldade em decidir. De facto,* c' *pode tanto ser* $c_1 + z_1$ *como* $c_2 + z_2$, *com* z_1 *e* z_2 *vectores erro de peso menor que* t.

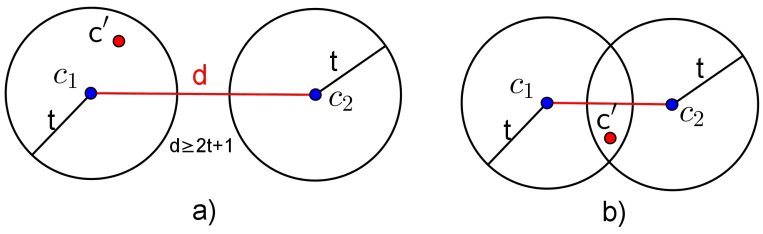

Figura 4.1: Distância mínima de um código C

Estabelecido o conceito de distância mínima, passamos a caracterizar um código linear, essencialmente, por três parâmetros a saber: comprimento, n, dimensão, k, e distância mínima, d. Assim, diz-se que C é um **código linear -** $[n, k, d]$.

A distância mínima ou peso mínimo de um código linear pode, também, ser encontrada a partir da sua matriz de paridade. O resultado seguinte mostra-nos como:

Teorema 4.16. *Seja um código linear C e H uma matriz de paridade de C. Então a distância mínima, d, de C é o número mínimo de colunas de H linearmente dependentes.*

Demonstração. As palavras de código são vectores, x, do espaço vectorial $GF(q)^n$ tais que $Hx^T = 0$ (def.:4.5). O produto Hx^T expressa uma combinação linear de colunas de H. Com efeito, tomemos H_i com $i = 1, \ldots, n$, como as colunas de H. Segue, então que $Hx^T = \sum_{i=1}^{n} H_i x_i$. Desta forma, uma palavra de código de peso w estabelece uma dependência linear entre w colunas de H. Combinando isto com (4.5), fica demonstrado o teorema. □

4.1.3 Descodificação por síndrome

A descodificação por síndrome tira proveito da definição de código linear implícita na relação (4.2), $x \in C$ se e só se $Hx^T = 0$. Assim, se um descodificador receber um vector y, bastará testá-lo para saber se houve erro ou não durante a transmissão: se $Hy^T = 0$ então, não houve erro na transmissão e y é a palavra de código transmitida. Por outro lado, se $Hy^T \neq 0$ fica-se a saber que y não é a palavra de código transmitida ou seja, $y = x + e$, onde e é o vector erro introduzido pelo canal e x a palavra de código transmitida.

Ao vector $s(y) = Hy^T$, chama-se **síndrome de** y. Uma propriedade fundamental da síndrome é que ela depende exclusivamente do erro introduzido e não da palavra de código transmitida. Com efeito:

$$\begin{aligned} s(y) = Hy^T &= H(x+e)^T \\ &= Hx^T + He^T \\ &= He^T \; porque, \; x \in C. \end{aligned}$$

Este facto implica que, para cada e fixo existe um conjunto

$$\{e + x \mid x \in C\} \tag{4.6}$$

com q^k vectores que têm a mesma síndrome. Desta forma, imaginemos uma tabela em que as linhas fossem tais conjuntos. A primeira linha é constituída pelas palavras de código incluindo o vector nulo. A primeira coluna é formada pelos vectores erro. Seria possível, através dela, identificar a palavra recebida, na posição (i, j), e saber que o erro introduzido é $e_{i,1}$ e que a palavra de código é $x_{1,j}$. Esta ideia concretiza-se no que se chama "standard array decoding" que é descrito mais pormenorizadamente, por exemplo, em [28, pág. 12]. A definição seguinte permitirá por um lado, estabelecer um conjunto de propriedades a que o conjunto definido em (4.6) deve satisfazer e, por outro lado, em combinação com conceito de síndrome, descrever um algoritmo prático de **descodificação por síndrome**.

Definição 4.17 (Coset)**.** *Seja C um código linear - $[n, k]$ sobre $GF(q)^n$. Para qualquer vector $a \in GF(q)^n$, o conjunto $a + C = \{a + x \mid x \in C\}$ é designado* ***coset de C.***

Propriedades 4.18. *Seja C um código linear - $[n, k]$ sobre $GF(q)^n$. Dado quaisquer vectores a e $b \in GF(q)^n$, tem-se:*

1. *b está em algum coset de C.*
2. *a e b estão no mesmo coset se e só se $(a - b) \in C$.*
3. *Todo coset contém q^k vectores.*

Em cada coset, o vector de menor peso é denominado **líder do coset**.

Proposição 4.19. *[20, pág. 15] Dois cosets ou são coincidentes ou são disjuntos.*

Estamos em condições de provar o seguinte resultado que estabelece uma relação de um por um entre cosets e síndromes:

Teorema 4.20. *Dois vectores pertencem ao mesmo coset se e só se têm o mesmo síndrome.*

Demonstração. Sejam v_1 *e* v_2 vectores de $GF(q)^n$ e C um código linear - $[n,k]$. Então,

$$\begin{aligned} & v_1 \text{ e } v_2 \text{ pertencem ao mesmo coset} \\ \Leftrightarrow\ & (v_1 - v_2) \in C \text{ (por definição de coset)} \\ \Leftrightarrow\ & H(v_1 - v_2)^T = 0 \text{ (por definição de código)} \\ \Leftrightarrow\ & Hv_1^T - Hv_2^T = 0 \\ \Leftrightarrow\ & Hv_1^T = Hv_2^T \text{ (ou seja } v_1 \text{ e } v_2 \text{ têm o mesmo síndrome)} \end{aligned}$$

□

Posto isto, e no que refere à descodificação de um código linear - $[n,k]$, é possível construir uma tabela de síndromes com os respectivos lideres dos cosets. Como C é um subgrupo abeliano do grupo aditivo $GF(q)^n$, tal tabela terá q^{n-k} (número de cosets = classes laterais de $GF(q)$ módulo C) entradas ao invés de q^n como acontece no "standard array decoding". Vamos considerar um exemplo de descodificação.

Exemplo 4.21. *O código do exemplo 4.4 tem matriz de paridade* H *dada por:*

$$H = \left[\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Vamos supor que foi enviada uma palavra $x = 1011010$ *através de um canal que adiciona um erro* $e = 0010000$. *O descodificador recebe* $y = 1001010$. *Para descodificar* y, *podemos proceder da seguinte forma:*

1. *Construir a tabela de síndrome (tabela 4.1);*
2. *Calcular o síndrome de* y: $s(y) = Hy^T = 110$;
3. *Consultando a tabela de síndrome, vê-se que o líder do coset correspondente é* 0010000 *que é assumido como erro de transmissão;*
4. *Por fim, descodifica-se* y *como* $x = y + e = 1011010$.

Nota 4.22. *Nem sempre é viável a descodificação por síndrome. Vamos supor que fosse conhecido o comprimento,* n, *da mensagem e quanto ao erro, soubéssemos apenas, que o peso é menor ou igual a* t *e quiséssemos descodificar tal mensagem. Neste caso, teriamos* $\sum_{w=1}^{t} \binom{n}{w}$ *erros possíveis. Isto quer dizer que se* $n = 1024$, $t = 50$ *teríamos um*

Padrão de erro	Síndrome
0000000	000
1000000	011
0100000	101
0010000	110
0001000	111
0000100	100
0000010	010
0000001	001

Tabela 4.1: Tabela de síndrome

valor na ordem de 10^{85}*, o que torna o problema intratável. Mais ainda, o problema de descodificação de um código linear genérico pertence à classe* $\mathcal{NP}$*-completo (ver [2]). Este facto é usado por McEliece na construção do seu sistema criptográfico que será abordado no capítulo 5.*

4.2 Códigos de Goppa

Os códigos de Goppa formam uma classe de códigos lineares. Constituem a mais interessante subclasse do códigos alternantes[2] [20]. São descritos em termos de um polinómio gerador, chamado polinómio de Goppa. Ao contrário dos códigos lineares em geral, para os Goppa, existe um algoritmo eficiente de descodificação. O nosso interesse em estudá-los prende-se com a necessidade de usá-los, na secção seguinte, para explicar a construção do sistema criptográfico de McEliece.

Definição 4.23 (Código de Goppa)**.** *Sejam* m *e* t*, inteiros positivos,* $g(X) = \sum_{i=1}^{t}(g_i X^i)$ *com* $g_i \in GF(q^m)$ *um polinómio mónico de grau* t *e* $\boldsymbol{L} = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\}$*, um conjunto com* n *elementos distintos de* $GF(q^m)$ *tal que* $g(\gamma_i) \neq 0\ \forall \gamma_i \in \boldsymbol{L}$*.*
Para todo vector $c = (c_1, c_2, \ldots, c_n) \in GF(q)^n$*, define-se o síndrome de* c *por*

$$S_c = -\sum_{i=0}^{n-1} \frac{c_i}{g(\gamma_i)} \frac{g(X) - g(\gamma_i)}{X - \gamma_i} \mod g(X). \tag{4.7}$$

O código de Goppa*,* $\mathcal{G}(L, g(X))$*, definido sobre* $GF(q)$ *é o conjunto de todo* $c \in$

[2]Para detalhes sobre códigos alternantes, consultar, por exemplo, [20, pág. 333]

$GF(q)^n$ tal que

$$S_c \equiv 0 \mod g(X) \tag{4.8}$$

ou equivalentemente, tal que $S_c = 0$ no anel de polinómios $GF(q)[X]/(g(X))$. Se $g(X)$ é irredutível então, $\mathcal{G}$ é denominado ***código de Goppa irredutível****.*

Em particular, quando $q = 2$, temos **código binário de Goppa**, classe sobre a qual fica restringido o nosso estudo.

Note-se que da equação (4.7), para cada $0 \leq i \leq n-1$, $h(X) = \frac{g(X)-g(\gamma_i)}{g(\gamma_i)(X-\gamma_i)}$ é um polinómio visto que $(X - \gamma_i)$ divide $(g(X) - g(\gamma_i))$ e, é único e de grau inferior a t - uma consequência do algoritmo de Euclides. Nestas condições, temos que

$$\begin{aligned}(X-\gamma_i)h(X) = \frac{g(X)}{g(\gamma_i)} - 1 \quad &\Leftrightarrow \quad (X-\gamma_i)h(X) + 1 = g(X)g(\gamma_i)^{-1} \Rightarrow \\ &\Rightarrow \quad (X-\gamma_i)h(X) \equiv 1 \mod g(X),\end{aligned}$$

o que significa que $h(X)$ é o inverso multiplicativo de $(X - \gamma_i) \mod g(X)$. Desta forma, podemos simplificar a relação (4.7) para:

$$S_c \equiv \sum_{i=1}^{n} \frac{c_i}{X-\gamma_i} \equiv 0 \mod g(X). \tag{4.9}$$

4.2.1 Matriz de paridade de um código de Goppa

Vamos exprimir o polinómio quociente, $\frac{g(X)-g(\gamma_i)}{X-\gamma_i}$ da equação (4.7) em função de X.

Note-se que $g(X) - g(\gamma_i) = g_t X^t + g_{t-1}X^{t-1} + \ldots + g_1 X - \sum_{j=1}^{t} g_j \gamma_i^j$. O facto deste polinómio ser divisível por $(X-\gamma_i)$, a regra de Ruffini permite-nos encontrar o referido quociente. Com efeito,

	g_t	g_{t-1}	$\cdots$	g_2	g_1	$-\sum_{j=1}^{t} g_j\gamma_i^j$
γ_i		$g_t\gamma_i$			$\sum_{j=1}^{t-1} g_{j+1}\gamma_i^j$	$\sum_{j=1}^{t} g_j\gamma_i^j$
	g_t	$g_{t-1}+g_t\gamma_i$	$\cdots$	$\sum_{j=0}^{t-2} g_{j+2}\gamma_i^j$	$\sum_{j=0}^{t-1} g_{j+1}\gamma_i^j$	0

De onde,

$$\begin{aligned}\frac{g(X)-g(\gamma_i)}{X-\gamma_i} &= g_tX^{t-1}+(g_{t-1}+g_t\gamma_i)X^{t-2}+\ldots+(g_1+g_2\gamma_i+\ldots+g_t\gamma_i^{t-1})X^0\\ &= \sum_{k=0}^{t-1}X^k(\sum_{j=k+1}^{t}g_j\gamma_i^{j-1-k}).\end{aligned}$$

Desta forma, escreve-se:

$$\frac{g(X)-g(\gamma_i)}{X-\gamma_i}g(\gamma_i)^{-1} = g(\gamma_i)^{-1}\sum_{k=0}^{t-1}X^k(\sum_{j=k-1}^{t}g_j\gamma_i^{j-1-k}).$$

Mais ainda, combinando a equação (4.8) com o facto de que $c\in\mathcal{G}(L,g(X))$ se e só se $Hc^T=0$, resulta que a matriz de paridade, H, de um código de Goppa seja do tipo $t\times n$ em que a i-ésima coluna é:
$\begin{pmatrix} g_t \\ g_{t-1}+g_t\gamma_i \\ g_{t-2}+g_{t-1}\gamma_i+g_t\gamma_i^2 \\ \vdots \\ g_1+g_2\gamma_i+\ldots+g_t\gamma_i^t \end{pmatrix} g(\gamma_i)^{-1}$. Portanto,

$$\begin{aligned}H &= \begin{pmatrix} g_tg(\gamma_1)^{-1} & \cdots & g_tg(\gamma_{n-1})^{-1} \\ (g_{t-1}+g_t\gamma_1)g(\gamma_1)^{-1} & \cdots & (g_{t-1}+g_t\gamma_{n-1})g(\gamma_{n-1})^{-1} \\ \vdots & \ddots & \vdots \\ (g_1+g_2\gamma_1+\ldots+g_t\gamma_1^t)g(\gamma_1)^{-1} & \cdots & (g_1+g_2\gamma_{n-1}+\ldots+g_t\gamma_{n-1}^t)g(\gamma_{n-1})^{-1}\end{pmatrix}\\ &= \begin{pmatrix} g_t & 0 & 0 & \ldots & 0 \\ g_{t-1} & g_t & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \ldots & g_t\end{pmatrix}\begin{pmatrix} 1 & 1 & \ldots & 1 \\ \gamma_1 & \gamma_2 & \ldots & \gamma_n \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_1^{t-1} & \gamma_2^{t-1} & \ldots & \gamma_n^{t-1}\end{pmatrix}\begin{pmatrix} g(\gamma_1)^{-1} & 0 & \ldots & 0 \\ 0 & g(\gamma_2)^{-1} & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & g(\gamma_n)^{-1}\end{pmatrix}.\end{aligned}$$

Chamemos M à matriz mais a esquerda da última equação. Prova-se que, sendo M, $t\times t$, invertível e definida sobre $\mathbb{F}_{q^m}$ então, H e H', tal que $H=MH'$, geram o mesmo código. Este facto permite usar H' como a forma mais simples da matriz de paridade de um código de Goppa:

$$H' = \begin{pmatrix} g(\gamma_1)^{-1} & g(\gamma_2)^{-1} & \cdots & g(\gamma_n)^{-1} \\ \gamma_1g(\gamma_1)^{-1} & \gamma_2g(\gamma_2)^{-1} & \cdots & \gamma_ng(\gamma_n)^{-1} \\ \vdots & & \ddots & \\ \gamma_1^{t-1}g(\gamma_1)^{-1} & \gamma_2^{t-1}g(\gamma_2)^{-1} & \cdots & \gamma_n^{t-1}g(\gamma_n)^{-1}\end{pmatrix} \tag{4.10}$$

A matriz geradora do código de Goppa pode ser calculada à custa da relação estabelecida em (4.3).

