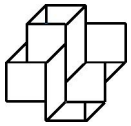

Laboratório Nacional de Computação Científica

Programa de Pós Graduação em Modelagem Computacional

Estudo Comparativo de Algoritmos de Decodificação para Códigos de Goppa Aplicados no McEliece

Por

Juan del Carmen Grados Vásquez

PETRÓPOLIS, RJ - BRASIL

ABRIL DE 2012

ESTUDO COMPARATIVO DE ALGORITMOS DE DECODIFICAÇÃO PARA CÓDIGOS DE GOPPA APLICADOS NO MCELIECE

Juan del Carmen Grados Vásquez

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM MODELAGEM COMPUTACIONAL

Aprovada por: _____

Prof. Renato Portugal, D.Sc

(Presidente)

_____

Prof. Jauvane Cavalcante de Oliveira, D.Sc.

_____

Prof. Anderson C. A. Nascimento, Ph.D.

_____

Prof. Marcelo Firer, Dr.

PETRÓPOLIS, RJ - BRASIL
ABRIL DE 2012

Grados Vásquez, Juan del Carmen

XXXX estudo comparativo de algoritmos de decodificação para códigos de goppa aplicados no mceliece / Juan del Carmen Grados Vásquez. Petropólis, RJ. : Laboratório Nacional de Computação Científica, 2012.

xx, yy p. : il.; 29 cm

Orientadore(s): Renato Portugal e Nolmar Melo

Dissertação (M.Sc.) – Laboratório Nacional de Computação Científica, 2012.

1. Códigos Corretores de Erros. 2. códigos de Goppa. 3. McEliece. 4. algoritmos de decodificação. I. Portugal, Renato. II. LNCC/MCT. III. Título.

CDD XXX.XXX

e.pí.gra.fe

Dá o primeiro passo com fé e verás que os sonhos são possíveis, o impossível é não ter sonhos. Quem não cumpre estes, não pensa em viver, quem sim, não pensa em morrer. Os sonhos abrem o coração e te inspiram, são inquebrantáveis, nunca desanimam. Os sonhos não só dão felicidade, uma vez cumpridos podem mudar a humanidade.

(July Sanchez.)

Dedicatória

Dedico esta obra a Deus e a meus pais

Ernesto Silvestre Grados Puelles e María

Magdalena Vásquez Cipirán.

Agradecimentos

Agradeço a Deus por ajudar-me em muitos passagens da minha vida, a meus pais pelo grande sacrifício que fizeram para poder chegar até aqui, a meus irmãos, Lolo, Ernesto e Vane pelo grande apoio, em todos os sentidos. A minha namorada July por seu carinho e compreensão. À família Villarroel Cruzado, a meus amigos Gustavo, Erik, Jorge, Micki, Pedro, Nils, Patty, Juan, Yessica, Cristina, Irma, Creuza. A Cristian Reyes por seu apoio desde os começos da minha carreira. Quero agradecer também ao preparatório Max Planck, a meus orientadores Renato e Nolmar por seu apoio e orientação acadêmica e a meus colegas do LNCC Alan, Geraldo, Maiana, Pablo, Guadalupe, José, Vitor, Giacomo e Rocio.

Agradeço a Nicolas Sendrier e Paulo Barreto pelas discussões que ajudaram a melhorar a qualidade deste trabalho. Meu agradecimento à instituições LNCC, CAPES e MCT que apoiaram a realização deste trabalho.

Resumo da Dissertação apresentada ao LNCC/MCT como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ESTUDO COMPARATIVO DE ALGORITMOS DE DECODIFICAÇÃO PARA CÓDIGOS DE GOPPA APLICADOS NO MCELIECE

Juan del Carmen Grados Vásquez

Abril, 2012

Orientador: Renato Portugal, D.Sc

Co-orientador: Nolmar Melo, Dr.

Em 1976, Diffie e Hellman, mudaram os rumos da criptografia criando a criptografia de chave pública ou criptografia assimétrica. Apareceram, logo, outros sistemas criptográficos assimétricos práticos, eficientes e seguros como RSA, sistemas baseados em curvas elípticas, etc. Em 1994 aparecere o algoritmo quântico de Shor, que quebra alguns destes sistemas criptográficos. No livro *"Post-Quantum Cryptography"* de Bernstein, Dahmen, e Buchmann, classifica-se os sistemas criptográficos em clássicos e pós-quânticos. Isto em função da aparente resistência destes últimos aos ataques provenientes dos algoritmos quânticos. Segundo essa classificação temos, por exemplo, que dentro dos sistemas criptográficos clássicos estão: RSA, Diffie-Hellman, sistemas cripgráficos baseados em curvas elípticas, etc, e candidatos, dentro dos sistemas criptográficos pós-quânticos estão: McEliece, *N-th degree Truncated Polynomial Ring*-NTRU, Merkle Hash tree, etc. Neste trabalho, descrevemos, implementamos e fazemos uma análise comparativa de alguns algoritmos de decodificação usados no processo de decifração do sistema criptográfico McEliece. Especificamente, a análise consiste em um estudo comparativo dos tempos de decodificação medidos em *cycles per byte*, dos algoritmos de Patterson, Barreto, Berlekamp-Massey e do algoritmo alternante apresentado por Sloane para os códi-

gos de Goppa e alternantes binários, aplicados ao McEliece, em relação ao melhor ataque de decodificação conhecido e do tamanho da chave pública deste sistema. Para o correto entendimento, os conceitos algébricos necessários, tais como os relacionados à teoria da codificação, códigos de Goppa e alternantes são apresentados. Por fim, são feitas considerações quanto sua implementação híbrida, usando o software HyMES e o CAS (*Computer Algebra System*) SAGE, cujos recursos foram usados para dar suporte ao nosso estudo e aos nossos experimentos

Abstract of Dissertation presented to LNCC/MCT as a partial fulfillment of the requirements for the degree of Master of Sciences (M.Sc.)

COMPARATIVE STUDY OF ALGORITHMS FOR DECODING GOPPA CODES APPLIED IN THE MCELIECE

Juan del Carmen Grados Vásquez

Abril, 2012

Advisor: Renato Portugal, D.Sc

Co-advisor: Nolmar Melo, Dr.

In 1976, Diffie and Hellman changed the course of the cryptography by creating the public key encryption cryptography or asymmetric cryptography. Others practical, efficient and secure asymmetric cryptosystems were developed, such as RSA, systems based on elliptic curve, etc. In 1994 Shor's quantum algorithm was published breaking some of these cryptographic systems. In the book "Post-Quantum Cryptography ", de Bernstein, Dahmen, e Buchmann, the cryptographic systems are classified as post-classical and post-quantum. It uses the apparent resistance of the latter to the attack quantum algorithms. According to this classification, we have included in the classical case. the folowing cryptographic systems: RSA, Diffie-Hellman cryptographic systems based on elliptic curves, etc. and candidates in post-quantum cryptography systems are: McEliece, N-th degree Truncated PolynomialRing – NTRU, Merkle Hash tree, etc. In this work we describe, implement and make a comparative analysis of some decoding algorithms used in the process of decryption of the McEliece cryptographic system. Specifically, the analysis consists of a comparative study of the decoding time measured in cycles per byte, of the algorithms of Patterson, Barreto, Berlekamp-Massey and the alternate algorithm presented by Sloane for the Goppa and the alternate codes applied to the McEliece related to the best decoding attack known and its public

key of this system size. To help to understand the results, required the algebraic concepts, such as those related to coding theory, and Goppa codes and alternating are described. At last, we discuss a hybrid implementation, using the software HyMES and the CAS (Computer Algebra System) SAGE, the resources were used to support our study and our experiments.