Exemplo 4.24. *Dado o polinómio de Goppa* $g(X) = X^2 + X + 1$ *e* $L = GF(2^3)$ *definida no exemplo 3.4 . Vamos construir um código de Goppa de comprimento 8, sobre o corpo* $GF(8)$*, a partir de sua matriz de paridade.*

Para tal, vamos determinar a matriz de paridade conforme mostrado em (4.10)*.*

Assumindo que $GF(8)$ *está definido pelo polinómio irredutível* x^3+x+1 *e que* $\alpha = [x]$*, têm-se:*

$$\begin{array}{ll}
g(0) = 1 & \rightarrow g(0)^{-1} = 1 \\
g(1) = 1 & \rightarrow g(1)^{-1} = 1 \\
g(\alpha) = \alpha^2 + \alpha + 1 = \alpha^5 & \rightarrow g(\alpha)^{-1} = \alpha^2 \\
g(\alpha^2) = \alpha^4 + \alpha^2 + 1 = \alpha^3 & \rightarrow g(\alpha^2)^{-1} = \alpha^4 \\
g(\alpha^3) = \alpha^6 + \alpha^3 + 1 = \alpha^5 & \rightarrow g(\alpha^3)^{-1} = \alpha^2 \\
g(\alpha^4) = \alpha^8 + \alpha^4 + 1 = \alpha^6 & \rightarrow g(\alpha^4)^{-1} = \alpha \\
g(\alpha^5) = \alpha^{10} + \alpha^5 + 1 = \alpha^6 & \rightarrow g(\alpha^5)^{-1} = \alpha \\
g(\alpha^6) = \alpha^{12} + \alpha^6 + 1 = \alpha^3 & \rightarrow g(\alpha^6)^{-1} = \alpha^4
\end{array}$$

Assim,

$$H = \begin{pmatrix} 1 & 1 & \alpha^2 & \alpha^4 & \alpha^2 & \alpha & \alpha & \alpha^4 \\ 0 & 1 & \alpha^3 & \alpha^6 & \alpha^5 & \alpha^5 & \alpha^6 & \alpha^3 \end{pmatrix}.$$

Representando cada elemento na sua forma binária, teríamos:

$$H = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

As palavras de código são os elementos que constituem o conjunto solução do sistema linear $Hc^T = 0$*:*

$$\mathcal{G}(L, g(X)) = \left\{ \begin{array}{l} 00000000 \\ 00111111 \\ 11001011 \\ 11110100 \end{array} \right\}$$

4.2.2 Distância mínima de um código binário de Goppa

Proposição 4.25. *Seja $g(X)$ é um polinómio de Goppa irredutível, de grau t, com coeficientes em $GF(2^m)$ e $\mathcal{G}(L, g(X))$, o código binário de Goppa com os parâmetros $[n, k, d]$ então, $k \geq n - mt$ e $d \geq 2t + 1$.*

Seja $c \in \mathcal{G}(\mathrm{L}, g(X))$, chamemos τ_c o conjunto das coordenadas não nulas de c (note-se que $c_i = 1$ para todo $i \in \tau$). Então definamos o polinómio

$$\sigma_c(X) = \prod_{i \in \tau_c} c_i(X - \gamma_i). \tag{4.11}$$

A derivada de $\sigma_c(X)$ é:

$$\sigma_c'(X) = \sum_{i \in \tau_c} c_i \prod_{j \in \tau_c \setminus \{i\}} (X - \gamma_j).$$

Veja que, se multiplicarmos a equação (4.9) por $\sigma_c(X)$ obtemos,

$$\sigma_c(X) S_c(X) \equiv \sigma_c'(X) \mod g(X). \tag{4.12}$$

Como $g(\gamma_i) \neq 0$ para todo $0 \leq i \leq n - 1$, tem-se que $g(X)$ é primo com $\sigma_c(X)$ o que equivale dizer que $\sigma_c(X)$ é invertível módulo $g(X)$. Assim,

$$\frac{\sigma_c'(X)}{\sigma_c(X)} \equiv S_c(X) \mod g(X).$$

Logo,

$$\forall c \in GF(2)^n, c \in \mathcal{G}(L, g(X)) \Leftrightarrow \sigma_c'(X) \equiv 0 \mod g(X).$$

O facto de, em $GF(2^m)$, $\sigma_c'(X)$ ser um quadrado perfeito (observe-se que os coeficientes das potências ímpar de X são múltiplos de 2 que é característica do corpo) e $g^2(X)$ também, por $g(X)$ ser irredutível (e pelo teorema 2.5), implica que,

$$\forall c \in GF(2)^n, c \in \mathcal{G}(L, g(X)) \Leftrightarrow \sigma_c'(X) \equiv 0 \mod g^2(X).$$

Desta forma, para todo $c \in \mathcal{G}(\mathrm{L}, g(X)) \setminus \{0\}$ tem-se,

$$\begin{aligned} wt(c) = gr(\sigma_c(X)) &\geq gr(\sigma_c'(X)) + 1 \\ &\geq 2gr(g(X)) + 1 \\ &\geq 2t + 1. \end{aligned}$$

Logo, $d \geq 2t + 1$.

Estes factos implicam o seguinte:

Observação 4.26. *Códigos binários de Goppa gerados por $g(X)$ e por $g^2(X)$ são equivalentes.*

Corolário 4.27. *Um código binário de Goppa $\mathcal{G}(L, g(X))$, onde $g(X)$ é polinómio irredutível de grau t e $L = GF(2^m)$ tem a capacidade de corrigir pelo menos t erros.*

Demonstração. Resulta imediatamente da observação 4.15 □

4.2.3 Descodificar um código binário de Goppa

Existem vários algoritmos algébricos para a descodificação de um código de Goppa. Alguns deles são o de Patterson (ver [27]) e o de Berlekamp-Massey (ver [9]). Modificações interessantes destes podem se encontradas em bibliografias. Em [13] usa-se Patterson com modificação ao nível do algoritmo de Euclides. Em [18] o autor mostra como descodificar facilmente um código de Goppa clássico assumindo a sua representação no contexto das transformada de Fourier ou, se quisermos, polinómios de Mattson-Solomon[3].

O que vamos apresentar a seguir, é uma restrição ao caso binário com polinómio de Goppa irredutível e livre de quadrados conforme descrito em [18].

Seja $\mathcal{G}(\mathrm{L}, g(X))$, um código de Goppa onde $L = GF(2^m) = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\}$ e $g(X)$, o polinómio de Goppa de grau t.

Dado um vector $r = (r_0, r_1, \ldots, r_{n-1}) \in GF(2)^n$, o seu síndrome é $S = Hr^T$ onde H é a matriz de paridade de $\mathcal{G}$. Logo, $S = (S_0, \ldots S_{t-1})^T$ é um vector de comprimento t.

Consideremos, então, a sequência $S = (S_0, S_1, \ldots)$. Usando a matriz H na forma definida em 4.10, escreve-se que:

$$S_j = \sum_{i=0}^{n-1} \frac{r_i \gamma_i^j}{g(\gamma_i)} \tag{4.13}$$

Nota 4.28. *A variável γ_i assume elementos de L e indexam a posição das coordenadas de $r \in GF(2)^n$.*

[3]Isto é feito, por exemplo, em [20, pág. 347]

Como o síndrome só depende do vector erro, e, e assumindo que o peso de e é $\omega \leq t$, a relação (4.13) pode ser reescrita:

$$S_j = \sum_{i \in \tau_e} \frac{e_i \gamma_i^j}{g(\gamma_i)}, \text{ com } |\tau_e| = \omega \tag{4.14}$$

e onde τ_e é o conjunto das coordenadas não nulas de e.

O polinómio definido em (4.11), com relação ao vector erro e, é denominado **polinómio localizador de erro** e pode ser reescrito assim:

$$\sigma_e(X) = \prod_{i \in \tau_e} (X - \gamma_i) = \sum_{k=0}^{\omega} \sigma_k X^k \qquad \text{onde } \sigma_k \in GF(2^m) \text{ e } \sigma_\omega = 1. \tag{4.15}$$

Os zeros deste polinómio indicam a posição do erro ou seja,

$$\sigma_e(X) = 0 \Leftrightarrow X = \gamma_{i_1} \vee \gamma_{i_2} \vee \ldots \vee \gamma_{i_\omega}$$

significa que, r tem erro nas posições $r_{i_1}, r_{i_2}, \ldots, r_{i_\omega}$.

Existe uma relação importante entre os coeficientes S_j e σ_k que define um sistema de equações lineares, a qual se resume no seguinte teorema:

Teorema 4.29. *Para todo j, o coeficiente S_j verifica a relação de recorrência:*

$$\sum_{k=0}^{\omega} \sigma_k S_{j+k} = 0 \tag{4.16}$$

Demonstração. Vamos supor que r tem um erro na posição r_i. É equivalente dizer que γ_i é zero de $\sigma_e(X)$. Substituindo γ_i na equação (4.15), obtemos:

$$\sigma_0 + \sigma_1 \gamma_i + \sigma_2 \gamma_i^2 + \ldots + \sigma_{\omega-1} \gamma_i^{\omega-1} + \gamma_i^\omega = 0.$$

Multiplicando ambos os membros da equação anterior por $\gamma_i^j g(\gamma_i)^{-1}$ e pondo i a variar em τ_e, chega-se à equação (4.16):

$$\sigma_0 \sum_{i \in \tau_e} \frac{e_i \gamma_i^j}{g(\gamma_i)} + \sigma_1 \sum_{i \in \tau_e} \frac{e_i \gamma_i^{j+1}}{g(\gamma_i)} + \sigma_2 \sum_{i \in \tau_e} \frac{e_i \gamma_i^{j+2}}{g(\gamma_i)} + \ldots + \sigma_{\omega-1} \sum_{i \in \tau_e} \frac{e_i \gamma_i^{j+\omega-1}}{g(\gamma_i)} + \sum_{i \in \tau_e} \frac{e_i \gamma_i^{j+\omega}}{g(\gamma_i)} = 0.$$

Por último, aplicando a igualdade (4.14), obtemos a relação em prova que é equivalente ao seguinte sistema de equações lineares:

$$\begin{bmatrix} S_0 & S_1 & S_2 & \ldots & S_{\omega-1} \\ S_1 & S_2 & S_3 & \ldots & S_\omega \\ \vdots & & & & \\ S_{\omega-1} & S_\omega & S_{\omega+1} & \ldots & S_{2\omega-2} \end{bmatrix} \times \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_{\omega-1} \end{bmatrix} = \begin{bmatrix} S_\omega \\ S_{\omega+1} \\ \vdots \\ S_{2\omega-1} \end{bmatrix} \tag{4.17}$$

□

Estas equações podem ser usadas como passo base para a descodificação de códigos binários de Goppa com polinómio $g(X)$ de grau t (ver [18]). A solução pode ser encontrada aplicando o procedimento de Paterson-Gorenstein-Zierler como se segue:

Começa-se pondo $\omega = t$ e, tenta-se resolver a equação (4.17). Se não for encontrada uma solução, diminui-se ω de uma unidade e tenta-se de novo. O procedimento é repetido até que se encontre uma solução. A garantia de que este procedimento devolve uma solução é dada pelo lema seguinte:

Lema 4.30. *Seja $\mu \leq t$ e seja*

$$M_\mu = \begin{bmatrix} S_0 & S_1 & S_2 & \dots & S_{\mu-1} \\ S_1 & S_2 & S_3 & \dots & S_\mu \\ \vdots & & & & \\ S_{\mu-1} & S_\mu & S_{\mu+1} & \dots & S_{2\mu-2} \end{bmatrix}$$

. Então, M_μ é invertível se $\mu = \omega$ e não invertível se $\mu > \omega$, sendo ω o número de erros ocorrido.

Demonstração. Consideremos as matrizes:

$$A_\mu = \begin{bmatrix} 1 & 1 & \dots & 1 \\ \gamma_{(\tau_e)_1} & \gamma_{(\tau_e)_2} & \dots & \gamma_{(\tau_e)_\omega} \\ \vdots & & \ddots & \\ \gamma_{(\tau_e)_1}^{\omega-1} & \gamma_{(\tau_e)_2}^{\omega-1} & \dots & \gamma_{(\tau_e)_\omega}^{\omega-1} \end{bmatrix} \quad e,$$

$$B_\mu = \begin{bmatrix} \frac{e_{(\tau_e)_1}\gamma_{(\tau_e)_1}}{g(\gamma_{(\tau_e)_1})} & 0 & \dots & 0 \\ 0 & \frac{e_{(\tau_e)_2}\gamma_{(\tau_e)_2}}{g(\gamma_{(\tau_e)_2})} & \dots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \dots & \frac{e_{(\tau_e)_\omega}\gamma_{(\tau_e)_\omega}}{g(\gamma_{(\tau_e)_\omega})} \end{bmatrix}$$

Tendo em conta a definição de S_j, observe-se que $M_\mu = A_\mu \times B_\mu \times (A_\mu)^T$ e portanto, $det(M_\mu) = det(A_\mu)^2 \times det(B_\mu)$ pois $det(A_\mu) = det(A_\mu^T)$.

Se $\mu > \omega$ então, os valores adicionais, $e_{(\tau_e)_{\omega+1}}, e_{(\tau_e)_{\omega+2}} \dots e_{(\tau_e)_\mu}$ são nulos. Logo, a matriz M_μ terá determinante nulo, pois a matriz diagonal, B_μ, tem zeros na diagonal. Consequentemente, M_μ é não invertível.

Se $\mu = \omega$, então $det(B_\mu) \neq 0$ por B_μ ser diagonal com elementos diagonais não nulos.

Igualmente, $det(A_\mu) \neq 0$ porque A_μ é uma matriz de Vandermond com, $\gamma_{(\tau_e)_i}$, com $i = 1, \ldots, \omega$, todos diferentes. Logo, M_μ é invertível. □

A localização do erro é obtida como raízes do polinómio $\sigma_e(X)$.

Pela observação 4.26, todos os cálculos podem ser feitos usando $g^2(X)$ no lugar de $g(X)$. Convenciona-se que $0^0 = 1$.

Exemplo 4.31. *Vamos retomar o exemplo 4.24 para ilustrar o processo de descodificação.*

Suponhamos que na transmissão da palavra de código $c = (11110100)$ *foi introduzido o erro* $e = (00001010)$. *O receptor recebe* $r = c + e = (11111110)$ *e pretende recuperar a mensagem.*

Efectuamos então, os seguintes passos:

1. *Calcula-se os coeficientes* S_j *segundo* (4.14). *Os resultados são reduzidos módulo* $\alpha^3 + \alpha + 1$ *que fica facilitado se recorrermos à tabela 3.2:*

$$S_j = \sum_{i \in \tau_r} \frac{\gamma_i^j}{g^2(\gamma_i)} = 0^j + 1^j + \alpha^{j+4} + \alpha^{2j+1} + \alpha^{3j+4} + \alpha^{4j+2} + \alpha^{5j+2}$$

   Assumindo que ocorreu dois erros, pomos $\omega = 2$. *Assim, é preciso calcular* S_0, S_1, S_2 *e* S_3*:*

$$\begin{aligned} S_0 &= 1 + 1 + \alpha^4 + \alpha + \alpha^4 + \alpha^2 + \alpha^2 = \alpha \\ S_1 &= 0^1 + 1^1 + \alpha^5 + \alpha^3 + \alpha^7 + \alpha^6 + \alpha^7 = 0 \\ S_2 &= 0^2 + 1^2 + \alpha^6 + \alpha^5 + \alpha^{10} + \alpha^{10} + \alpha^{12} = \alpha^2 \\ S_3 &= 0^3 + 1^3 + \alpha^7 + \alpha^7 + \alpha^{13} + \alpha^{14} + \alpha^{17} = \alpha^4 \end{aligned}$$

2. *Resolve-se o sistema de equações:*

$$\begin{bmatrix} \alpha & 0 \\ 0 & \alpha^2 \end{bmatrix} \times \begin{bmatrix} \sigma_0 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} \alpha^2 \\ \alpha^4 \end{bmatrix} \Rightarrow \sigma_0 = \alpha \qquad e \qquad \sigma_1 = \alpha^2$$

3. *Resolve-se a equação* $\sigma(X) = X^2 + \alpha^2 X + \alpha = 0$, *de onde se tem* $X = \alpha^3$ *ou* $X = \alpha^5$ *como soluções correspondendo, respectivamente, aos elementos de* L, γ_4 *e* γ_6. *Conclui-se que ocorreram dois erros nas posições* r_4 *e* r_6.

Capítulo 5

Sistema Criptográfico de McEliece

5.1 O sistema criptográfico de McEliece e suas variantes

Em [21], McEliece propõe um sistema criptográfico baseado em códigos. Para tal, usa o facto de existir um algoritmo rápido para descodificar um código de Goppa e o facto de o mesmo não existir para os códigos lineares em geral. Assim, constrói um sistema cuja chave parece ser um código linear aleatório mas que "esconde", por trás, o código de Goppa. Este sistema criptográfico, como todos os sistemas criptográficos de chave pública, consiste em três algoritmos. Um algoritmo para gerar chaves, o qual produz o par de chaves (pública e privada); um algoritmo para cifrar e um algoritmo para decifrar. Uma das componentes da chave pública é uma matriz "disfarce" da matriz geradora, G, de um código binário de Goppa de comprimento n e dimensão k, capaz de corrigir até t erros.

Apesar de possuir algoritmos eficientes tanto para cifrar como para decifrar, o sistema é muito pouco utilizado devido, sobretudo, ao tamanho das chaves que gera. Muitas variantes deste sistema foram propostas com o objectivo de diminuir o tamanho das chaves. De entre estas propostas estão o sistema criptográfico de Niederreiter baseado em códigos Reed-Solomon, quebrado por Sidenikov e Shestakov [33]; o sistema criptográfico de Sidelnikov baseado em códigos de Reed-Muller quebrado por Lorenz Minder e Amin Shokrollahi [25]; e o sistema criptográfico GPT, baseado em códigos de Ga-

bidulin, quebrado pelo ataque de Gibson [11] ataque este extendido por R. Overbeck [26].

Quase todas as variantes do sistema de McEliece são marcadas pela substituição do código secreto de Goppa por outra família de códigos, variantes estas que se revelaram inseguras. Actualmente, somente as variantes cujos códigos secretos são os códigos de Goppa permanecem seguras. A variante proposta por Niederreiter usando códigos de Goppa, oferece um nível de segurança equivalente ao sistema de McEliece (ver [35]) e, além disso, permite a construção de assinaturas digitais (ver [5]).

5.2 Descrição do sistema de McEliece

Para cada polinómio irredutível de grau t sobre um corpo finito $GF(2^m)$, tem-se um código binário de Goppa irredutível de comprimento $n = 2^m$ e dimensão $k \geq n - tm$, capaz de corrigir, pelo menos, t erros. Este facto é obtido por dedução directa da própria definição de código binário de Goppa 4.23 e da proposição 4.25.

O sistema criptográfico de McEliece considera n e t inteiros positivos e $g(X)$, um polinómio aleatório irredutível de grau t sobre $GF(2^m)$. Em seguida, o sistema produz, via as equações (4.10) e (4.3), uma matriz geradora G, do tipo $k \times n$, para o código. O sistema disfarça a matriz G seleccionando, aleatoriamente, uma matriz S, de ordem k, invertível e uma matriz de permutação P, de ordem n, e calcula a matriz $G' = SGP$ a qual gera um código linear. A matriz G' é tomada como uma componente da chave pública.

5.2.1 Funcionamento do sistema de McEliece

Suponhamos que a mensagem a ser cifrada pudesse ser dividida em blocos de k bits e seja u um dos blocos. Será, então, enviado um vector $x = uG' + z$ onde G' é a matriz geradora pública e z é um vector de comprimento n e peso t, localmente gerado pelo emissor.

Recebendo x, o sistema pode recuperar de forma eficiente a mensagem u procedendo

de seguinte forma:

Calcula $x' = xP^{-1}$, onde P^{-1} é a inversa da matriz de permutação P. Assim,

$$x' = xP^{-1} = (uG' + z)P^{-1} = (uSGP + z)P^{-1} = uSG + zP^{-1}.$$

Note-se que P^{-1} é também uma matriz de permutação, logo, $z' = zP^{-1}$ é um vector de mesmo peso que z. Usando uma tabela de síndrome ou procedendo como descrito na subsecção 4.2.3 o sistema obtém z'. Adicionando este valor a x obtém a palavra de código $u' = uSG$. Se G estiver na forma canónica, os primeiros k bits de u' serão os da mensagem sob o efeito da matriz S, pelo que restará multiplicar, à direita, por S^{-1} par obter u. Caso contrário, define-se uma matriz, G_i, do tipo $n \times k$ tal que $G \times G_i = I$, onde I é a matriz identidade. G_i pode ser determinada da seguinte forma (ver [13]):
O sistema selecciona, aleatoriamente, k colunas de G de forma a ter uma matriz, G_0, de ordem k, invertível e determina a sua inversa, G_0^{-1}. As linhas de G_0^{-1}, são inseridas nas posições em que foram retiradas as k colunas de G, formando a matriz, G_i, do tipo $n \times k$, sendo as restantes posições preenchidas com zeros. Assim, tendo a matriz G_i, o sistema multiplica $u' = uSG$ sucessivamente, à direita, por G_i e S^{-1}, obtendo a mensagem original u.

Exemplo 5.1. *A fim de facilitar a compreensão, vamos ilustrar o procedimento descrito anteriormente utilizando a matriz geradora (na forma canónica) do código* - $[7, 4, 3]$ *de Hamming usada no exemplo 4.4, em vez de uma matriz geradora de um código de Goppa. Para descodificar a mensagem usaremos uma tabela de síndrome.*

Então, seja a matriz geradora

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Consideremos a matriz invertível S e a matriz de permutação P dados a seguir:

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

e

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

O receptor Bob terá, como uma componente de chave pública, a matriz

$$G' = SGP = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Se o emissor Alice, quiser enviar a mensagem $u = (1101)$ *a Bob, procede de seguinte forma:*

1. *constrói um vector erro,* z*, de peso 1. Seja este* $z = (0000100)$.

2. *Envia* $x = uG' + z = (0110010) + (0000100) = (0110110)$

Quando Bob recebe $x = (0110110)$*, ele procede de seguinte forma para descodificar:*

1. *Calcula* $x' = xP^{-1} = (0110110) \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} = (1000111)$

2. *Calcula o síndrome de x':*

$$S_{x'} = x'H^T = (1000111)\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (001),$$

onde H é a matriz de paridade do código gerado por G, tal que $GH^T = 0$.

3. *Consulta a tabela de coset leader (neste caso, todos os possíveis vectores de peso 1 com os respectivos síndromes - ver tabela 5.1) para encontrar, na posição correspondente, o padrão de erro: O padrão correspondente é (0000001), o qual deve*

Padrão de erro	Síndrome
0000000	000
1000000	110
0100000	101
0010000	011
0001000	111
0000100	100
0000010	010
0000001	001

Tabela 5.1: Coset leader

ser somado a x', resultando em $x'' = 1000110$.

4. *Multiplica x'', sucessivamente, por G_i e S^{-1}.*

Nota 5.2. *Como a matriz G está na forma canónica, o vector uS corresponde aos quatro primeiros dígitos de x'', pelo que não é necessário a multiplicação por G_i. Um caso no qual é necessário o cálculo de G_i, é apresentado no exemplo a seguir.*

5. *Finalmente, calcula*

$$uSS^{-1} = (1000)S^{-1} = (1000)\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} = (1101),$$

que é a mensagem original.

Exemplo 5.3. *Neste exemplo vamos considerar uma matriz geradora, G, não na forma canónica a fim de ilustrarmos o processo de construção da matriz G_i tal que $GG_i = I$, descrito anteriormente.*

Vamos supor, então, que a matriz geradora é

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Consideremos a matriz invertível S e a matriz de permutação P dados a seguir:

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

e

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Como a matriz G não está na forma canónica, convém calcular e armazenar G_i tal que $GG_i = I$: Se tomarmos 4 colunas de G na seguinte ordem de índice 6^a, 4^a, 1^a, 7^a, obtemos

$$G_0 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix},$$

invertível, com

$$G_0^{-1} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Inserindo as linhas de G_0^{-1}*, respectivamente, como* $6^a, 4^a, 1^a$ *e* 7^a *linha de uma matriz do tipo* 7×4 *e preenchendo as restantes linhas com zeros, obtemos*

$$G_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Uma componente de chave pública do receptor Bob, será a matriz

$$G' = SGP = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Se a Alice quiser enviar a mensagem $u = (1101)$ *a Bob, procede de seguinte forma:*

1. *constrói um vector erro,* z*, de peso 1. Seja este* $z = (0000100)$*.*

2. *Envia* $x = uG' + z = (0111000) + (0000100) = (0111100)$

Quando Bob recebe $x = (0111100)$*, ele procede de seguinte forma para descodificar:*

1. *Calcula* $x' = xP^{-1} = (0111100)\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} = (1100101)$

2. *Calcula o síndrome de* x'*:*

$$S_{x'} = x'H^T = (1100101)\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (001).$$

Nota 5.4. *A matriz* H *pode ser obtida a partir da relação* $GH^T = 0$*.*

3. *Consulta a tabela de coset leader, 5.1, para encontrar, na posição correspondente, o padrão de erro:*

   O padrão correspondente é (0000001). Deve ser somado a $x' = uSG + zP^{-1}$*, resultando em* $x'' = 1100100 = uSG$*.*

4. *Multiplica* x''*, sucessivamente, por* G_i *e* S^{-1}*, obtém-se* u*. Com efeito,*

$$(1100100)\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = 1000 = uS$$

e, calculando

$$uSS^{-1} = (1000)S^{-1} = (1000)\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} = (1101),$$

que é a mensagem original.

O exemplo seguinte utiliza um código binário de Goppa tal como descrito por McEliece e o processo de correcção de erros descrito em 4.2.3. Este exemplo é construído com recurso a uma implementação do sistema criptográfico de McEliece feito no Maple e cujos procedimentos apresentamos no apêndice A.1. O programa recebe a mensagem (em texto), converte-a em binário (8 bits) usando a tabela ASCII (0-255). As etapas do processo dão-se com a mensagem e a matriz geradora na forma binária. No processo de decifragem, especificamente, na etapa de correcção de erros, é usada a matriz de paridade com elementos definidos sobre o corpo finito $GF(2^5)$. Devido ao tamanho desta matriz, apresentamo-la no apêndice A.2 em vez daqui, no texto.