Sumário

Lista de Figuras

Figura

Lista de Tabelas

Tabela

Lista de Siglas e Abreviaturas

- det(A): determinante da matriz A.
- $\mathbb{F}_{p^n}$: Corpo Finito de característica p com p^n elementos.
- deg($p(X)$): grau do polinômio $p(X)$.
- d: distância mínima do código.
- $w(x)$: peso do vetor $x \in K^n$.
- w: peso mínimo do código.
- $d(x, y)$: distância de Hamming de x a y.
- $mdc(a, b)$: máximo divisor comúm de a e b; $a, b \in \mathbb{Z}$
- coeff($p(X)$): coeficiente do polinômio $p(X)$.
- $\langle G \rangle$: Espaço gerado pelas linhas da matriz G.
- [ALT]: Algoritmo Alternante.
- [BAR]: Algoritmo de Barreto.
- [BMA]: Algoritmo de Berlekamp-Massey.
- [PAT]: Algoritmo de Patterson.
- (bwf): Logaritmo em base 2 do (*Binary Work Factor*) para o Algoritmo Canteaut-Chabaud.

Capítulo 1

Introdução

Os métodos criptográficos podem ser divididos em duas categorias: criptografia de chave pública e criptografia de chave privada. A criptografia de chave privada utiliza a mesma chave tanto para a cifração como para a decifração, mas tem como desvantagem a necessidade de um meio seguro para compartilhar a chave entre entidades que precisem ter uma comunicação criptografada. A criptografia de chave pública foi inventada por Diffie e Hellman em 1976, e é uma solução para o problema mencionado acima, pois cada entidade, participante na comunicação criptografada, mantém duas chaves: uma pública, que pode ser divulgada livremente, e outra privada, que deve ser mantida em segredo pelo seu dono. As mensagens codificadas com a chave pública só podem ser decodificadas com a chave privada correspondente.

Dentro dos sistemas criptográficos assimétricos, ou de chave pública, práticos, eficientes e seguros temos: RSA, criptografia baseada em curvas Elípticas, Diffie-Hellman, etc. A segurança está ameaçada, devido ao algoritmo quântico desenvolvido por Peter Shor em 1994, que resolve o problema da fatoração em primos em tempo polinomial. No ano 1968, McEliece criou um sistema criptográfico randômico, que não foi posto em prática na época, porque o tamanho de suas chaves era consideravelmente grande. Atualmente, este sistema criptográfico tem tido muita aceitação por parte da comunidade de pesquisadores em criptografia, devido à sua resistência ao ataque proveniente de algoritmos quânticos, como o algoritmo

de Shor. A redução do tamanho das chaves é um tema de pesquisa atual como apresentado em Misoczki e Barreto (2009). Este sistema criptográfico tem como vantagem que, seus algoritmos são mais rápidos, no que se refere à complexidade de tempo de cifração e decifração, em comparação com o sistema consolidado RSA. Assim, por exemplo, num código de Goppa de comprimento n, a complexidade de tempo de cifração e decifração é de $O(n^2)$. Já o sistema RSA tem complexidade de tempo, de $O(m^3)$ passos (onde m é o numero de bits do texto plano), Misoczki e Barreto (2009). Os algoritmos usados no processo de decifração baseiam-se nos algoritmos de decodificação para códigos de Goppa ou alternantes binários, onde o passo principal destes algoritmos é a solução de uma equação de congruência, chamada "equação chave". Estes algoritmos são rápidos e ajudam para que o processo de decifração, do sistema McEliece seja também.

Este trabalho está focado na análise dos algoritmos de decodificação usados no processo de decifração do sistema proposto por McEliece. Apresentaremos um estudo dos conceitos algébricos, tais como os relacionados à teoria da codificação, códigos GRS, alternantes e de Goppa. Para comparar os algoritmos, analisamos também os aspectos de segurança da mensagem e da estrutura do sistema criptográfico de McEliece. Apresentamos assim os principais algoritmos de ataque ao sistema McEliece: Lee e Brickell (1989), Leon (1988), Stern (1989) e Canteaut e Chabaud (1998).

Nossa contribuição consiste em um estudo experimental comparativo de alguns algoritmos de decodificação do código de Goppa e alternante binários. A eficiência dos algoritmos é medida através do parâmetro ciclos-de-cpu por byte (*cpu-cycles per byte*). Os algoritmos de decodificação analisados são Patterson (1974), Barreto et al. (2011), Berlekamp-Massey (1969) e o algoritmo alternante (MacWilliams e Sloane, 1997, cap. 12.9). Variamos os parâmetros (n, δ) (n é comprimento do código e δ o grau do polinômio de Goppa) do código de Goppa e comparamos com um dos melhores ataques conhecidos, Canteaut e Chabaud (1998). Os resultados podem ser usados na escolha dos algoritmos de decifração

do sistema McEliece e no esquema de assinatura apresentado por Courtois Courtois et al. (2001). Neste último caso, nossa análise tem relevância, pois a complexidade computacional no algoritmo de assinatura é da ordem $O(\delta!)$.

Este texto está organizado da seguinte forma. No Capítulo 2, apresentamos conceitos básicos relacionados aos códigos lineares, códigos GRS e alternantes. No Capítulo 3, são descritos os algoritmos de decodificação para códigos de Goppa e alternantes binários, já mencionados, além de algoritmos auxiliares usados por estes. No Capítulo 4, são apresentados os sistemas criptográficos RSA, McEliece e Niederreiter, com seus aspectos de segurança. O sistema RSA é apresentado para motivar parte do o nosso trabalho. No Capítulo 5, descrevemos a implementação dos algoritmos apresentados no Capitulo 3. Esta implementação teve suporte no CAS (*Computer Algebra System*) SAGE e no software HyMES, apresentado por Biswas (2010). Apresentamos, também, no Capitulo 5 os experimentos computacionais realizados. Por último no Capítulo 6 apresentamos as ideias conclusivas de nosso trabalho.

Capítulo 2

Códigos Lineares

Um código é um conjunto finito, C, de mensagens denominadas palavras códigos, as quais tem seus símbolos num alfabeto K e são transmitidas entre duas entidades num canal de comunicação. Um código corretor de erros é um código, tal que suas palavras código formam um subconjunto próprio de K^n, para algum $n \in \mathbb{N}$. Estes tipos de códigos tornam possíveis, muitas vezes, a correção e detecção de erros durante a transmissão de mensagens num canal de comunicação com ruido, como mostrado na Figura 2.1. Dentro dos códigos corretores de erro estão os códigos não lineares (e.g. Hadamard e Reed-Muller, (veja Klima et al., 2000)), os quais não são bons em aplicações práticas. Uma das desvantagens mais notórias, em relação aos códigos lineares (Definição 2.2.1), é o alto custo computacional para encontrar a distância mínima de tal código (parâmetro muito importante dentro da teoria de códigos como veremos na seção 2.2.2). Este capítulo está divido em três partes, na Seção 2.1 apresentaremos alguns conceitos preliminares , na Seção 2.2 trataremos sobre os códigos lineares em si e na Seção 2.5 trataremos sobre os códigos códigos de Goppa, os quais são de nosso interesse.

2.1 Preliminares

Antes de começar com o desenvolvimento dos códigos lineares é preciso conhecer alguns conceitos que serão usados ao longo deste capítulo.

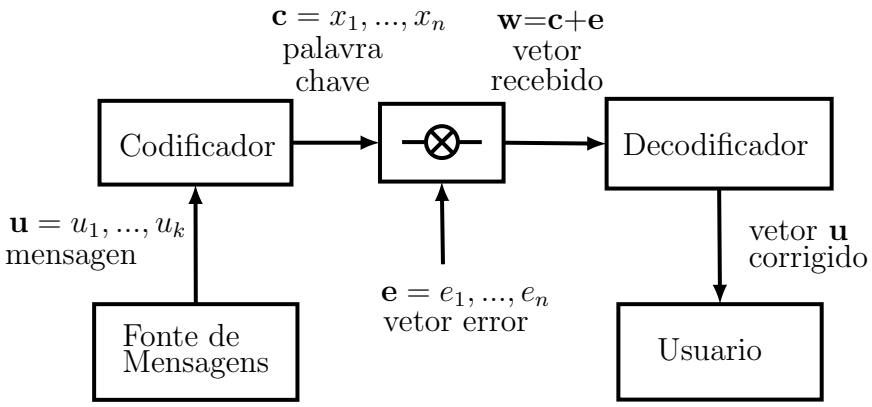

Figura 2.1: Sistema Básico de Comunicação.