Exemplo 5.5. *Seja o corpo $GF(2^5)$ gerado por $f \in GF(2)[x]$ tal que $f(x) = x^5 + x^4 + x^3 + x + 1$ e $f(\alpha) = 0$, sendo α um elemento primitivo. Vamos construir um código binário de Goppa com suporte $L = GF(2^5) = \{0, 1, \alpha, \alpha^2, \ldots \alpha^{29}, \alpha^{30}\}$ e polinómio gerador, $g(x) = x^4 + x^3 + 1$, irredutível em $GF(2)$ e sem zeros em L. Este código, $\mathcal{G}(L, g)$ - $[32, 12, 9]$, permite corrigir até 4 erros.*

A matriz de paridade, H, calculada conforme a equação (4.10) e a matriz geradora, G, calculada a partir da relação $GH^T = 0$ (equivalente a (4.3)) estão exibidas a seguir:

$H =$

$$\begin{bmatrix}
1&1&0&0&0&1&0&1&1&1&0&1&1&1&1&0&0&1&0&1&1&1&0&0&1&1&0&0&0&1&0&1\\
0&0&1&1&1&1&0&1&1&1&1&0&1&1&0&1&0&0&1&0&0&1&1&0&1&0&0&0&1&0&0&0\\
0&0&1&1&1&0&1&0&0&1&0&1&0&1&1&0&0&1&1&0&1&0&0&0&0&1&1&1&0&0&1&1\\
0&0&1&0&0&1&1&1&0&0&1&1&1&0&1&1&1&0&1&1&1&0&0&1&1&0&1&1&0&1&1&1\\
0&0&1&1&1&1&0&1&1&0&0&1&0&0&0&0&0&1&1&0&1&1&1&0&0&1&0&1&1&0&1&0\\
0&1&1&1&1&0&1&0&1&0&1&0&0&1&1&1&0&1&1&1&0&0&0&1&0&1&0&0&1&1&1&1\\
0&0&1&0&0&1&0&1&1&1&1&1&0&0&1&1&1&1&0&1&1&1&1&1&1&1&1&1&0&0&1&1\\
0&0&1&1&0&1&1&1&0&1&0&1&1&0&1&1&1&0&1&1&1&1&1&1&1&0&1&0&1&1&1&0\\
0&0&0&0&0&1&0&1&1&0&1&1&0&1&0&0&0&1&0&0&0&0&0&1&1&0&1&1&0&0&0&1\\
0&0&0&1&1&1&1&0&1&1&1&1&1&1&1&0&1&1&0&0&1&1&0&1&1&0&1&1&1&1&0&1\\
0&1&0&1&0&1&1&1&0&1&0&1&1&0&0&0&0&1&0&0&1&0&0&0&0&1&0&0&0&1&1&0\\
0&0&1&0&1&1&1&1&0&0&1&1&1&0&1&1&0&0&0&0&0&1&0&0&1&0&0&0&0&1&1&0\\
0&0&1&0&0&0&1&0&1&1&1&1&1&0&0&0&1&0&1&1&1&0&0&0&0&1&0&1&0&0&1&0\\
0&0&1&1&0&0&1&1&1&0&0&0&1&1&0&0&0&0&1&0&1&1&0&1&0&1&1&0&1&1&1&0\\
0&0&0&1&1&0&1&0&1&0&0&1&0&0&0&1&0&1&1&1&0&1&1&1&0&0&0&0&1&0&0&1\\
0&1&0&0&0&1&1&1&1&0&1&1&1&0&0&0&1&1&1&0&1&0&1&0&0&1&0&0&0&0&1&0\\
0&0&0&1&1&1&0&0&1&0&0&1&1&0&1&1&1&0&1&0&1&1&1&1&1&0&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&1&0&0&0&1&1&1&0&0&1&0&1&1&0&1&0&0&1&1&1&1&1&1&0\\
0&0&1&0&1&1&1&1&0&1&1&0&0&1&0&1&1&1&0&1&1&1&1&1&0&1&0&0&1&0&1&1\\
0&0&1&1&0&0&1&0&1&0&0&1&1&0&0&1&0&0&1&0&1&0&1&1&1&1&1&1&1&0&0&0
\end{bmatrix},$$

$G =$

$$\begin{bmatrix}
1&1&0&0&1&1&1&1&0&1&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&1&0&0&1&1&0&1&1&0&1&0&0&1&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&1&1&0&1&0&1&1&1&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&1&1&1&1&0&1&0&1&1&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&1&1&1&0&1&1&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&1&0&1&1&0&1&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0\\
0&1&0&1&0&1&0&1&0&1&0&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&1&0&0&0&0&0\\
1&0&0&1&0&1&1&1&0&0&0&1&0&1&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&1&0&0&1&1&0&1&1&1&0&0&1&0&1&1&0&1&0&1&0&0&0&0&0&0&0&0&1&0&0&0\\
1&0&1&0&1&1&1&0&1&1&1&1&1&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&1&1&1&1&0&0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0\\
0&1&0&0&0&1&1&0&0&1&0&1&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}.$$

Geramos, em seguida, aleatoriamente, uma matriz de permutação, P, de ordem 32 e matrizes de ordem 12 até obtermos uma matriz S, invertível, completando a chave secreta, $\{G, S, P, g\}$.

$$S = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1
\end{bmatrix}$$

$P =$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}.$$

Agora, calculamos a matriz geradora pública, $G' = SGP$:

$G' =$

$$\begin{bmatrix}
0&1&0&1&0&0&1&1&0&1&1&1&0&1&1&0&0&1&1&1&0&1&0&1&1&1&1&0&0&1&0&0\\
1&1&1&0&1&1&0&0&0&1&1&1&0&0&1&0&1&0&0&1&0&0&0&0&1&1&0&1&0&0&1&0\\
1&0&0&0&1&1&1&0&1&0&1&1&0&1&0&0&1&1&1&1&0&1&0&1&1&1&0&0&1&1&0&0\\
1&1&1&0&0&0&1&0&1&1&0&1&1&1&0&1&1&0&0&1&0&1&0&1&1&1&1&1&1&1&1&0\\
0&0&0&0&0&1&1&1&1&0&1&0&0&1&1&1&1&0&1&0&1&0&0&0&0&1&0&1&0&0&1&1\\
0&0&0&1&0&0&0&1&0&0&1&0&1&0&0&1&1&1&1&0&0&1&1&0&1&0&1&1&0&1&1&0\\
1&0&0&1&0&1&0&1&0&1&0&0&0&1&0&1&0&0&0&1&1&1&1&1&1&1&1&0&1&1&1&1\\
0&1&0&1&1&0&1&0&0&1&1&1&1&0&1&1&0&1&0&0&1&1&0&0&1&0&1&0&1&1&0&1\\
0&1&1&1&0&1&1&1&1&1&1&0&1&1&1&0&0&1&0&0&0&0&0&1&1&1&0&1&0&0&0&1\\
1&1&1&1&1&0&0&1&0&1&1&1&1&1&1&1&1&1&0&0&1&0&1&1&0&1&0&1&1&0&0&0&0\\
1&0&1&1&1&0&0&1&0&0&1&1&0&0&1&1&0&1&1&0&0&0&0&1&0&1&1&0&1&1&1&0\\
0&1&0&1&0&0&1&1&1&0&0&1&0&1&1&1&1&1&1&1&0&0&1&0&1&0&0&1&0&1&0&1&0
\end{bmatrix}.$$

A chave pública é constituída pelo par $\{G', t = 4\}$. Cada entidade cria a sua chave pública e a correspondente secreta. Querendo, a Alice, enviar-nos uma mensagem, "Olá Ilídio, tudo bem?", correspondendo ao string binário (usando o código ASCII):

$$01001111011011001110000100100000010010010110110011101101011001000110100$$
$$10110111100101100001000000111010001110101011001000110111100100000000110001$$
$$001100101010110110100111111$$

ela requisita a nossa chave pública, calcula e envia-nos $c = uG' + e$, para cada bloco u de $k = 12$ bits. O vector e, de 32 bits e de peso não superior a 4, é gerado por Alice. Seja $e = 01100100010000000000000000000000$, então, a mensagem que recebemos é a concatenação dos blocos c_i de 32 bits cada (neste caso são 14 blocos):

$$00100111100000100011000010110001111100111000011010000100000011100111000$$
$$10011000100011110111010111111100011010000110100100010110000110111001111 1$$
$$00000110101010100100100010011010000001101011000010011001011110011001011 1$$
$$00101101000001100100011011110111110111011101000001101111001110101110011 0$$
$$10011011110111111110101011010011100110010111100110010111001011010110110 0$$
$$11000011010010111011001010101011100010000011010111001100100110100110110 0$$
$$10110000000101110$$

Observação 5.6. *O vector e é aleatório e gerado para cada bloco a transmitir. A fim de facilitar o exemplo, mantivémo-lo igual para todos os blocos.*

Para decifrarmos a mensagem, a cifra é dividida em blocos de comprimento 32. A decifrágem é feita bloco a bloco, usando o procedimento *DECIFRARr()* (ver apêndice A.1). Cada bloco, escrito na forma de matriz linha, é multiplicado pela matriz de permutação P. Em seguida, o algoritmo de correcção de erro, através do procedimento *Localiza-Erro()*, localiza e corrige os erros devolvendo-nos um vector, ainda, de comprimento n, contendo os 12 bits da mensagem sob a acção das matrizes S e G, cumulativamente. Multiplicando, à direita, sucessivamente, por G_i e S obtemos o bloco com os k bits da mensagem, u. Concatenando todos os blocos, obtemos a mensagem decifrada na forma binária. Finalmente, convertemos a mensagem binária para texto com auxilio do procedimento *BinToTexto()*.

Nota 5.7. *A matriz geradora, G, é obtida, aqui, como espaço nulo da matriz de paridade, H. Ela tem, pelo menos k colunas de peso 1 que formam, se agrupadas convenientemente, uma matriz identidade de ordem k. Os índices destas colunas, com respeito à matriz G, indicam a posição em que se encontram os k bits da mensagem em cada bloco uSG. Sendo assim, em vez de implementarmos a matriz G_i, usado no exemplo anterior, usamos o procedimento* KbitsDams() *para extrair estes k bits.*

5.3 Variante de Niederreiter

O sistema criptográfico de Niederreiter foi proposto por H. Niederreiter em 1986. O sistema originalmente proposto, cujo código secreto é o de Reed-Solomon, foi quebrado. No entanto, o sistema é seguro se for usado um código de Goppa como código secreto. A diferença em relação ao sistema de McEliece, reside na estrutura. No lugar da matriz geradora, G, é usada a matriz de paridade, H.

Dado um código binário de Goppa, $\mathcal{G}(L, g(X))$, toma-se uma matriz de paridade, H, do tipo $(n-k) \times n$ sobre $GF(2)$, gera-se, aleatoriamente, uma matriz invertível, S, de ordem $(n-k)$ sobre $GF(2)$ e uma matriz de permutação, P, de ordem n. Calcula-se a matriz $H' = SHP$. Publica-se o par (H', t), onde t é o grau de g, e mantém-se secreto os valores (S, H, P, g).

Se Alice quer enviar uma mensagem, m, a Bob, ela procede da seguinte forma:

1. Calcula $c = H'm'^T$;

Nota 5.8. *A mensagem $m' \in GF(2)^n$ e tem peso t. É resultado de uma pre-codificação feita por meio de uma função bijectiva, denominado* ***codificador a peso constante*** *(ver [31]) tal que:*

$$\begin{aligned} \boldsymbol{Cod} \ : \ & GF(2)^n \to GF(2)^n \\ & m \mapsto \{m' \mid wt(m') = t\} \end{aligned}$$

2. Envia $c \in GF(2)^{n-k}$.

Quando Bob recebe c, ele decifra a mensagem da seguinte forma:

1. Calcula

$$\begin{aligned} z &= S^{-1}c \\ &= S^{-1}SHPm'^T \\ &= HPm'^T; \end{aligned}$$

2. Aplicando o algoritmo da descodificação por síndrome a HPm'^T, obtém Pm'^T;
3. Calcula $m' = (P^{-1}Pm'^T)^T$;
4. Finalmente, calcula $m = Cod^{-1}(m')$.

Capítulo 6

Ataques ao sistema criptográfico de McEliece

Pretendemos aqui, descrever alguns, possíveis, ataques ao sistema criptográfico de McEliece, tendo em conta os parâmetros, originalmente, propostos por McEliece, isto é, $n = 1024$, $t = 50$ e $k = 524$.

Já aquando da proposta do sistema, McEliece [21] identifica duas classes de possíveis ataques. A classe dos ataques estruturais, que eventualmente, conduz ao desvendar da chave secreta, G, S, P e o polinómio $g(X)$ e a classe dos ataques não estruturais, que permite descobrir a mensagem, u, directamente da mensagem cifrada, r, sem o conhecimento de G.

6.1 Ataques não estruturais

Estes ataques têm sempre como alvo os textos cifrados a partir dos quais procuram descobrir o texto claro ou a mensagem. Com estes ataques, mesmo em caso de sucesso, o sistema permanece intacto. A seguir descreveremos alguns deste ataques.

1. **Comparação exaustiva das palavras de código** [21]. Pode-se comparar cada vector recebido, $r = uG' + e$, com as 2^k palavras de código geradas por G'. Seja c a palavra resultado da comparação. Esta deve estar a uma distância $\leq t$ de r

e, será única, pela distância mínima. Segue então, que $c = uG'$. A solução deste sistema devolve-nos u a partir de c. No entanto, encontrar c pode significar 10^{158} comparações.

2. **Descodificação por síndrome** [21]. Uma aproximação da força bruta baseada no lideres dos cosets, tem uma carga de trabalho na ordem de 2^{500}.

3. **Descobrindo os k bits da mensagem** [21]. Selecciona-se, aleatoriamente, k posições e assume-se que não haja erro nessas posições. Se a restrição da matriz G' a essas k posições continuar tendo característica k, então pode-se encontrar um candidato, u', para o vector de informação u através da eliminação de Gauss. Se a característica for $\leq k$, o processo de eliminação de Gauss devolve-nos algumas possibilidades para u' ou o sistema não terá solução. Para cada possível candidato, u', calcula-se $u'G'$ e, verifica-se a que distância está do vector recebido, r. Se $d \leq t$ então, $u = u'$. A probabilidade de os k bits estarem isentos de erro é de $(1 - \frac{t}{n})^k$. Este processo traduz numa carga de trabalho na ordem de 10^{19}.

   Não obstante isso, segundo McEliece, este é o mais promissor dos ataques. Actualmente, muitas versões deste ataque foram introduzidas. Recentemente, Bernstein et. all [4], propõe um melhoramento do ataque de Stern (ver [34]) e mostram que o sistema pode ser "quebrado" (no sentido de extrair o texto claro a partir do cifrado, pois, não é um ataque estrutural) em uma semana, se o programa for corrido em um cluster de 200 CPU Intel Core 2 Quad Q6600 de 2.4 GHz. Se for em, apenas, um computador com as mesmas características, o tempo médio esperado para completar o ataque seria de 1400 dias.

   Recentemente D. Bernstein (ver [3, pág. 73-80]) mostra que a versão quântica deste ataque é muito mais rápida e, recomenda quadruplicar os parâmetro de McEliece.

A complexidade destes ataques depende, essencialmente, dos parâmetros do sistema pelo que pode ser aumentada alterando os parâmetros. Actualmente, utilizadores do sistema de McEliece usam parâmetros maiores tal como $n = 4096$, $k = 3556$ (ver [3]).

Outras formas de ataques são ocasionadas pelo tipo de chaves e forma como são utilizadas. A saber:

Múltiplas cifragens da mesma mensagem e mensagens relacionadas. Não é seguro permitir relações lineares entre duas ou mais cifras. Em particular, cifrar a

mesma mensagem duas ou mais vezes com a mesma chave, permite ao atacante obter a mensagem original por simples comparação destas cifras. Este ataque não é aplicável à versão de Niederreiter (ver [10]).

6.2 Ataques estruturais

McEliece já levantara essa hipótese:**e se alguém descobre** S **e** P - Ataque estrutural. O criptanalista pode tentar descobrir a matriz S e a permutação P para depois calcular $G = S^{-1}G'P^{-1}$. Tendo G, não será difícil encontrar o polinómio de Goppa, $g(X)$, que define o código que tenha G como matriz geradora. E, desta forma encontrar a mensagem u. Todavia, o número de possíveis matrizes de ordem k, invertíveis, e de matrizes de permutações de ordem n, é muito grande o que torna impraticável tal tentativa.