Definição 2.1.1. *Sejam dois elementos quaisquer $\boldsymbol{u}$ e $\boldsymbol{v} \in K^n$, a distância de Hamming entre $\boldsymbol{u}$ e $\boldsymbol{v}$, $d(\boldsymbol{u}, \boldsymbol{v})$, é definida como o número de componentes distintas entre $\boldsymbol{u}$ e $\boldsymbol{v}$.*

$$d(\boldsymbol{u}, \boldsymbol{v}) = \#\{i; u_i \neq v_i\}$$

Exemplo. Sejam $K = \mathbb{F}_2$ e $n = 3$. Dados os vetores $\mathbf{u} = (0, 1, 1)$ e $\mathbf{v} = (0, 1, 0)$, então $d(\boldsymbol{u}, \mathbf{v}) = 1$.

Pode-se provar que a distância de Hamming induz uma métrica (veja Hefez e Villela, 2008, cap. 1) fazendo $|\mathbf{v}| = d(0, \mathbf{v})$ e por isso muitos autores a chamam de métrica de Hamming.

Definição 2.1.2. *Sejam dois espaços vetoriais U e V, dizemos que uma transformação linear $T : U \longrightarrow V$ é uma isometria se, e somente se, preserva a métrica. Isto é $|u| = |Tu|$, onde $u \in U$ e $|\ \boldsymbol{.}\ |$ é uma métrica.*

Definição 2.1.3. *A distância mínima de um código C é o inteiro*

$$d := min\{d(\boldsymbol{u}, \boldsymbol{v}); \boldsymbol{u}, \boldsymbol{v} \in C, \boldsymbol{u} \neq \boldsymbol{v}\}.$$

Definição 2.1.4. *Sejam U e V espaços vetoriais, e seja $T : U \to V$ uma transformação linear, então, define-se o espaço nulo da transformação T como o conjunto*

$$N(T) = \{\boldsymbol{u} \in U; T\boldsymbol{u} = \boldsymbol{0}\}.$$

2.2 Códigos Lineares

Se o conjunto K^n indicado acima é um espaço vetorial de dimensão $n \in \mathbb{N}$ e C é um subespaço então o código chama-se de código linear.

Definição 2.2.1. *Um código $C \subset K^n$ será chamado de código linear se for um subespaço vetorial de K^n, sendo este código descrito por seus parâmetros $[n, k]$, onde k é a dimensão de C.*

O K^n denota um espaço vetorial de dimensão n sobre o corpo finito K. É fácil demonstrar que C é um espaço vetorial, e que, se o número de elementos do corpo K é q e a dimensão do espaço vetorial C é k, então o número de elementos de C é q^k.

Definição 2.2.2. *Dado $\boldsymbol{c} \in C$, define-se o peso de Hamming, $w(\boldsymbol{c})$, como o número de coordenadas diferentes de zero em $\boldsymbol{c}$, ou seja $w(\boldsymbol{c}) = d(\boldsymbol{0}, \boldsymbol{c}) = |\boldsymbol{c}|$.*

Teorema 2.2.1. *Dado C um código linear e supondo $w = min\{w(\boldsymbol{c}); \boldsymbol{c} \in C, \boldsymbol{c} \neq \boldsymbol{0}\}$, então $w = d$.*

Demonstração. Seja $\mathbf{c} \in C$ tal que $w = w(\mathbf{c})$. É claro que existem $\mathbf{x}$, $\mathbf{y} \in C$ tais que $\mathbf{x} - \mathbf{y} = \mathbf{c}$, e, portanto, pela Definição 2.1.1, tem-se que $w = w(\mathbf{c}) = d(\mathbf{x}, \mathbf{y})$. Sejam agora $\mathbf{x'}$ e $\mathbf{y'} \in C$, tal que $d = d(\mathbf{x'}, \mathbf{y'})$, e pela Definição 2.2.2 $d = w(\mathbf{x'} - \mathbf{y'})$. Agora, usando a demonstração por absurdo, suponha $w > d = w(\mathbf{x'} - \mathbf{y'})$ então w não seria o $\min\{w(\mathbf{c'}); \mathbf{c'} \in C, \mathbf{c'} \neq 0\}$. De igual forma, se $d(\mathbf{x}, \mathbf{y}) = w < d$ então o d não seria o $\min\{d(\mathbf{u}, \mathbf{v}); \mathbf{u}, \mathbf{v} \in C, \mathbf{u} \neq \mathbf{v}\}$, portanto $d = w$. □

2.2.1 Matriz Geradora e Matriz de Teste de Paridade

Suponha que a base $\{v_1, ..., v_k\}$ para o K-espaço vetorial C (código linear) forma uma matriz G, com posto-linha completa, de dimensões (n, k), onde v_i para

$1 \leq i \leq k$, são as linhas de G, da seguinte forma

$$G = \begin{bmatrix} v_{1,1} & \dots & v_{1,n} \\ v_{2,1} & \dots & v_{1,n} \\ \vdots & \dots & \vdots \\ v_{k,1} & \dots & v_{k,n} \end{bmatrix}.$$

Seja T a transformação linear induzida pela matriz G como segue:

$$T : K^k \longrightarrow K^n$$
$$\mathbf{x} \longrightarrow \mathbf{x}G.$$

Portanto $C = \{\mathbf{x}G; \mathbf{x} \in K^k\}$. Esta matriz é importante porque pode-se descrever o código linear C de dimensão k mediante uma transformação linear induzida pela matriz G. Esta matriz é normalmente conhecida como a *matriz geradora do código.*

Pelo fato de um espaço vetorial poder ter mais de uma base, então pode-se obter uma outra matriz geradora G' para C, aplicando as seguintes operações, sequencialmente, sobre a matriz G:

- Permutação de duas linhas.
- Multiplicação de uma linha por um escalar não nulo.
- Adição de um múltiplo escalar de uma linha a outra.
- Permutação de duas colunas.
- Multiplicação de uma coluna por um escalar não nulo.

Definição 2.2.3. *Seja K um corpo finito. Dois códigos lineares C e C' são equivalentes se existe uma isometria linear $T : K^n \longrightarrow K^n$ tal que $T(C) = C'$.*

Definição 2.2.4. *Diremos que uma matriz geradora G está na forma padrão se tivermos:*

$$G = (I|A),$$

onde I é uma matriz identidade $k \times k$ e A, uma matriz $k \times (n-k)$.

A matriz G na forma padrão vai nos ajudar no processo de decodificação que descreveremos mais na frente. Esta forma pode ser obtida através das operações sobre matrizes descritas acima. Tal processo pode ser enunciado com o teorema a seguir.

Teorema 2.2.2 (Hefez e Villela (2008))**.** *Dado um código C, existe um código equivalente C' com matriz geradora na forma padrão.(veja demostração em Hefez e Villela, 2008).*

Seja a matriz H, tal que suas linhas sejam uma base para o espaço nulo da transformação linear induzida pela matriz G. A matriz H é conhecida como a matriz de teste de paridade do código C (veja Klima et al., 2000) e pode-se demonstrar que tem dimensões $(n-k, n)$ (veja corolário da Proposição 3, seção 5.3 em Hefez e Villela, 2008). Com esta explicação pode-se definir um código linear C, usando a matriz de teste de paridade, como a seguir.

Proposição 2.2.3. *Seja C um código linear e suponhamos que H seja a matriz de teste de paridade de C. Temos então que :*

$$\boldsymbol{v} \in C \Longleftrightarrow H\boldsymbol{v}^T = \boldsymbol{0}.$$

Na Figura 2.1, suponha que o vetor w chega ao decodificador, então para determinar se $w \in C$, basta verificar que $H(w^T) = \mathbf{0}$. A expressão $H(w^T)$ é conhecida como a *síndrome* de w.

Teorema 2.2.4 (Klima et al. (2000))**.** *Seja C um código linear com matriz de paridade H e s o mínimo número de colunas linearmente dependentes em H, então s é a distância mínima do código.*

Demonstração. Seja $d = w$ a distância mínima do código C e seja, a menos de reordenação, o vetor $\mathbf{x} = (x_1, ...x_s, 0, .., 0) \in C$, com $x_i \neq 0$, sendo $1 \leq i \leq s$, então $w(\mathbf{x}) = s$. Suponha que $s < d$ então $w(\mathbf{x}) < w$, portanto w não seria a distância

mínima do código. Seja $(y_1, ...y_d, 0, 0, ..., 0) = \mathbf{y} \in C$, com $y_i \neq 0$, sendo $1 \leq i \leq d$, tal que $w(\mathbf{y}) = d$, e seja $H = (\mathbf{h}_1, ..., \mathbf{h}_n)$. Fazendo $H\mathbf{x}$ temos $\mathbf{0} = x_1\mathbf{h}_1 + ... + x_s\mathbf{h}_s$. Suponha agora que $s > d$, então já que $H\mathbf{y} = y_1\mathbf{h}_1 + ... + y_d\mathbf{h}_d = \mathbf{0}$ existem d colunas linearmente dependentes em H, não se respeitando, portanto, a hipótese do teorema. Logo, conclui-se que $s = d$. □

2.2.2 Decodificação

Sendo C um código corretor de erros, a decodificação é o processo pelo qual uma palavra $c \in C$, recebida pelo decodificador num sistema de comunicação com ruido, é convertida numa palavra chave do código C. Explicaremos a seguir dois métodos para decodificar palavras de um código linear nestes tipos de sistemas.

(i) Politica da Máxima Proximidade.

Este método decodifica um vetor recebido numa palavra código, tendo em conta o menor número de posições nas quais estes vetores diferem. Seja $\mathbf{r}$ o vetor recebido no decodificador. Então, o vetor $\mathbf{x}$ tal que: $d(r) = \min\{d(\mathbf{r}, \mathbf{x}); \mathbf{x} \in C\}$, será o vetor decodificado. Este método é muito limitado como veremos num dos passos na demostração da Proposição a seguir, 2.2.5.

Proposição 2.2.5. *Um código com distância mínima d pode corrigir até* $\lfloor (d-1)/2 \rfloor$ *erros. E detectar até* $d-1$ *erros.*

Demonstração. Suponha que d é ímpar, então terá a forma $d = 2t + 1$ com $t \in \mathbb{N}$. Seja a esfera S_x de raio r com centro em $x \in K^n$ (K corpo finito), tal que $d(\mathbf{x}, \mathbf{y}) \leq r$, onde $y \in K^n$. Da Figura 2.2, suponha que $r \leq t$ (isto para que as esferas não se sobreponham), então se fosse transmitido o vetor $\mathbf{u}$ num canal de comunicação com ruido e chegasse, ao decodificador, o vetor $\mathbf{a} \in K^n$, tal que $d(\mathbf{a}, \mathbf{u}) \leq t$, então, pela *politica da máxima proximidade*, $\mathbf{a}$ pode ser decodificado em $\mathbf{u}$. Se, $t < d(\mathbf{a}, \mathbf{u}) < d$, da mesma forma, pela *politica da máxima proximidade*, $\mathbf{a}$ seria decodificado em $\mathbf{v}$, o que estaria errado. Portanto, se d é impar o código pode corrigir até $\lfloor (d-1)/2 \rfloor$ erros e detetar até $2t = d - 1$ erros. Vejamos a

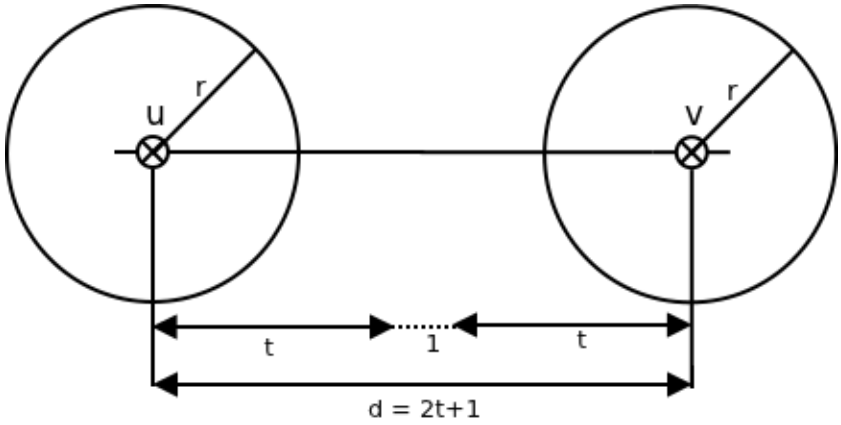

Figura 2.2: Método da política máxima proximidade, caso impar.

segunda parte da demostração, supondo que $d = 2t$, como mostrado na Figura 2.3. De igual maneira como foi explicado acima, suponha que $r < t$ e chegou o vetor $\mathbf{a}$ ao decodificador tal que $d(\mathbf{a}, \mathbf{u}) \leq t - 1 = ((2t-2)/2) = (d-2)/2 = \lfloor (d-1)/2 \rfloor$, então $\mathbf{a}$ pode ser decodificado em $\mathbf{u}$ pelo *politica da máxima proximidade*. Se $d > d(\mathbf{a}, \mathbf{u}) \geq t = 2t/2 = d/2$, então $\mathbf{a} \in S_u$ e também $\mathbf{a} \in S_v$. Logo não se poderia afirmar que $\mathbf{a}$ pode ser decodificado em $\mathbf{u}$ nem $\mathbf{v}$, contudo pode-se afirmar que ocorreu algum erro no vetor $\mathbf{a}$, ficando, assim, a segunda parte da proposição demonstrada. □

(ii) Usando a Síndrome.

A seguir veremos um resultado que relaciona a síndrome do vetor que chega ao decodificador e o erro que ele possui. Este resultado será usado no desenvolvimento deste método.

Proposição 2.2.6. *Seja C um código linear em K^n com capacidade de correção t. Se $\boldsymbol{r} \in K^n$ e $\boldsymbol{c} \in C$ são tais que $d(\boldsymbol{c}, \boldsymbol{r}) \leq t$, então existe um único vetor $\boldsymbol{e}$ com $w(\boldsymbol{e}) \leq t$ cuja síndrome é igual à síndrome de $\boldsymbol{r}$.*

Demonstração. Seja $\mathbf{c} \in C$ o vetor transmitido num sistema de comunicação com ruido, onde é adicionado um vetor erro $\mathbf{e} = (e_1, ..., e_n)$ tal que $w(\mathbf{e}) \leq t$. Portanto, $\mathbf{r}$, $\mathbf{e}$ e $\mathbf{c}$ são tais que $\mathbf{r} = \mathbf{c} + \mathbf{e} \iff \mathbf{r} - \mathbf{c} = \mathbf{e}$, logo $w(\mathbf{e}) = d(\mathbf{c}, \mathbf{r}) \leq t$. Demonstraremos a seguir que o erro $\mathbf{e}$ é único. Para isto, suponha que existe um outro vetor $\mathbf{e'} = (e'_1, ..., e'_n)$ com a mesma síndrome que $\mathbf{r}$, tal que $w(\mathbf{e'}) \leq t$, então se

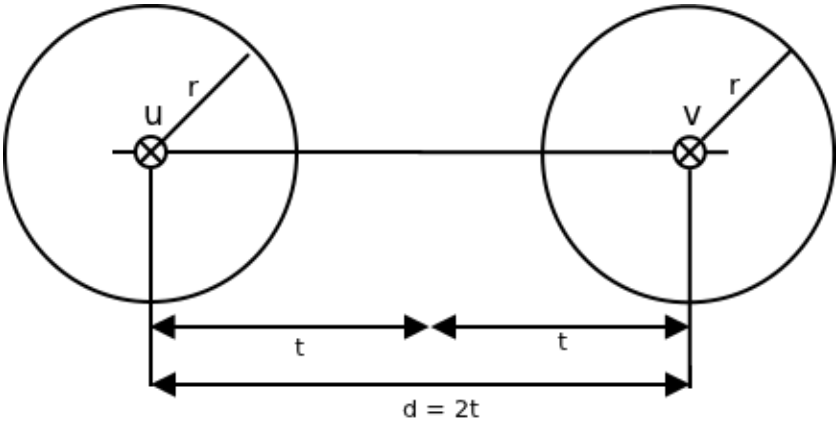

Figura 2.3: Método da política da máxima proximidade, caso par.