Uso de chaves fracas. Piere Loindreu e Nicolas Sendrier, em [19], mostram que os códigos de Goppa com polinómio gerador de coeficientes sobre $GF(2)$ constituem uma família de chaves fracas para o sistema criptográfico de McEliece e podem induzir ataques estruturais ao sistema. Os mesmos mostram que é possível detectar quando um sistema de McEliece usa esta classe de chaves fracas e propõem, nesta situação, um ataque contra o sistema que, para os parâmetros proposto por McEliece, pode ser completado, mesmo que num longo tempo computacional - 500 Anos. Este tempo é considerado, por eles, alcançável uma vez que o algoritmo permite uma distribuição massiva.

As chaves fracas para o sistema criptográfico de McEliece são os códigos de Goppa tendo o polinómio gerador coeficientes sobre o corpo $GF(2)$. Primeiramente, procede-se à detecção do uso de chaves fracas pelo sistema. Em seguida, busca-se um polinómio $g(X)$ de grau t, de entre todos os polinómios de grau t mónicos e irredutíveis sobre $GF(2)$, tal que o código $\mathcal{G}(L, g(X))$ seja equivalente ao código público C - o que tem $G' = SGP$ como matriz geradora. A última etapa do ataque é a recuperação de uma permutação, π, entre os dois códigos, público e secreto que, por definição, são equivalentes. É usado o sub-código idempotente projectado como um espaço de busca mais pequeno tornando mais rápido o algoritmo e o **algoritmo de separação do**

suporte ($\mathcal{SSA}$[1]) para encontrar permutações entre dois códigos equivalentes [30].

6.2.1 Detecção do uso de chave fraca

Seja $\mathcal{G}(L, g(X))$, um código de Goppa onde g é o polinómio gerador binário de grau t e $L = GF(2^m) = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\}$ o suporte do código.

Por aplicação do $\mathcal{SSA}$ a um código de Goppa com polinómio gerador binário, detecta-se o seu grupo de automorfismos não trivial (ver [19]). $\mathcal{SSA}$ é tomado, aqui, como uma "caixa preta" que recebe a matriz geradora de um código linear como argumento e devolve uma partição etiquetada $\mathcal{P} = \{(\mathcal{P}_i, E)\}_{i \in E}$, onde E é o conjunto das etiquetas. $\mathcal{P}_i \neq 0$ chamam-se células de $\mathcal{P}$ e formam uma partição de L.

Quando um sistema de McEliece usa um polinómio de Goppa binário, o $\mathcal{SSA}$ aplicado sobre G' produz uma partição $\mathcal{P}$ cuja cardinalidade das células é, exactamente, a das "classes de conjugados" de $GF(2^m)$ sobre $GF(2)$, definida em 3.11. Desta forma, se sabe que o sistema usa uma chave fraca, e pode ser descoberta.

6.2.2 Encontrar o polinómio $g(X)$

Nesta subsecção, vamos fazer uma descrição teórica de como, no contexto de uso de chave fraca, descobrir o polinómio, $g(X)$, que define o código secreto de Goppa. Para tal, o conceito seguinte é de fundamental importância.

Definição 6.1 (Sub-código idempotente)**.** *Uma palavra $c \in \mathcal{G}(L, g(X))$ é dita um idempotente se qualquer uma das seguintes afirmações equivalentes for satisfeita:*

a) *$\rho(c) = c$ onde ρ pertence ao grupo de automorfismos de Frobenius (ver definição 3.3);*

b) *O suporte de c é a união das classes de conjugados de $GF(2^m)$.*

O conjunto de todos os idempotentes de $\mathcal{G}(L, g(X))$ é um sub-código linear de $\mathcal{G}(L, g(X))$ denotado por $I(L, g(X))$ e denominado ***sub-código idempotente*** *de $\mathcal{G}(L, g(X))$.*

[1]Sigla proveniente do inglês - Support Splitting Algorithm

É possível, a partir de $I(L, g(X))$, construir um outro código com a mesma dimensão de $I(L, g(X))$ mas, de menor comprimento e cuja matriz de paridade pode ser definida em função de g. Com efeito, uma vez que as coordenadas de c rotuladas pelos conjugados do mesmo elemento de $GF(2^m)$ são idênticas, considera-se o conjunto $R \subset L$ constituído só pelos representantes das classes de conjugados. Com $N = |R|$, o número de classes de conjugados de $GF(2^m)$, define-se a seguinte projecção:

$$\begin{array}{rcccc} \kappa_R & : & GF(2)^n & \rightarrow & GF(2)^N \\ & & (c_i)_{\gamma_i \in L} & \mapsto & (c_i)_{\gamma_i \in R} \end{array} \tag{6.1}$$

Aplicando κ_R a cada elemento de $I(L, g(X))$, obtém-se um código linear

$$I(R, g(X)) = \{\kappa_R(c) \mid c \in I(L, g(X))\} \tag{6.2}$$

denominado **sub-código idempotente de $\mathcal{G}(L, g(X))$ projectado**.

Matriz de paridade de $I(R, g(X))$

Para todo elemento $\gamma \in GF(2^m)$, $GF(2)[\gamma]$ é o mais pequeno corpo contendo γ. Denotamos por L_γ, a classe de conjugado de γ e por Tr_γ, o operador traço de $GF(2)[\gamma]$ sobre $GF(2)$ definido assim:

$$\begin{array}{rcccc} Tr_\gamma & : & GF(2)[\gamma] & \rightarrow & GF(2) \\ & & \alpha & \mapsto & \sum_{i=0}^{|L_\gamma|-1} \alpha^{2^i}. \end{array}$$

Proposição 6.2. *Seja $g(X)$ um polinómio de grau t sobre $GF(2)$. A matriz binária*

$$H_I = \begin{bmatrix} Tr_{\gamma_1}(g^{-1}(\gamma_1)) & \dots & Tr_{\gamma_N}(g^{-1}(\gamma_N)) \\ Tr_{\gamma_1}(\gamma_1 g^{-1}(\gamma_1)) & \dots & Tr_{\gamma_N}(\gamma_N g^{-1}(\gamma_N)) \\ \vdots & \ddots & \vdots \\ Tr_{\gamma_1}(\gamma_1^{t-1} g^{-1}(\gamma_1)) & \dots & Tr_{\gamma_N}(\gamma_N^{t-1} g^{-1}(\gamma_N)) \end{bmatrix}$$

é a matriz de paridade do $I(R, g(X))$.

A busca da chave $g(X)$ passa a ser feita sobre o sub-código idempotente projectado, $I(R, g(X))$, com auxilio de um código equivalente derivado a partir do código público.

Seja C um código binário de comprimento n, o código público de um sistema criptográfico de McEliece. Toma-se $\mathcal{P} = \mathcal{SSA}(C)$ e considera-se $E_{\mathcal{P}}$, o conjunto de todas as palavras binárias cujo suporte é a união de células de $\mathcal{P}$ ou seja, o conjunto de todo $a \in GF(2)^n$ tal que para qualquer célula $\mathcal{P}_i$ de $\mathcal{P}$, $\forall \gamma, \gamma' \in \mathcal{P}_i$ tem-se $a_\gamma = a_{\gamma'}$. Com isso define-se o código:

$$\mathcal{I}(C) = E_{\mathcal{P}} \cap C. \tag{6.3}$$

Como as palavras de $\mathcal{I}(C)$ têm coordenadas repetidas (as coordenadas indexadas por elementos pertencentes a mesma célula $\mathcal{P}_i$), à semelhança do que se fez para o subcódigo idempotente do código de Goppa, toma-se um elemento para cada célula de $\mathcal{P}$. Considerando o subconjunto, T, do suporte do código constituído exactamente por um elemento de cada célula de $\mathcal{P}$ e a projecção κ_T definida em (6.1), define-se o código:

$$\widetilde{\mathcal{I}}(C) = \{\kappa_T(x) \mid x \in \mathcal{I}(C)\}.$$

Proposição 6.3. *[19] Se a cardinalidade das células de $\mathcal{P} = \mathcal{SSA}(C)$ coincide com a das classes de conjugados e se $C \sim \mathcal{G}(L, g(X))$, então $\widetilde{\mathcal{I}}(C) \sim I(R, g(X))$.*

A matriz geradora de $\widetilde{\mathcal{I}}(C)$ pode ser calculada a partir de qualquer matriz geradora de C. Uma vez calculado $\mathcal{P} = \mathcal{SSA}(C)$, calcula-se a matriz geradora de $\mathcal{I}(C)$ pela equação (6.3). Eliminado as colunas repetidas obtém-se a matriz geradora de $\widetilde{\mathcal{I}}(C)$.

Desta forma, o problema reduz-se a encontrar um polinómio $g(X) \in GF(2)[X]$ tal que $I(R, g(X)) \sim \widetilde{\mathcal{I}}(C)$.

Resumindo, o procedimento a aplicar é:

1. Calcula-se $\mathcal{P} = \mathcal{SSA}(G')$, sendo G' a matriz geradora do código público.
2. Calcula-se a matriz de paridade, H, de $\widetilde{\mathcal{I}}(C)$.
3. Calcula-se $\mathcal{P}' = \mathcal{SSA}(H)$.
4. Para cada polinómio irredutível $g(X) \in GF(2)[X]$:
   - determina-se a matriz de paridade, H_I, de $I(R, g(X))$;
   - se $\mathcal{SSA}(H_I) \sim \mathcal{P}'$, então retornar $g(X)$.

Conhecendo $g(X)$, o passo seguinte é descobrir uma permutação, π, tal que $C \sim \mathcal{G}$.

6.2.3 Permutação entre códigos equivalentes

Este problema é tratado em [30] e é usado em [19] para efectivar o ataque que estamos a descrever. No que segue, apresentaremos os conceitos fundamentais à sua compreensão.

Definição 6.4. *Sejam um código linear binário, C, de comprimento n e o conjunto, $I_n = \{1, 2, \ldots, n\}$, pelo qual estão indexadas as coordenadas de cada palavra de C. Para toda permutação ρ em I_n, define-se*

$$\rho(C) = \{x_{\rho^{-1}(1)} x_{\rho^{-1}(2)} \ldots x_{\rho^{-1}(n)}\}, \text{ com } x_1 x_2 \ldots x_n \in C \tag{6.4}$$

Diz-se que C e $\rho(C)$ são equivalentes por permutação e denotamos por $C \sim \rho(C)$.

Definição 6.5. *O grupo de permutações de automorfismos de um código, C, de comprimento n, denotado por $PAut(C)$, é o subgrupo de todas as permutações, ρ, do grupo simétrico de I_n tal que $\rho(C) = C$.*

Se $PAut(C)$ é não trivial e se C' é permutação equivalente a C então, existem permutações, ρ, tais que $C' = \rho(C)$.

Definição 6.6. *Para qualquer subconjunto, J, de I_n define-se:*

- *"Punctured codes", C_J, constituído por todas as palavras de C nas quais as coordenadas indexadas por J são substituídas por zero.*

- *"Shortened codes", $C_{/J}$, é o subconjunto de todas as palavras de C cujas coordenadas indexadas por J são iguais a zero.*

Exemplo 6.7. *Seja um código binário $C - [6, 3, 2]$, com a matriz geradora*

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

As palavras deste código são:

$$\begin{Bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Se as coordenadas das palavras de códigos estão indexadas pelo conjunto $I_n = \{1, 2, \ldots, 6\}$, *e, se consideramos um subconjunto,* $J = \{5, 6\}$, *de* I_n, *então definimos o "punctured code",* C_J, *com as seguintes palavras:*

$$C_{\{5,6\}} = \begin{Bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

O "shortened code", $C_{/J}$, *é:*

$$C_{/\{5,6\}} = \begin{Bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Nota 6.8. *Na definição usual (ver [20, 14]) dos "punctured codes" e "shortened codes", as coordenadas indexadas por* J *são suprimidas. Mas aqui, são deixadas, por conveniência, na palavra (todas nulas).*

Definição 6.9. *Sejam* $\mathcal{L}_n$, *o conjunto de todos os códigos de comprimento* n *e* $\mathcal{L} = \bigcup_{n>0} \mathcal{L}_n$, *o conjunto de todos os códigos. Uma invariante sobre um conjunto* $\boldsymbol{E}$, *é definida como sendo uma transformação de* $\mathcal{L}$ *em* $\boldsymbol{E}$, *sob acção da qual, quaisquer dois códigos equivalentes por permutação, tomam o mesmo valor.*

O enumerador de peso de Hamming, $\mathcal{W}$, definido pelo polinómio $\mathcal{W} = \sum_{i=1}^{n} A_i x^i$, onde A_i denota o número de palavras de peso i, é uma invariante que pode ajudar a decidir quando é que dois códigos são equivalentes ou não. Dois códigos com enumerador de peso diferentes não podem ser equivalentes. Infelizmente, pode-se ter códigos não equivalentes com mesmo enumerador de erro. Todavia, isso ocorre com pequena probabilidade quando o código é escolhido aleatoriamente.

Aplicando o enumerador de peso de Hamming sobre os "punctured codes" consegue-se produzir uma propriedade local, que seja ao mesmo tempo propriedade do código e de uma das suas posições, que se designa por **assinatura**.

Definição 6.10. *Uma assinatura, $\mathcal{S}$, sobre um conjunto $\mathbb{F}$ mapeia um código, C, de comprimento n e um elemento, i, de I_n em um elemento de $\mathbb{F}$ e, é tal que para toda permutação ρ de I_n,*

$$\mathcal{S}(C, i) = \mathcal{S}(\rho(C), \rho(i)). \tag{6.5}$$

Assim, pode-se associar a invariante, $\mathcal{W}$, uma assinatura

$$\mathcal{S}_{\mathcal{W}} : (C, i) \mapsto \mathcal{W}(C_i). \tag{6.6}$$

Desta forma, se dois códigos, C e C', são equivalentes por permutação com $C' = \rho(C)$, então temos,

$$\mathcal{W}(C_i) = \mathcal{W}(C'_{\rho(i)}),\ \forall i \in I_n. \tag{6.7}$$

Com isso, é fácil ver que, se $\mathcal{S}(C, i) \neq \mathcal{S}(C', j)$ para qualquer assinatura, $\mathcal{S}$, então $\rho(i) \neq j$. Assim, *a imagem de i pela permutação ρ, tem de ser escolhida de entre os índices j que verificam $\mathcal{S}(C', j) = \mathcal{S}(C, i)$*. O número de valores diferentes assumido por uma dada assinatura para um código, C, é, deste modo, de capital importância para medir quão eficiente ela é. Além disso, se a transformação $i \mapsto \mathcal{S}(C, i)$ assume valores diferentes para todos elementos de I_n, então a permutação entre C e qualquer versão permutada de C pode ser recuperada.