$H = (\mathbf{h}_1, ..., \mathbf{h}_n)$ é uma matriz de teste de paridade de C, onde h_i são as colunas de H, temos que $H\mathbf{e} = H\mathbf{e'} \Longrightarrow (e_1 - e'_1)\mathbf{h}_1 + ... + (e_n - e'_n)\mathbf{h}_n = \mathbf{0}$. Mas, nesta expressão, pelo fato que $w(\mathbf{e}) + w(\mathbf{e'}) \leq 2t$, no pior dos casos, deveria-se ter no máximo $d - 1$ expressões diferentes de zero da forma $(e_x - 0)$, onde e_x pode ter valores e_i ou e'_i para $i = 1...n$. Logo, pela Proposição 2.2.4, todos os e_x seriam iguais ao zero, concluindo-se que $e_i = e'_i$ para todo i. □

Definição 2.2.5 (Classe Lateral)**.** *Dado um código linear $C \subset K^n$, define-se como classe lateral de v segundo C todos os vetores que pertencem a K^n tais que estes diferem de v numa palavra chave do código. Isto é $\boldsymbol{v} + C = \boldsymbol{w} + C \Longleftrightarrow \boldsymbol{v}\text{-}\boldsymbol{w} \in C$*

Suponha que num canal de comunicação, como na Figura 2.1, é transmitida a palavra $\mathbf{c}$ do código e que fosse recebida no decodificador a palavra $\mathbf{r} = \mathbf{c} + \mathbf{e}$. Dado que $\mathbf{r} - \mathbf{e} = \mathbf{c}$ então $\mathbf{r}$ e $\mathbf{e}$ pertencem à mesma *classe lateral*. É fácil provar também que a união de todas as classes laterais num código C é igual a K^n. Numa *classe lateral*, define-se como elemento líder da classe aquele que tiver o menor peso.

Proposição 2.2.7. *Seja C um código linear em K^n com distância mínima d. Se $\boldsymbol{u} \in K^n$ é tal que:*

$$w(\boldsymbol{u}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor,$$

então $\boldsymbol{u}$ é o único elemento líder da classe lateral.

Demonstração. Suponha que existem $\mathbf{u}$ e $\mathbf{v} \in K^n$ tais que $w(\mathbf{u}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ e $w(\mathbf{v}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor$. Pela Definição 2.1.1 tem-se que $w(\mathbf{u}-\mathbf{v}) = d(\mathbf{u},\mathbf{v})$, $w(\mathbf{u}-\mathbf{0}) = d(\mathbf{u},\mathbf{0})$ e $w(\mathbf{v}-\mathbf{0}) = d(\mathbf{v},\mathbf{0})$. Devido ao fato que a distância de Hamming é uma métrica, então $w(\mathbf{u}-\mathbf{v}) = d(\mathbf{u},\mathbf{v}) \leq d(\mathbf{u},\mathbf{0}) + d(\mathbf{0},\mathbf{v}) \leq 2\left\lfloor \frac{d-1}{2} \right\rfloor \leq d-1$. Dado que a distância mínima do código é d então $\mathbf{u}-\mathbf{v} = \mathbf{0} \Longrightarrow \mathbf{u} = \mathbf{v}$. □

Teorema 2.2.8 (Klima et al. (2000))**.** *Dado C um código linear com matriz de teste de paridade H. Então $\boldsymbol{u}, \boldsymbol{v} \in K^n$ estão na mesma classe lateral se, e somente se, $H\boldsymbol{u}^T = H\boldsymbol{v}^T$.*

Demonstração. Sejam $\mathbf{u}, \mathbf{v} \in K^n$ então pela Definição 2.2.5 temos que $\mathbf{u}-\mathbf{v} = \mathbf{c} \in C$, multiplicando nos dois lados pela matriz de teste de paridade H e tendo em conta que o código é linear, tem-se que $H\mathbf{u}^T - H\mathbf{v}^T = H\mathbf{c}^T$, mas pela Proposição 2.2.3 $H\mathbf{u}^T - H\mathbf{v}^T = 0 \Longleftrightarrow H\mathbf{u}^T = H\mathbf{v}^T$. □

Suponha que uma palavra código $\mathbf{c}$ do código linear C, que tem distância mínima d, é transmitida num canal de comunicação com ruido e chega ao decodificador o vetor $\mathbf{r} = \mathbf{c} + \mathbf{e}$. Dado que $\mathbf{e}$ e $\mathbf{r}$ diferem em $\mathbf{c}$ então estão na mesma *classe lateral*, ou já que $H\mathbf{e}^T = H\mathbf{r}^T$ e Teorema 2.2.8. Se $w(\mathbf{e}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ então pela Proposição 2.2.7 $\mathbf{e}$ é o único elemento líder da *classe lateral*, e pela Proposição 2.2.6 $\mathbf{e}$ é o <u>único</u> vetor tal que $H\mathbf{e}^T = H\mathbf{r}^T$. Logo, se existisse alguma forma de saber quem é esse único elemento líder da *classe lateral* o problema de decodificar, usando a síndrome, estaria resolvido. Para isto, vamos construir uma tabela de duas colunas, onde os elementos da primeira coluna são os vetores $\mathbf{l} \in C$, tais que tenham peso menor ou igual a $\lfloor \frac{d-1}{2} \rfloor$ (líderes de cada *classe lateral*). Enquanto isto, os elementos da segunda coluna serão as respectivas síndromes dos vetores mencionados. Logo, como já conhecemos que $H\mathbf{r}^T = H\mathbf{e}^T$, $\mathbf{e}$ será algum $\mathbf{l}$ tal que $H\mathbf{r}^T = H\mathbf{l}^T$. Portanto $\mathbf{c} = \mathbf{r}-\mathbf{l}$ e o problema de decodificação usando este método fica resolvido.

Em 1981, V. D. Goppa descobriu uma construção de códigos lineares baseados em curvas algébricas definidas sobre corpos finitos. Esses códigos são conhecidos como

códigos de Goppa. Na seção 2.5 apresentamos uma descrição destes códigos. Para isso consideraremos primeiro os códigos generalizados de Reed Solomon (GRS), Seção 2.3, e os códigos Alternantes, Seção 2.4.

2.3 Códigos GRS

Os códigos Reed-Solomon foram introduzidos em 1960 por I. Reed e G. Solomon. Nesta seção definiremos a generalização destes.

Definição 2.3.1. *Seja $F = \mathbb{F}_q$ um corpo finito, onde $q \in \mathbb{N}$ é um número primo. Dado $v_1, ..., v_n \in F$ elementos diferentes de zero e $\alpha_1, ...\alpha_n \in F$. Seja $\boldsymbol{v} = (v_1, ..., v_n)$ e $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)$. Para $0 \leq k \leq n$ define-se os códigos generalizados de Reed Salomon como:*

$$GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{v}) = \{(v_1 f(\alpha_1), v_2 f(\alpha_2), ..., v_n f(\alpha_n)) | f(X) \in F[X]_k\}$$

com distância mínima $k + 1$.

Na definição anterior $F[X]_k$ representa o conjunto de polinômios no anel $F[X]$, de grau menor que k. É fácil ver que este conjunto é um espaço vetorial de dimensão k sobre o corpo F.

Um código GRS é um código linear com parâmetros $[n, k]$ sobre o corpo F, com a seguinte matriz de teste de paridade.

$$H = \begin{bmatrix} v_1 & v_2 & \dots & v_n \\ v_1\alpha_1 & v_1\alpha_2 & \dots & v_n\alpha_n \\ \vdots & \dots & \dots & \vdots \\ v_1\alpha_1^{k-1} & v_2\alpha_2^{k-1} & \dots & v_n\alpha_n^{k-1} \end{bmatrix}$$

2.4 Códigos Alternantes

Os códigos alternantes são definidos fazendo uma restrição nos códigos GRS. A ideá desta restrição é selecionar a parcela do código GRS que também esteja num subcorpo do corpo original. Esta restrição é conhecida como subcódigo de subcorpo.

Definição 2.4.1. *Considere o corpo finito $\mathbb{F}_{q^m}$ extensão de $\mathbb{F}_q$. Dado $C \subseteq (\mathbb{F}_{q^m})^n$ um código sobre $\mathbb{F}_{q^m}$.*

$$C|_{\mathbb{F}_q} := C \cap \mathbb{F}_q^n$$

é chamado de subcódigo de subcorpo de C.

Definição 2.4.2. *Seja $\mathbb{F}_q$ um subcorpo de $\mathbb{F}_{q^m}$, então o código alternante de $GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{v})$ no subcorpo $\mathbb{F}_q$ é $GRS_{n,k}(\boldsymbol{\alpha}, \boldsymbol{v}) \cap \mathbb{F}_q$.*

2.5 Códigos de Goppa

Dentre os códigos alternantes, como apresentados em MacWilliams e Sloane (1997), existe uma subclasse de códigos assintoticamente bons, no sentido a que atingem a cota de Gilbert-Varshamov, descoberto por V. D. Goppa (1970). Estes códigos são interessantes, também, devido à facilidade para o cálculo de sua distância mínima. A criação destes códigos são motivados pelos códigos BCH, e da mesma forma podem ser descritos em função de um polinômio gerador $g(X)$, denominado polinômio de Goppa, e de um conjunto de elementos, L, que está contido num corpo finito onde o código de Goppa tem seus símbolos.(veja seção 10.5 Hefez e Villela, 2008)

Definição 2.5.1. *Seja F um corpo finito, extensão de um corpo K. Sejam $g(X) \in F[X]$ e $L = \{\alpha_0, \alpha_1, ..., \alpha_{n-1}\} \subset F$, onde os α_i são dois a dois distintos e tais que $g(\alpha_i) \neq 0$ para $i = 0, ..., n-1$. Define-se como código de Goppa o conjunto*

$$\Gamma_K(L, g) = \left\{ (c_0, ..., c_{n-1}) \in K^n; \sum_{i=0}^{n-1} c_i g(\alpha_i)^{-1} = 0 \right\}.$$

.

Pode-se demonstrar facilmente que o conjunto acima é um código linear, e é dito: *código de Goppa clássico sobre o corpo K associado ao conjunto L e ao polinômio $g(X)$.*

Devido a fato que os códigos de Goppa são assintoticamente bons, pode-se implementar códigos de Goppa longos, os quais são de muita utilidade nas aplicações práticas, dado que estas usam códigos de Goppa com uma grande quantidade de símbolos.

2.5.1 Matriz de Teste de Paridade

Abaixo apresentaremos uma das formas que existem para a construção da matriz de teste de paridade para os códigos de Goppa.

Seja

$$g(X) = \sum_{j=0}^{\delta} g_j X^j, \text{ com } g_\delta \neq 0 \in F,$$

então

$$\frac{g(X) - g(\alpha)}{X - \alpha} = \sum_{j=0}^{\delta} g_j \cdot \frac{X^j - \alpha^j}{X - \alpha}.$$

Portanto, $(c_0, c_1, ...c_{n-1}) \in \Gamma_K(L, g)$ se e somente se

$$\sum_{i=0}^{n-1} (g(\alpha_i)^{-1} \sum_{j=t+1}^{\delta} g_j \alpha_i^{j-1-t}) c_i = 0,\ 0 \leq t \leq \delta - 1.$$

Da expressão acima temos que $c \in \Gamma_K(L, g) \iff Bc^t = 0$, onde:

$$B = \begin{bmatrix} g(\alpha_0)^{-1} g_\delta & \dots & g(\alpha_{n-1})^{-1} g_\delta \\ g(\alpha_0)^{-1}(g_{\delta-1} + g_\delta \cdot \alpha_0) & \dots & g(\alpha_{n-1})^{-1}(g_{\delta-1} + g_\delta \cdot \alpha_{n-1}) \\ \vdots & \dots & \vdots \\ g(\alpha_0)^{-1} \sum_{j=1}^{\delta} g_j \alpha_0^{j-1} & \dots & g(\alpha_{n-1})^{-1} \sum_{j=1}^{\delta} g_j \alpha_{n-1}^{j-1} \end{bmatrix}$$

e

$$c^T = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{\delta-1} \end{bmatrix}.$$

Fazendo $l_j \longleftarrow l_j - (g_\delta \sum_{i=1}^{j-1} g_{\delta-i}\alpha_0^{j-i-1})l_1$ para $1 \leq j \leq \delta - 1$, onde l_j são as linhas de B, obtemos a matriz

$$\hat{H} = \begin{bmatrix} g(\alpha_0)^{-1} & \dots & g(\alpha_{n-1})^{-1} \\ g(\alpha_0)^{-1}\alpha_0 & \dots & g(\alpha_{n-1})^{-1}\alpha_{n-1} \\ \vdots & \dots & \vdots \\ g(\alpha_0)^{-1}\alpha_0^{\delta-1} & \dots & g(\alpha_{n-1})^{-1}\alpha_{n-1}^{\delta-1} \end{bmatrix}.$$

Portanto $c \in \Gamma_K(L, g) \Longleftrightarrow \hat{H}c^t = 0$. Observe que as entradas desta matriz de teste de paridade $\hat{H}$ estão no corpo F. Pode-se considerar F como um K-espaço vetorial de dimensão m. Assim podemos escrever cada entrada da matriz $\hat{H}$ como um vector de comprimento m com coordenadas em K, obtendo uma matriz H' de dimensões $(m * \delta)$ x n com coeficientes em K, sendo, portanto, $c \in \Gamma_K(L, g) \Longleftrightarrow H'c^t = 0$.

A distância mínima d deste código é pelo menos δ. Para demonstrar isto vemos que, criando a matriz $\hat{H'}$ a partir de quaisquer $\delta - 1$ colunas de $\hat{H}$, temos:

$$\det(\hat{H'}) = \prod_{i=0}^{\delta-1} g_(\alpha_i)^{-1} \cdot \det(\begin{bmatrix} 1 & \dots & 1 \\ \alpha_0 & \dots & \alpha_{\delta-1} \\ \vdots & \dots & \vdots \\ \alpha_0^{\delta-1} & \dots & \alpha_{\delta-1}^{\delta-1} \end{bmatrix}).$$

A matriz do lado direito da expressão acima é a matriz de Vandermonde, na qual temos que $\det(\hat{H}) \neq 0$. Logo, quaisquer $\delta - 1$ colunas de $\hat{H}$ são linearmente independentes. Este resultado e o Teorema 2.2.4 demonstram que a distância mínima deste código de Goppa é pelo menos δ.

A matriz B também pode ser descrita como o produto de três matrizes: Toepliz,T ,Vandermonde, V e uma matriz diagonal D, (Seção 2.4.2.1 Hoffmann, 2011). Assim,

$$B = T \cdot V \cdot D,$$

onde

$$T = \begin{pmatrix} g_\delta & 0 & 0 & \dots & 0 \\ g_{\delta-1} & g_\delta & 0 & \dots & 0 \\ g_{\delta-2} & g_{\delta-1} & g_\delta & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \dots & g_\delta \end{pmatrix}, \qquad V = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \dots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{\delta-1} & \alpha_2^{\delta-1} & \dots & \alpha_{n-1}^{\delta-1} \end{pmatrix}$$

$$D = \begin{pmatrix} 1/g(\alpha_0) & 0 & \dots & 0 \\ 0 & 1/g(\alpha_1) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1/g(\alpha_{n-1}) \end{pmatrix}.$$

2.5.2 Matriz Geradora do Código

A matriz geradora do código G, como já explicamos na seção 2.2.1, pode ser calculada a partir de H, computando uma base para o espaço nulo da transformação linear induzida pela matriz H. O resultado a seguir fornece uma outra maneira para definir um código de Goppa.