Para qualquer assinatura, $\mathcal{S}$, e qualquer código, C, de comprimento n, consideremos uma relação $\mathcal{R}$ no conjunto $I_n = \{1, 2, \ldots, n\}$, definida da seguinte forma:

$$i\mathcal{R}j \Leftrightarrow \mathcal{S}(C, i) = \mathcal{S}(C, j) \text{ com } i \text{ e } j, \text{ índices de } I_n.$$

$\mathcal{R}$ é uma relação de equivalência. As suas classes laterais produzem uma partição de I_n que coincidirá com qualquer outra obtida a partir de um código C' equivalente a C.

Definição 6.11. *Dada uma assinatura, $\mathcal{S}$, sobre $\boldsymbol{E}$, define-se $J_\epsilon = \{i \in I_n \mid \mathcal{S}(C,i) = \epsilon\}$. À sequência $(J_\epsilon)_{\epsilon \in \boldsymbol{E}}$ designa-se por (C,S)-**partição de** I_n. E, para qualquer subconjunto $\boldsymbol{E}'$ de $\boldsymbol{E}$, o conjunto*

$$\bigcup_{\epsilon \in \boldsymbol{E}'} J_\epsilon = \{i \in I_n \mid \mathcal{S}(C,i) \in \boldsymbol{E}'\} \tag{6.8}$$

*é designado (C,S)-**subconjunto discriminado de** I_n.*

Proposição 6.12. *Seja um código, C, de comprimento n e dada uma assinatura, $\mathcal{S}$, sobre $\boldsymbol{E}$ e seja $(J_\epsilon)_{\epsilon \in \boldsymbol{E}}$ a (C,S)-partição de I_n. Para toda permutação, ρ, em I_n, $(\rho(J_\epsilon))_{\epsilon \in \boldsymbol{E}}$ é a $(\rho(C),S)$-partição de I_n.*

A (C,S)-partição de I_n é a partição obtida por aplicação da assinatura $\mathcal{S}$ a todas as posições do código C. Um (C,S)-subconjunto discriminado de I_n, é o conjunto de posições que somos capazes de reconhecer em qualquer código que seja equivalente por permutação a C. Se $C \sim C' = \rho(C)$, a qualquer (C,S)-subconjunto discriminado, J, de I_n corresponderá um (C',S)-subconjunto discriminado,

$$J' = \{i \in I_n \mid \mathcal{S}(C',i) \in \mathcal{S}(C,J)\} = \rho(J). \tag{6.9}$$

E, em particular, tem-se $C'_{J'} = \rho(C_J)$.

Quanto mais fino for a partição associada, melhor será a correspondente assinatura. O ideal é essa partição ser constituída por n elementos simples, significando que se conseguiu definir propriedades diferentes para cada posição do código. Com isso podemos classificar as assinaturas:

Definição 6.13. *Dado um código, C, de comprimento n.*

- *Uma assinatura $\mathcal{S}$ é denominado, **Assinatura discriminante** para C, se existe i e j em I_n, tal que $\mathcal{S}(C,i) \neq \mathcal{S}(C,j)$;*
- *Uma assinatura $\mathcal{S}$ é denominado, **Assinatura totalmente discriminante** para C, se para todo i e j em I_n, $\mathcal{S}(C,i) \neq \mathcal{S}(C,j)$;*
- *Se $C' = \rho(C)$ e se $\mathcal{S}$ é uma assinatura totalmente discriminante para C, então para todo $i \in I_n$ existe um único $j \in I_n$, tal que $\mathcal{S}(C,i) = \mathcal{S}(C',j)$ e tem-se $\rho(i) = j$. Assim, se obtém a permutação ρ.*

Para um código, C, se uma assinatura, $\mathcal{S}$, é discriminante mas não totalmente discriminante, pode-se, por um processo de refinamento desta assinatura, obter informações adicionais sobre ρ tal que $C' = \rho(C)$. Este processo baseia-se no seguinte:

Proposição 6.14. *Seja $\mathcal{T}$ uma assinatura sobre $\boldsymbol{E}$. Para qualquer subconjunto $\boldsymbol{E}'$ de $\boldsymbol{E}$, e para todo código, C, de comprimento n, o mapa $\mathcal{S}_{\mathcal{T},\boldsymbol{E}'}$ definido por*

$$\mathcal{S}_{\mathcal{T},\boldsymbol{E}'}(C,i) = \mathcal{S}(C_{\mathcal{K}_{\mathcal{T},\boldsymbol{E}'}(C)}, i) \tag{6.10}$$

onde $\mathcal{K}_{\mathcal{T},\boldsymbol{E}'}(C) = \{i \in I_n \mid \mathcal{T}(C,i) \in \boldsymbol{E}'\}$, é uma assinatura.

Exemplo 6.15. *Sejam dois códigos (não lineares) equivalentes, C e C', definidos a seguir:*

$$C = \{10101, 00111, 10011, 11100, 11011\}$$

e

$$C' = \{01110, 10110, 11010, 01101, 11011\}.$$

Tomamos, como invariante, o enumerador de peso de Hamming denotado aqui por $\mathcal{W}$, e recuperamos a permutação entre eles, como segue. Definimos a assinatura associada a $\mathcal{W}$, $\mathcal{S}_{\mathcal{W}}$, sobre os "punctured codes", C_J e C'_J:

$$\begin{aligned}
C_1 &= \{00101, 00111, 00011, 01100, 01011\} \rightarrow \mathcal{S}_{\mathcal{W}}(C,1) = \mathcal{W}(C_1) = 3X^2 + 2X^3 \\
C_2 &= \{10101, 00111, 10011, 10100\} \rightarrow \mathcal{S}_{\mathcal{W}}(C,2) = \mathcal{W}(C_2) = X^2 + 3X^3 \\
C_3 &= \{10001, 00011, 10011, 11000, 11011\} \rightarrow \mathcal{W}(C_3) = 3X^2 + X^3 + X^4 \\
C_4 &= \{10101, 00101, 10001, 11100, 11001\} \rightarrow \mathcal{W}(C_4) = 2X^2 + 3X^3 \\
C_5 &= \{10100, 00110, 10010, 11100, 11010\} \rightarrow \mathcal{W}(C_5) = 3X^2 + 2X^3
\end{aligned}$$

Como se vê, a assinatura $\mathcal{W}$ não é totalmente discriminate para C, pois, as posições 1 e 5 não se distinguem.

Em relação ao código C', temos:

$$\begin{aligned}
C'_1 &= \{01110, 00110, 01010, 01101, 01011\} \rightarrow \mathcal{S}_{\mathcal{W}}(C',1) = \mathcal{W}(C'_1) = 2X^2 + 3X^3 \\
C'_2 &= \{00110, 10110, 10010, 00101, 10011\} \rightarrow \mathcal{S}_{\mathcal{W}}(C',2) = \mathcal{W}(C'_2) = 3X^2 + 2X^3 \\
C'_3 &= \{01010, 10010, 11010, 01001, 11011\} \rightarrow \mathcal{W}(C'_3) = 3X^2 + X^3 + X^4 \\
C'_4 &= \{01100, 10100, 11000, 01101, 11001\} \rightarrow \mathcal{W}(C'_4) = 3X^2 + 2X^3 \\
C'_5 &= \{01110, 10110, 11010, 01100\} \rightarrow \mathcal{W}(C'_5) = X^2 + 3X^3
\end{aligned}$$

comparando as assinaturas para os dois códigos, chegamos por (6.9) *a que:*

$$\begin{aligned} \rho(2) &= 5 \\ \rho(3) &= 3 \\ \rho(4) &= 1 \\ \rho(\{1,5\}) &= \{2,4\} \end{aligned}$$

Tomamos um par de códigos equivalentes, C_2 *e* C_5' *por exemplo, sobre os quais aplicamos a assinatura* $\mathcal{S}_{\mathcal{T},\boldsymbol{E}'}$*, onde o conjunto* $\boldsymbol{E}'$ *é* $\{2\}$ *e* $\{5\}$ *respectivamente e as posições variam em* $J = \{1,5\}$ *e* $J' = \{2,4\}$*:*

$$\begin{aligned} C_{\{1,2\}} &= \{00101, 00111, 00011, 00100\} \ \rightarrow \ \mathcal{W}(C_{\{1,2\}}) = X + 2X^2 + X^3 \\ C_{\{2,5\}} &= \{10100, 00110, 10010\} \ \rightarrow \ \mathcal{W}(C_{\{2,5\}}) = 3X^2 \end{aligned}$$

Como se pode ver, conseguimos, desta forma, discriminar totalmente C*. Espera-se que o mesmo aconteça com* C'*.*

$$\begin{aligned} C'_{\{2,5\}} &= \{00110, 10110, 10010, 00100\} \ \rightarrow \ \mathcal{W}(C'_{\{2,5\}}) = X + 2X^2 + X^3 \\ C'_{\{4,5\}} &= \{01100, 10100, 11000\} \ \rightarrow \ \mathcal{W}(C'_{\{4,5\}}) = 3X^2 \end{aligned}$$

Comparando, de novo, as assinaturas, conclui-se que $\rho(1) = 2$ *e* $\rho(5) = 4$*. E, desta forma, a permutação* ρ *fica totalmente determinada:*

$$\rho = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix} = (2,5,3,1,4).$$

Podemos verificar que, se aplicarmos a permutação ρ ao código C, o que significa aplica-la a cada uma das palavras de C, obteremos o código C'.

Se $x_1x_2x_3x_4x_5$ é uma palavra de C, então $\rho(x_1x_2x_3x_4x_5) = x_{\rho^{-1}(1)}x_{\rho^{-1}(2)}x_{\rho^{-1}(3)}x_{\rho^{-1}(4)}x_{\rho^{-1}(5)}$, por definição. Assim, $\rho(10101) = 01110$ que pertence a C'. As restantes palavras podem ser, analogamente, verificadas.

Um exemplo com um código linear binário pode ser encontrado em [30]

Apesar destas possibilidades de ataques, o sistema criptográfico de McEliece é considerado seguro. Os ataques, até agora conhecidos, não afectam a estrutura do sistema porque não revelam, totalmente, a chave secreta. As chaves fracas constituem, apenas, uma subclasse de possíveis chaves para o sistema. A segurança pode ser mantida com uma boa escolha da chave secreta [19]. Ademais, recentemente foi apresentada uma variante interessante do sistema de McEliece que, para além de ser resistente a este tipo de ataque reforçando a segurança da chave pública, permite reconsiderar a hipótese do uso de famílias de códigos anteriormente consideradas inseguras para o sistema como é o caso de Reed-Solomon. Esta variante baseia-se no facto de existirem algumas classes de matrizes densas com efeito de propagação limitada sobre o vector erro. O uso desta matriz no lugar da permutação, P, do sistema original, permite disfarçar melhor a chave privada dentro da pública evitando que esta seja permutação equivalente daquela (ver [1]).

Capítulo 7

Assinatura digital

O sistema criptográfico de McEliece bem como a variante de Niederreiter, usando código de o Goppa, tem resistido a vários tipos de ataques e, são considerados seguros mesmo na presença da computação quântica. Por isso, são considerados candidatos forte às soluções criptográficas do futuro. Seria importante contar-se com assinaturas digitais que beneficiasse da segurança destes sistemas. Vários esforços neste sentido fracassaram. Alias, o próprio McEliece teria dito em [21] que, o algoritmo de descodificação que descreve não pode ser usado para produzir assinaturas digitais não falsificáveis. Isto deve-se ao facto de que, de entre todos os vectores -(2^n)- de comprimento n, que constituem o espaço sobre o qual está definido o código, apenas uma pequena fracção são descodificáveis. Portanto, uma eventual obtenção de assinatura digital funcional, passava por encontrar um algoritmo que pudesse descodificar quaisquer das 2^n palavras - *algoritmo de descodificação completa.*

Courtois et al, [5], propõe uma solução aproximada para o problema, a que chamaram de *descodificação quase completa.* Supõe-se t a capacidade de correcção de código e quer-se um algoritmo que corrija mais do que t erros. Seja uma palavra binária de comprimento n, com erros em $t+1$ posições. O algoritmo primeiro altera um bit numa das n posições e, em seguida, aplica o algoritmo de correcção de erros. Se resultar, foi porque o bit alterado é um dos $t+1$ bits erróneos. Se não, só temos de tentar de novo alterando outra posição. Desta forma, temos a certeza que ao fim de, no máximo, n tentativas, conseguiremos descodificar qualquer palavra com $t+1$ erros. Se as palavras tiverem $t+2, t+3, \ldots, t+\theta$ erros, procede-se da mesma forma alterando $2, 3, \ldots, \theta$

bits aleatórios, ao mesmo tempo.

Para as palavras de comprimento n, contendo $t+\theta$ erros a descodificar, a probabilidade de sucesso deste algoritmo é de

$$P = \frac{\binom{t+\theta}{\theta}}{\binom{n}{\theta}}$$

que aumenta exponencialmente com θ. Todavia, quando θ é muito grande, o problema torna-se intratável, pois é cada vez maior o número de combinações das posições dos bits a serem alterados. Portanto, θ deve ser pequeno. Considera-se o mais pequeno inteiro, θ_{min}, para o qual o volume da esfera de raio $t+\theta$ seja maior que 2^{n-k}. A preocupação é aumentar a densidade de palavras descodificáveis no espaço de todas as palavras, o que significa aumentar a probabilidade de uma palavra, escolhida aleatoriamente, ser descodificável. Isto, na linguagem de síndrome, significa aumentar a hipótese de um síndrome aleatório corresponder a um padrão de erro corrigível pelo código.

Para descodificar um síndrome correspondente a um erro de peso $t+\theta$ quando o código só corrige t erros, adiciona-se θ colunas aleatórias de H - matriz de paridade do código - a este síndrome, e tenta-se a descodificação. Se todas as θ colunas corresponderem às posições de erros, então o novo síndrome obtido corresponderá a um padrão de erro de peso t, logo descodificável. Como as θ colunas são tomadas de forma aleatória, este método produz o mesmo resultado se for gerado um novo síndrome para cada tentativa de descodificação.

Os parâmetros originais de McEliece, não são convenientes pois, dos 2^{500} síndromes apenas $\sum_{i=1}^{50}\binom{1024}{i} \backsimeq 2^{284}$ são descodificáveis - os cujo peso é inferior ou igual a 50, o que traduz numa probabilidade de sucesso de 2^{-216} para cada síndrome ou numa média de 2^{216} tentativas, o que é elevado. É necessário, portanto, ajustar os parâmetros por forma a conseguir valores mais razoáveis.