Proposição 2.5.1. *Sejam K e F corpos finitos, com F sendo uma extensão de K. Sejam $g(X) \in F[X]$, com $\deg(g(X)) = \delta$, e $L = \{a_0, ..., a_{n-1}\} \subset F$, onde a_i são dois a dois distintos e tais que $g(\alpha_i) \neq 0$, $i = 0, ..., n-1$. Temos que*

$$\Gamma_K(L, g) = \left\{ (c_0, ..., c_{n-1}) \in K^n; \sum_{i=0}^{n} \frac{c_i}{X - a_i} \equiv 0 \mod g(X) \right\} \qquad (2.1)$$

(veja demostração em Proposição 1 Hefez e Villela, 2008).

Para um melhor entendimento sobre a construção das matrizes G e H vejamos um exemplo de sua construção .

Exemplo. Começamos construindo uma extensão $F = \mathbb{F}_{2^3}$ de um corpo finito $K = \mathbb{F}_2$, com o anel de classes residuais $K[X]_{P(X)}$, onde $P(X) = X^3 + X + 1$ é um polinômio irreduzível. Seja o polinômio de Goppa $g(X) = X^2 + X + 1$, assim a distância mínima d é δ =deg$(g(X))$= 2. Seja a um elemento primitivo de F e seja o suporte de código $L = F \setminus \{\mathbf{0}\}$, obtemos então:

$$T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},$$

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & a & a^2 & a+1 & a^2+a & a^2+a+1 & a^2+1 \end{pmatrix},$$

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a^2+a & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a^2+a \end{pmatrix}$$

e, portanto,

$$B = \begin{pmatrix} 1 & a^2 & a^2+a & a^2 & a & a & a^2+a \\ 0 & a^2+a+1 & a+1 & a+1 & a^2+1 & a^2+a+1 & a^2+1 \end{pmatrix}.$$

Expressando cada entrada de B como um vetor de dimensão 3, obtemos:

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

Seguindo o processo do Exemplo 3.5 como apresentado em Klima et al. (2000), obtemos a matriz geradora:

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

Seja $w \in (F_q)^n$ uma palavra transmitida num canal de comunicação com ruído, e seja $\hat{H}$ a matriz de teste de paridade associada ao código $C = \Gamma_K(L, g)$. Chamaremos o vetor $\hat{H}w^t = (S_0, ..., S_{\delta-1})^T \in F$ de vetor síndrome. Com este vetor pode-se construir $S(X) = S_0X^0 + ... + S_{\delta-1}X^{\delta-1}$, o qual será denominado polinômio síndrome.

2.5.3 Códigos de Goppa Binário

Consideraremos agora uma classe especial de códigos de Goppa, chamados de códigos de Goppa binário, isto porque a característica do corpo, onde o código tem seus símbolos, é dois.

Definição 2.5.2. *A restrição $K = \mathbb{F}_2$ e $F = \mathbb{F}_{2^m}$ na Definição 2.5.1, da origem aos códigos de Goppa binário.*

Proposição 2.5.2. *(Hoffmann (2011)) Dado $\Gamma(L, g)$ um código de Goppa binário, tal que $g(X)$ é livre de quadrados, então:*

$$\Gamma(L, g) = \Gamma(L, g^2)$$

Veja que a propriedade acima faz com que a distância mínima do código seja ao menos $2 \cdot \delta$, é dizer garanta-se a correção de δ erros.

Definição 2.5.3. *Sejam m, n e δ inteiros positivos tal que $n \leq 2^m$ e $\delta < n/m$. Um código de Goppa binário é chamado* ***código de Goppa livre de quadrado*** *se o polinômio de Goppa $g(X)$ é livre de quadrados e não tem raízes em F. Se $g(X)$ for irredutível o código é chamado de* ***código de Goppa irredutível****.*

2.5.4 Equação Chave

Esta equação é uma expressão essencial para detecção e correção de erros. Surge de uma relação de congruência entre três polinômios: o polinômio síndrome, o avaliador e o localizador (veja a definição 2.5.4) . A forma de resolver esta equação, tendo em conta a quantidade de erros para detectar e corrigir, dá origem aos algoritmos de decodificação que apresentaremos no Capítulo 3. Veremos a seguir como obter o polinômio síndrome para os códigos de Goppa binário.

Suponha que num canal de comunicação é transmitido a palavra código $\mathbf{c} \in \Gamma$ e é adicionado o vetor erro $\mathbf{e} \in K^n$, então a palavra recebida é

$$\mathbf{w} = \mathbf{c} + \mathbf{e}.$$

Assim,

$$\sum_{y \in L} \frac{w_y}{X - y} = \sum_{y \in L} \frac{c_y}{X - y} + \sum_{y \in L} \frac{e_y}{X - y},$$

já que se $\mathbf{c}$ é uma palavra código então, devido a equação (2.1),

$$\sum_{y \in L} \frac{w_y}{X - y} \equiv \sum_{y \in L} \frac{e_y}{X - y} \bmod g(X).$$

Fazendo $S(X) = \sum_{y \in L} \frac{e_y}{X - y}$ temos

$$S(X) \equiv \sum_{y \in L} \frac{w_y}{X - y} \bmod g(X),$$

que é a definição de polinômio síndrome com $\deg(S(X)) < \deg(g(X))$. Pela pro-

priedade transitiva da congruência:

$$S(X) \equiv \sum_{y \in L} \frac{e_y}{X - y} \bmod g(X).$$

Dado M, o subconjunto de L, onde $e_y \neq 0$, então:

$$S(X) \equiv \sum_{y \in M} \frac{e_y}{X - y} \bmod g(X). \tag{2.2}$$

Introduziremos agora a definição de dois polinômios de muita importância para a correção de erros.

Definição 2.5.4. ***(Polinômio Localizador e Polinômio Avaliador)***
-Polinômio Localizador: $\sigma(X) = \prod_{y \in M} (X - y)$*;*
-Polinômio Avaliador: $\omega(X) = \sum_{y \in M} e_y \prod_{z \neq y} (X - z)$.

Dado que $\omega(\mathbf{y}) \neq \mathbf{0}$ e $\sigma(\mathbf{y}) = \mathbf{0}$, $\forall \mathbf{y} \in M$, os polinômios definidos acima são primos entre si. Calculando a derivada de $\sigma(X)$, obtemos:

$$\sigma'(X) = \sum_{y \in M} \prod_{z \neq y} (X - z).$$

Agora, multiplicando a equação (2.2) por $\sigma(X)$, obtemos:

$$\begin{aligned} S(X)\sigma(X) &\equiv \prod_{y=M} (X - y) \sum_{y \in M} \frac{e_y}{X - y} \bmod g(X) \\ &\equiv \omega(X) \bmod g(X). \end{aligned} \tag{2.3}$$

Esta congruência pode ser descrita como o seguinte problema:

Problema 1

Dado $S(X) = \sum_{i=0}^{\deg(S)} S_j X^j$ e $g(X) = \sum_{i=0}^{\deg(g)} g_j X^j$, quais são os polinômios $\sigma(X) = \sum_{i=0}^{\deg(\sigma)} \sigma_j X^j$ e $\omega(X) \sum_{i=0}^{\deg(\omega)} \omega_j X^j$ de menor grau, tais que a congruência acima seja satisfeita?. Mostraremos a seguir que existe um único par $\omega(X)$ e $\sigma(X)$

com

$$\deg(\sigma(X)) \leq \delta \text{ e } \deg(\omega(X)) < \delta, \tag{2.4}$$

tal que a congruência (2.3) seja satisfeita, se $g(X)$ é um polinômio livre de quadrados. A solução do problema acima implica num algoritmo de decodificação, de capacidade $\lfloor \delta \rfloor$. É importante indicar que esta, também, é uma demostração da Proposição 2.5.2.