Dado um código binário de Goppa de comprimento $n = 2^m$ capaz de corrigir t erros, a sua dimensão é dada por $k = n - tm$. A probabilidade de sucesso na escolha aleatória de um síndrome que seja descodificável é a razão entre o número de síndromes descodificáveis, S_{dec}, e número total de síndromes, S_{tot}. E, como n é muito grande comparativamente a t, tem-se que:

$$S_{dec} = \sum_{i=1}^{t}\binom{n}{i} \backsimeq \binom{n}{t} \backsimeq \frac{n^t}{t!}$$

como, $S_{tot} = 2^{n-k} = 2^{mt} = n^t$, logo

$$P = \frac{S_{dec}}{S_{tot}} \backsimeq \frac{1}{t!}.$$

O número de tentativas para encontrar um síndrome descodificável é próximo de $t!$, e, para ter um esquema de assinatura razoável, propõe-se que t não seja superior a 10. Valor estabelecido após análise da eficiência do algoritmo e da carga de trabalho exigida numa eventual implementação dos melhores ataques ao sistema de McEliece, (ver [5, Pág. 168]). Porém, para um código que corrige poucos erros, é necessário um comprimento muito grande por forma a garantir um nível de segurança aceitável. Desta forma, aconselha-se dimensão de, ao menos, 2^{15} com capacidade de corrigir 10 erros ou de 2^{16} corrigindo 9 erros. A segunda proposta é considerada melhor visto ser dez vezes mais rápida.

Esquema de assinatura

Definição 7.1. *[23][Função de Hash] Uma* ***função de Hash****, h, é uma função, computacionalmente eficiente, que transforma qualquer string binário de comprimento arbitrário em uma string binária de comprimento fixo, denominado* ***valor-hash****.*

Para uso criptográfico, uma função de Hash, h, é normalmente definida de tal forma que seja computacionalmente inviável qualquer esforço para encontrar dois elementos diferentes (x e y) com o mesmo valor-hash ($h(x) = h(y)$) e, que dado um valor-hash específico, y, seja computacionalmente inviável encontrar um elemento, x, tal que $h(x) = y$.

Como a variante utilizada para assinaturas digitais é a de Niederreiter, um dos inputs para a sua construção são os síndromes, palavras binárias de comprimento $n - k$ (obtidos, aqui, por aplicação de uma função de Hash) no lugar das de comprimento n.

Seja $\mathbf{h}$, uma função Hash cujos outputs sejam palavras de comprimento $n-k$. $\mathbf{h}$ resume qualquer documento, D, num síndrome $\mathbf{s}$ (descodificável ou não). Com os parâmetros escolhidos, $n = 2^{16}$ e $t = 9$, é preciso, cerca de, 9! tentativas para obter, aleatoriamente, um síndrome descodificável. Encontrando o primeiro síndrome descodificável, seja isto feito à i-ésima tentativa[1] representado por i_0, calcula-se o padrão de erro cor-

[1]Em cada tentativa faz-se $D_i = (D, i)$

respondente, e, que, concatenado i_0, passa a ser a assinatura, e, é enviada anexada à mensagem original D. Na tabela 7.1 ilustramos este processo:

Alice				**Bob**		
Chave privada: $H_a,\ S,\ P$ Chave pública: H'_a				Chave privada: $H_b,\ S,\ P$ Chave pública: H'_b		
Hashing(D)	Cifra(D)	Decifra(s_{i_0})	Envia	Decifra(r)	Cifra(e)	Hashing(D, i_0)
$s_{i_0} = h(D, i_0)$	$r = H'_b \cdot D^T$	$e^T \cdot H_a = s_{i_0}$	r e (e, i_0)	D	$s_1 = H'_a \cdot e^T$	$s_2 = h(D, i_0)$
Obs: A assinatura (e, i_0), é aceite como fidedigna se $s_1 = s_2$						

Tabela 7.1: Assinatura e sua verificação

Capítulo 8

Conclusão

A tese que ora apresentamos, descreve um sistema criptográfico, baseado em códigos correctores de erros, proposto por McEliece a, sensivelmente, trinta anos atrás.

Para a compreensão e consequente descrição do sistema, foi preciso uma visão geral sobre os códigos lineares, em particular, o código binário de Goppa, sua estrutura e o processo de correcção de erros. Isto exigiu, por sua vez, conhecimentos sobre corpos de Galois, sua construção e operações, o que nos forneceu apetrechos suficientes para, adicionalmente, implementar, no Maple, um programa que cifra e decifra mensagem com base no referido sistema criptográfico.

Do estudo bibliográfico sobre ataques a este sistema, ficamos cada vez mais convencidos da sua segurança. O ataque estrutural proposto em [19] pode ser evitado. Além disso, a segurança da chave pública pode ser reforçada conforme a variante proposta em [1] que, ao mesmo tempo, abre hipótese de utilização em, segurança, de outras famílias de códigos, mormente, o Reed-Solomon que permite obter chaves públicas mais pequenas resolvendo, assim, uma antiga desvantagem do sistema em relação, por exemplo, ao RSA.

A construção de assinaturas digitais, um dos factores que ditou o seu pouco uso, foi ultrapassado em [5]. Desta forma, o sistema de McEliece reúne condições de constituir alternativa aos sistemas mais usado hoje como é o RSA, acrescido, ainda, o facto de ser resistente a ataques quânticos.

Bibliografia

[1] Baldi, M., Bianchi, M., Chiaraluce , F., Rosenthal, J. and Schipani, D. A variant of the McEliece cryptosystem with increased public key security. In Proceedings of the WCC 2011 - Seventh Workshop on Coding and Cryptography, Paris, FR, 11 April 2011 - 14 April 2011 (Paris, FR, 2011). HAL-Inria, [insert City of Publication],[insert 2011 of Publication]. 71, 77

[2] Berlekamp, E., McEliece, R. and van Tilborg, H. On the inherent intractability of certain coding problems (Corresp.). Information Theory, IEEE Transactions on, 24, **3** (1978), 384-386. 1, 35

[3] Bernstein, D. Grover vs. McEliece. Springer Berlin Heidelberg, City, 2010. 60

[4] Bernstein, D. J., Lange, T. and Peters, C. Attacking and Defending the McEliece Cryptosystem. In Proceedings of the Proceedings of the 2nd International Workshop on Post-Quantum Cryptography (Cincinnati, OH, 2008). Springer-Verlag, [insert City of Publication],[insert 2008 of Publication]. 60

[5] Courtois, N., Finiasz, M. and Sendrier, N. How to Achieve a McEliece-Based Digital Signature Scheme. Advances in Cryptology - ASIACRYPT 2001. Springer Berlin / Heidelberg, City, 2001. 2, 46, 73, 75, 77

[6] Diffie, W. and Hellman, M. New directions in cryptography. Information Theory, IEEE Transactions on, 22, **6** (1976), 644-654. 1

[7] Dinh, H., Moore, C. and Russell, A. McEliece and Niederreiter Cryptosystems That Resist Quantum Fourier Sampling Attacks. Advances in Cryptology - CRYPTO 2011. Springer Berlin / Heidelberg, City, 2011. 2

[8] Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. Information Theory, IEEE Transactions on, 31, **4** (1985), 469-472. 1

[9] Elia, M., Viterbo, E. and Bertinetti, G. Decoding of binary separable Goppa codes using Berlekamp-Massey algorithm. Electronics Letters, 35, **20** (1999), 1720-1721. 40

[10] Engelbert, D., Overbeck, R. and Schmidt, A. A Summary of McEliece-Type Cryptosystems and their Security. City, 2007. 61

[11] Gibson, K. The Security of the Gabidulin Public Key Cryptosystem. Advances in Cryptology - EUROCRYPT '96. Springer Berlin / Heidelberg, City, 1996. 46

[12] Hamming, R. W. Error detecting and error correcting codes. Bell System technical journal, 29, **2** (1950), 147-160. 25

[13] Heyse, S. Code-based cryptography: Implementing the McEliece scheme in reconfigurable hardware. 2009. 1, 40, 47

[14] Huffman, W. C. and Pless, V. Fundamentals of error-correcting codes. Cambridge Univ Pr, 2003. 26, 30, 66

[15] Ireland, K. F. and Rosen, M. I. A Classical Introduction to Modern Number Theory. Springer-Verlag, 1990. 11

[16] Koblitz, N. Elliptic Curve Cryptosystems. Mathematics of Computation, 48, **177** (1987), 203-209. 1

[17] Lidl, R. and Niederreiter, H. Finite Fields. Cambridge University Press, 1996. 3, 7, 11, 14, 15, 19, 20

[18] Loeloeian, M. and Conan, J. A transform approach to Goppa codes. Information Theory, IEEE Transactions on, 33, **1** (1987), 105-115. 40, 42

[19] Loidreau, P. and Sendrier, N. Weak keys in the McEliece public-key cryptosystem. Information Theory, IEEE Transactions on, 47, **3** (2001), 1207-1211. 20, 61, 62, 64, 65, 71, 77

[20] MacWilliams, F. J. *The theory of error-correcting codes*. North-Holland, Amsterdam, 1977. 18, 26, 29, 33, 35, 40, 66

[21] McEliece, R. A public-key cryptosystem based on algebraic coding theory. 1978. 1, 45, 59, 60, 73

[22] McEliece, R. J. The theory of information and coding. Cambridge Univ Pr, 2002. 28

[23] Menezes, A. J., Van Oorschot, P. C. and Vanstone, S. A. Handbook of Applied Crytography. Crc Press, 1997. 75

[24] Miller, V. Use of Elliptic Curves in Cryptography Advances in Cryptology - CRYPTO '85 Proceedings. Springer Berlin / Heidelberg, City, 1986. 1

[25] Minder, L. and Shokrollahi, A. Cryptanalysis of the Sidelnikov Cryptosystem. Advances in Cryptology - EUROCRYPT 2007. Springer Berlin / Heidelberg, City, 2007. 45

[26] Overbeck, R. Extending Gibson's Attacks on the GPT Cryptosystem. Coding and Cryptography. Springer Berlin / Heidelberg, City, 2006. 46

[27] Patterson, N. The algebraic decoding of Goppa codes. Information Theory, IEEE Transactions on, 21, **2** (1975), 203-207. 40

[28] Purser, M. Introduction to Error-Correcting Codes. Artech House, 1995. 33

[29] Rivest, R. L., Shamir, A. and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21, **2** (1978), 120-126. 1

[30] Sendrier, N. Finding the permutation between equivalent linear codes: the support splitting algorithm. Information Theory, IEEE Transactions on, 46, **4** (2000), 1193-1203. 62, 65, 70

[31] Sendrier, N. Encoding information into constant weight words. City, 2005. 58

[32] Shor, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings of the Proceedings of the 35th Annual Symposium on Foundations of Computer Science (1994). IEEE Computer Society, [insert City of Publication],[insert 1994 of Publication]. 2

[33] Sidelnikov V, M. and Shestakov S, O. On insecurity of cryptosystems based on generalized Reed-Solomon codes. City, 1992. 45

[34] Stern, J. A method for finding codewords of small weight. In Proceedings of the Proceedings of the 3rd International Colloquium on Coding Theory and Applications (1989). Springer-Verlag, [insert City of Publication],[insert 1989 of Publication]. 60

[35] Yuan Xing, L., Deng, R. H. and Xin Mei, W. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. Information Theory, IEEE Transactions on, 40, **1** (1994), 271-273. 46

Apêndice A

Apêndices

A.1 Apêndice A1

Aqui, apresentamos com conjunto de procedimentos desenvolvidos em Maple 14.01, que constituiu numa plataforma de cálculo e simplificação de expressões sobre o corpo finito $GF(2^m)$, necessários à implementação do sistema criptográfico de McEliece:

1. Define o corpo $GF(2^m)$, guardando na tabela T, os seus elementos:

```
> f:=alpha^5+alpha^4+alpha^3+alpha+1:    #Polinómio gerador do corpo de Galois
> g:=x->x^4+x^3+1:                       #Polinómio de Goppa
> p:=2:
> m:=degree(f):
> t:=degree(g(x)):
> k:=p^m-m*t:
> Corpo:=GF(p,m,f):
> T:=Matrix(p^m,2):                      # Tabela GF(p^m)
> a:=Corpo:-ConvertIn(alpha):
> T(1,1):=0: T(1,2):=0:
> T(2,1):=1:
> T(2,2):=1:
> printf("potencia                         Polin\n"):
> printf(".....................................\n"):
> for i from 3 to p^m do
>    T(i,1):=alpha^(i-2):
>    T(i,2):=Corpo:-`^`(a,i-2):
>    #printf("%A                        %A\n",T(i,1),T(i,2)):
>    #print(T(i,1).............T(i,2)):
> od:
> printf(".....................................\n"):
```

2. Procedimento que converte um polinómio de $GF(2^m)$ na potência do corpo:

```
ConvNaPotDoCorpo:=proc(pol)
> local pot,i,polRed:
> polRed:=rem(pol,f,alpha)mod 2:
> for i from 1 to p^m do
>   if convert(T(i,2),polynom)=polRed then
>   pot:=T(i,1):
> fi:
> od:
> return pot;
> end proc:
```

3. Procedimento calcula o quociente de dois polinómios de $GF(2^m)$, convertendo o resultado na potência do corpo:

```
RedFracAPol:=proc(frac)
> local fracNaFormDeProd,a,b,result:
> Gcdex(denom(frac),f,alpha,'a','b')mod 2:
> fracNaFormDeProd:=numer(frac)*a:
> result:=ConvNaPotDoCorpo(fracNaFormDeProd):
> return result:
> end proc:
```

4. Procedimento que reduz os coeficientes de um polinómio aos elementos do corpo:

```
RedCoefDePol:=proc(pol)
> local i, vectorDeCoef, vectorDeCoefRed:=Vector(),numCoef:
> numCoef:=degree(pol,x)+1:
> vectorDeCoef:=CoefficientVector(pol,x):
> for i from 1 to numCoef do
>    vectorDeCoefRed(i):=RedFracAPol(vectorDeCoef(i)):
> od:
> return vectorDeCoefRed():
> end proc:
```

5. Procedimento que devolve polinómio em x com coeficientes sobre o corpo $GF(p^m)$:

```
PolComCoefNumCorpo:=proc(pol)
> local PolResult, coefRed:
> coefRed:=RedCoefDePol(pol):
> PolResult:=FromCoefficientVector(coefRed,x):
> return PolResult:
> end proc:
```

6. Calcula e guarda numa matriz, a inversa de cada $g(\gamma_i)$:

```
Inver:=Matrix(1,p^m):
> for j from 1 to p^m do
> Gcdex(g(T(j,1)),f,alpha,'u','v')mod 2:
> Inver(j):=PolComCoefNumCorpo(u):
> od:
```

7. Procedimento que localiza e corrige até t erros na transmissão de um vector c:

```
LocalizaErro:=proc(t,c::Matrix())
> local i,j,k,l,u,v,o,q,w,lc,coordS,test,ExpS,M1,M2,sol,coefpoLacaliz,zeros,pos,erro,codword:
> coordS:=Matrix(1,2*t):test:=true:
> w:=t:
> while test=true do
>    for j from 1 to 2*w do  #Ciclo que calcula as coordenadas do síndrome
>      ExpS:=0:
>      for i from 1 to p^m do
>         ExpS:=ExpS+c(i)*(T(i,1)^(j-1))*Inver(i)^2:
>      od:
>      coordS(j):=PolComCoefNumCorpo(ExpS):
>    od:
>    M1:=Matrix():
>    for k from 1 to w do
>      for l from 0 to w-1 do
>        M1(k,l+1):=coordS(k+l):
>      od:
>    od:
>    M2:=Vector():
>    for u from 1 to w do
>      M2(u):=coordS(w+u):
>    od:
>    sol:=convert(linsolve(M1,M2,'we'),Vector)mod 2:
>    coefpoLacaliz:=Vector(w+1):
>    for o from 1 to w do
>       coefpoLacaliz(o):=PolComCoefNumCorpo(sol(o)):
>       coefpoLacaliz(w+1):=1:
>    od:
>    test:=IsVectorShape( coefpoLacaliz(1..w), zero ):
>    w:=t-1:
> od:
> lc:=FromCoefficientVector(coefpoLacaliz,x):  #Polinómio localizador de erros
> zeros:={}:
# Força bruta para encontrar as raízes do pol. localizador de erros
> for q from 1 to p^m do
>   if rem(eval(lc,x=T(q,1))mod 2,f,alpha)mod 2=0 then
>      zeros:={op(zeros),T(q,1)}:
>   fi:
> od:
> #return zeros;
> pos:=map(x->degree(x)+2,zeros):erro:=Matrix(1,p^m):
> if -(infinity) in pos then
>    erro(1,1):=1:
> fi:
> for v from 2 to p^m do
>    if v in pos then
>      erro(1,v):= 1:
>    else
>      erro(1,v):=0:
>   fi:
> od:
> codword:=Matrix(c+erro)mod 2:    #Correcção do erros
> #return convert(codword,array),zeros,pos,convert(erro,array):
> return codword:
```

```
> end proc:
```

8. Calcula a matriz de paridade com elementos em $GF(2^m)$:

```
MatrizDeParidade:=proc(f,g)
> local i,j, t, q,matriz:=Matrix():
> t:=degree(g):q:=p^m:
> for i from 0 to t-1 do
>   for j from 1 to q do
>     matriz(i+1,j):=PolComCoefNumCorpo(rem(Inver(j)*T(j,1)^(i),f,alpha) mod 2):
>   od:
> od:
> return matriz:
> end proc:
```

9. Calcula a matriz de paridade com elementos em $GF(2)$

```
MatrizParidBinario:=proc()
> local i,j,s,aux,H,ListaDeVectores:=list[]:
> for i from 1 to t*p^m do
>   if (i mod p^m)=0 then
>     j:=p^m:
>     s:=iquo(i-1,p^m):
>   else
>     j:=i mod p^m:
>     s:=iquo(i,p^m):
>   fi:
>   aux:=Rem((T(j,1)^s)*Inver(1,j),f,alpha)mod 2;
>   if degree(aux)< m then
>      ListaDeVectores:=[op(ListaDeVectores), CoefficientVector(Rem((T(j,1)^s)\
        *Inver(1,j),f,alpha)mod 2+2*alpha^(m-1),alpha)mod 2]:
            #(2*alpha^(m-1))- força o comprimento do vector a m.
>   else
>      ListaDeVectores:=[op(ListaDeVectores),CoefficientVector(Rem((T(j,1)^s)\
         *Inver(1,j),f,alpha)mod 2,alpha)]:
>   fi:
> od:
> H:=convert(blockmatrix(t,p^m,ListaDeVectores),Matrix);
> end proc:
```

10. calcula peso de um vector:

```
pesoVector:=proc(v::list)::integer:
> local i,peso:=0:
> for i from 1 to nops(v) do
>   if v[i]=1 then
>      peso:=peso+1:
>   fi:
> od:
> return peso:
> end proc:
```

11. Procedimento que devolve o índice de colunas de peso 1 de uma matriz:

```
colunasPeso1:=proc(M::Matrix)
> local i,ii,indiceColunas:=[],j,ordCol:=[]:
> for i from 1 to p^m do
>   if pesoVector(convert(Vector(Column(M,i)),list))=1 then
>      indiceColunas:=[op(indiceColunas),i]:
>   fi:
> od:
> return indiceColunas:
> end proc:
```

12. Ordena índice de colunas (de peso 1) de forma a que tais colunas formem uma matriz identidade de ordem k:

```
ordInd:=proc(l)
> local j,ii,ordCol::list,vet::Matrix(1,k):
> ordCol:=[]:
> j:=1:
> while j<k+1 do
>   for ii from 1 to nops(l) do
>      if Vector(Column(G,l[ii]))(j)=1 then
>         ordCol:=[op(ordCol),l[ii]]:
>         j:=j+1:
>         break:
>      fi:
>   od:
> od:
> return ordCol:
> end proc:
```

13. Selecciona os k-bits da mensagem a partir do índice ordenado de colunas de peso 1:

```
KbitsDams:=proc(l1,l2)
> local iii,Bits,l:
> l:=convert(l2,list):
> Bits:=[]:
> for iii from 1 to k do
>    Bits:=[op(Bits),l[l1[iii]]]:
> od:
> return convert(Bits,Matrix);
> end proc:
```

14. Gera a matriz binária(aleatória), S, de ordem k e invertível:

```
 GeraMatrizKKinvert:=proc(n)
> local d,S:
> d:=0:
> while d<>1 do
>    S:=RandomMatrix(n)mod 2:
>    d:=Det(S)mod 2:
> od:
> #S:=Inverse(aux)mod 2:
> return S:
> end proc:
```

15. Gera, aleatoriamente, uma Matriz de permutação P:

```
 MatPermutP:=proc(n)
> local v,P:
> v:=RandomVector(n,generator=rand(1..n)):
> P:=CreatePermutation(v, output='Matrix'):
> return P:
> end proc:
```

16. Converte um inteiro decimal a binário de 8 bits:

```
 DecTOBin:=proc(N)
> local bin:=[],r,q,n,DifDigit,bin1::list:
> n:=N;
> q:=iquo(n,2):
> while q>0 do
>   r:=n mod 2:
>   bin:=[r,op(bin)]:
>   q:=iquo(n,2):
>   n:=q:
> od:
> DifDigit:=8-nops(bin):
> bin1:=[(seq(0,i=1..DifDigit)),op(bin)]:
> return seq(j,j in bin1):
> end proc:
```

17. Gera a tabela ASCII-256:

```
ASCII:=proc(n)
> local res:="",tab:
> tab:=convert( [$1..255], 'bytes' ):  # Tabela ASCII
> if n<>0 then
>    res:=tab[n]
> fi:
>  return res:
> end proc:
```

18. Convert binário a decimal:

```
> BinToDec:=proc(d)
> local dec:=0,numDig:=8,posDig:
> posDig:=numDig:
> while posDig>0 do
>    dec:=dec+d[posDig]*2^(numDig-posDig):
>    posDig:=posDig-1:
> end do:
> return dec:
> end proc:
```

19. Converte uma sequência binária a texto (ASCII):

```
 BinToTexto:=proc(m)
> local i,j,r,bloc,textRec,DifDigit,ultbloc,ultbloccomp,textultbloccomp:="":
> textRec:="":
> if (nops(m) mod 8)<>0 then
```

```
>    ultbloc:=m[floor(nops(m)/8)*8+1..nops(m)];
>    DifDigit:=8-nops(ultbloc):
>    ultbloccomp:=[(seq(0,i=1..DifDigit)),op(ultbloc)]:
>    textultbloccomp:=ASCII(BinToDec(ultbloccomp)):
> for i from 1 to floor(nops(m)/8)*8 by 8 do
>   bloc:=m[i..i+7]:
>   r:=ASCII(BinToDec(bloc)):
>   textRec:=cat(textRec,r);
> od:
> else:
> for i from 1 to floor(nops(m)/8)*8 by 8 do
>   bloc:=m[i..i+7]:
>   r:=ASCII(BinToDec(bloc)):
>   textRec:=cat(textRec,r);
> od:
> fi:
> return cat(textRec,textultbloccomp):
> end proc:
```

20. Procedimento que converte texto em binário

```
ConverTextoAbinario:=proc(texto::string)
> local textoAjust,textBin::list, indx::list, textoNum, i, j:
> textBin:=[]:
> textoAjust:="":
> textoNum:=map(Ord,Explode(texto)):
> for i from 1 to nops(textoNum) do
>   indx:=DecTOBin(textoNum[i]):
>   textBin:=[op(textBin),indx]:
> od:
> end proc:
```

21. Calcula a matriz geradora, G, com espaço nulo da matriz H.

```
MatrizG:=proc(H)
> local i,Gl,G:
> Gl:=Nullspace(H)mod 2:
> G:=Transpose(Matrix([seq(Gl[i],i=1..k)]))mod 2;
> return G:
> end proc:
```

22. Procedimento para cifrar mensagens:

```
CIFRAm:=proc(m,k)
> local i,j,r,bloc,M_enviada,r1,ultbloc,DifDigit,ultbloccomp,ultr:
> M_enviada:=[]:ultr:=[]:
> if (nops(m) mod k)<>0 then
>    ultbloc:=m[floor(nops(m)/k)*k+1..nops(m)]:
>    DifDigit:=k-nops(ultbloc):
>    ultbloccomp:=convert([seq(0,i=1..DifDigit),op(ultbloc)],Matrix):
>    ultr:=MatrixAdd(MatrixMatrixMultiply(ultbloccomp,Glinha),e)mod 2:
>    for i from 1 to floor(nops(m)/k)*k by k do
>       bloc:=convert(m[i..(i+k-1)],Matrix):
```

```
>       r:=MatrixAdd(MatrixMatrixMultiply(bloc,Glinha),e)mod 2:
>       r1:=(seq(j,j in r)):
>       M_enviada:=[op(M_enviada),r1];
>    od:
> else
>    for i from 1 to floor(nops(m)/k)*k by k do
>       bloc:=convert(m[i..(i+k-1)],Matrix):
>       r:=MatrixAdd(MatrixMatrixMultiply(bloc,Glinha),e)mod 2:
>       r1:=(seq(j,j in r)):
>       M_enviada:=[op(M_enviada),r1];
>    od:
> fi:
> return [op(M_enviada),seq(i,i in ultr)]:
> end proc:
```

23. Procedimento para decifrar:

```
DECIFRAr:=proc(r,n)
> local i,j,blocxP,a1,bloc,rDecifrada,codword,ms,textoClaro:="",OsBitsDams,colpivo,acolpivo:
> rDecifrada:=[]:
> acolpivo:=colunasPeso1(G):
> colpivo:=ordInd(acolpivo):
> for i from 1 to nops(r) by n do
>   bloc:=convert(r[i..i+n-1],Matrix):
>   blocxP:=MatrixMatrixMultiply(bloc,InvP)mod 2:
>  #####Imlementar aqui o processo de descodificação#####
>   codword:=LocalizaErro(t,blocxP):                #Localiza e corrige o erro
>   OsBitsDams:=KbitsDams(colpivo,codword):
>   ms:=MatrixMatrixMultiply(OsBitsDams,InvS)mod 2:
>   a1:=seq(j,j in ms):
>   rDecifrada:=[op(rDecifrada),a1]:
> od:
> textoClaro:=BinToTexto(rDecifrada,n):
> return textoClaro:
> end proc:
```

A.2 Apêndice A2

Matriz de paridade do código de Goppa $\mathcal{G} - [2^5, 12, 9]$, usado no exemplo 5.5.

$$
H = \left[\begin{array}{cccccccccccccccccccccccccccccc}
1 & 1 & \alpha^{9} & \alpha^{18} & \alpha^{18} & \alpha^{5} & \alpha^{15} & \alpha^{5} & \alpha^{29} & \alpha^{10} & \alpha^{27} & \alpha^{30} & \alpha^{17} & \alpha^{10} & \alpha^{6} & \alpha^{27} & \alpha^{3} & \alpha^{20} & \alpha^{9} & \alpha^{23} & \alpha^{30} & \alpha^{29} & \alpha^{24} & \alpha^{3} & \alpha^{17} & \alpha^{20} & \alpha^{15} & \alpha^{12} & \alpha^{24} & \alpha^{23} & \alpha^{12} & \alpha^{6} \\
0 & 1 & \alpha^{10} & \alpha^{20} & \alpha^{21} & \alpha^{9} & \alpha^{20} & \alpha^{11} & \alpha^{5} & \alpha^{18} & \alpha^{5} & \alpha^{9} & \alpha^{28} & \alpha^{22} & \alpha^{19} & \alpha^{10} & \alpha^{18} & \alpha^{5} & \alpha^{26} & \alpha^{10} & \alpha^{18} & \alpha^{18} & \alpha^{14} & \alpha^{25} & \alpha^{9} & \alpha^{13} & \alpha^{9} & \alpha^{7} & \alpha^{20} & \alpha^{20} & \alpha^{10} & \alpha^{5} \\
0 & 1 & \alpha^{11} & \alpha^{22} & \alpha^{24} & \alpha^{13} & \alpha^{25} & \alpha^{17} & \alpha^{12} & \alpha^{26} & \alpha^{14} & \alpha^{19} & \alpha^{8} & \alpha^{3} & \alpha & \alpha^{24} & \alpha^{2} & \alpha^{21} & \alpha^{12} & \alpha^{28} & \alpha^{6} & \alpha^{7} & \alpha^{4} & \alpha^{16} & \alpha & \alpha^{6} & \alpha^{3} & \alpha^{2} & \alpha^{16} & \alpha^{17} & \alpha^{8} & \alpha^{4} \\
0 & 1 & \alpha^{12} & \alpha^{24} & \alpha^{27} & \alpha^{17} & \alpha^{30} & \alpha^{23} & \alpha^{19} & \alpha^{3} & \alpha^{23} & \alpha^{29} & \alpha^{19} & \alpha^{15} & \alpha^{14} & \alpha^{7} & \alpha^{17} & \alpha^{6} & \alpha^{29} & \alpha^{15} & \alpha^{25} & \alpha^{27} & \alpha^{25} & \alpha^{7} & \alpha^{24} & \alpha^{30} & \alpha^{28} & \alpha^{28} & \alpha^{12} & \alpha^{14} & \alpha^{6} & \alpha^{3}
\end{array}\right]
$$