Calculando o resto da divisão $S(X)\sigma(X)$ por $g(X)$ obtém-se o polinômio $\omega(X) = \lambda(X) = \sum_{i=0}^{\deg(\omega(X))} \lambda_i X^i$, onde os λ_i estão em função dos coeficientes conhecidos, S_i e g_i, e dos desconhecidos, σ_i. Esta equação pode ser expressa como um sistema de equações lineares da seguinte maneira:

$$\omega_i = \lambda_i, \text{ para } 0 \leq i \leq \deg(g(X)) - 1.$$

Para provar que um decodificador pode corrigir até $\lfloor \delta \rfloor$ erros, sem ambiguidades (veja *política da máxima proximidade*), o sistema acima tem que ter uma única solução com as condições (2.4). Para demonstrar isto suponha que existem dois pares, (σ_1,ω_1) e (σ_2,ω_2), de soluções diferentes para a equação (2.3), tal que:

$$S(X)\sigma_1(X) \equiv \omega_1(X) \bmod g(X) \tag{2.5}$$

e

$$S(X)\sigma_2(X) \equiv \omega_2(X) \bmod g(X), \tag{2.6}$$

onde $\sigma_i(X)$ e $\omega_i(X)$, para $1 \leq i \leq 2$, são primos entre si. Dividindo (2.5) e (2.6) por $\sigma_i(X)$, para $i = 1$ e $i = 2$, respectivamente, obtemos:

$$S(X) \equiv \frac{\omega_1(X)}{\sigma_1(X)} \bmod g(X)$$

e

$$S(X) \equiv \frac{\omega_2(X)}{\sigma_2(X)} \bmod g(X).$$

Esta divisão é possível porque $\sigma_i(X)$ e $g(X)$ não possuem fator algum em comum, já que se houver algum fator em comum então esse fator pode dividir ω_i, contradizendo a hipótese de que $\sigma_i(X)$ e $\omega_i(X)$ são primos entre si.

Seguindo com a demostração, aplicamos transitividade e obtemos:

$$\frac{\omega_1(X)}{\sigma_1(X)} \equiv \frac{\omega_2(X)}{\sigma_2(X)} \bmod g(X)$$

$$\implies \sigma_1(X)\omega_2(X) \equiv \sigma_2(X)\omega_1(X) \bmod g(X) \tag{2.7}$$

Supondo que o $\deg(g(X)) = 2 \cdot \delta$ e que as condições (2.4) são satisfeitas, obtemos:

$$\sigma_1(X)\omega_2(X) = \sigma_2(X)\omega_1(X).$$

Sendo $\sigma_i(X)$ e $\omega_i(X)$ primos entre si e mônicos (veja definição de polinômio localizador), para $i = 1, 2$, então $\sigma_1(X) = \sigma_2(X)$. Portanto $\omega_1(X) = \omega_2(X)$, ficando a unicidade do sistema linear acima garantida com as condições (2.4) e $\deg(g(X)) = 2 \cdot \delta$.

No caso binário, devido a $e_y = 1$, com $y \in M$, a equação chave é:

$$S(X)\sigma(X) \equiv \sigma'(X) \bmod g(X). \tag{2.8}$$

Vamos demonstrar a seguir que se $\deg(g(X)) = \delta$ o código de Goppa binário pode

corrigir até δ erros. Tomando (2.7), temos:

$$\sigma_1(X)\omega_2(X) \equiv \sigma_2(X)\omega_1(X) \bmod g(X). \tag{2.9}$$

Desdobrando $\sigma_i(X)$ em sua parte par e ímpar, obtemos:

$$\sigma_i(X) = \sigma_{p_i}(X) + X\sigma_i'(X). \tag{2.10}$$

Já que $\sigma_i(X)$ pertence a um corpo de característica dois, então $\sigma_i'(X)$ é sempre um polinômio par. Portanto,

$$(\sigma_{p_1}(X) + X\sigma_1'(X))\sigma_2'(X) \equiv (\sigma_{p_2}(X) + X\sigma_2'(X))\sigma_1'(X).$$

Somando σ_{p_2} nos dois lados da equação acima, e tendo em conta que estamos trabalhando em um corpo de característica 2, temos :

$$\sigma_{p_1}(X)\sigma_2'(X) + \sigma_{p_2}(X)\sigma_1'(X) \equiv 0 \bmod g(X).$$

Sabendo-se que o resultado da soma de duas funções pares é uma função par, temos que a expressão do lado esquerdo da equação acima é um quadrado perfeito, levando-nos a concluir que

$$\sigma_{p_1}(X)\sigma_2'(X) + \sigma_{p_2}(X)\sigma_1'(X) \equiv 0 \bmod g_2(X),$$

onde $g_2(X)$ é o menor múltiplo de $g(X)$ tal que $g_2(X)$ seja um polinômio quadrado perfeito. Supondo que $g(X)$ é livre de quadrado, então $g_2(X) = g(X)^2$, e assim $\deg(g_2(X)) = 2 \cdot \delta$. Se $\deg(\sigma_{p_i}(X)) \leq \delta$, então:

$$\sigma_{p_1}(X)\sigma_2'(X) = \sigma_{p_2}(X)\sigma_1'(X).$$

Como $\sigma_{p_1}(X)$ e $\sigma_1'(X)$ são primos entre si então $\sigma_1 = \sigma_2$. O processo descrito

acima dá como resultado a prova da Proposição 2.5.2. Devido a que toda função polinomial par é um quadrado perfeito então a equação (2.10) pode ser expressa como :

$$\sigma(X) = a_0(X)^2 + Xa_1(X)^2, \tag{2.11}$$

com

$$\deg(a_0(X)) \leq \delta/2 \text{ e } \deg(a_1(X)) \leq (\delta - 1)/2, \tag{2.12}$$

Derivando (2.11) acima, temos :

$$\sigma'(X) = 2a_0(X)a_0'(X) + a_1(X)^2 + 2a_1(X)a_1'(X)X = a_1(X)^2.$$

Substituindo este resultado na equação (2.3), obtemos

$$\begin{aligned} & a_1(X)^2 \equiv (a_0(X)^2 + Xa_1(X)^2)S(X) \bmod g(X) \\ \Leftrightarrow\ & a_1(X)^2(1 + XS(X)) \equiv a_0(X)^2 S(X) \bmod g(X) \\ \Leftrightarrow\ & a_0(X) \equiv a_1(X)\sqrt{X + 1/S(X)} \bmod g(X). \end{aligned} \tag{2.13}$$

Na Definição 2.5.5 o polinômio localizador está na forma recíproca, levando na mudança do polinômio avaliador. Estes polinômios junto com o polinômio síndrome estão relacionados na equação chave da Proposição 2.5.3. Esta equação é usada para algoritmos de decodificação dos códigos de Goppa ou alternantes, não necessariamente binários, com capacidade de correção até $\lfloor \delta/2 \rfloor$ erros.

Definição 2.5.5. ***(Polinômio Localizador e Polinômio Avaliador, caso alternante)***

-Polinômio Localizador: $l(X) = \prod_{y \in M} (1 - yX)$*;*

-Polinômio Avaliador: $a(X) = l(X) + \sum_{y \in M} e_y \prod_{z \neq y} (1 - zX)$.

Proposição 2.5.3. *(Hefez e Villela (2008)) No anel de polinômios* $F[X]$ *vale a congruência*

$$a(X) \equiv S(X)l(X) \mod X^{\delta}, \tag{2.14}$$

tal que

$$\deg(l(X)) \leq \lfloor \delta/2 \rfloor \ e \ \deg(a(X)) < \lfloor \delta/2 \rfloor - 1. \tag{2.15}$$

Demonstração. Veja a demostração em Hefez e Villela (2008). □

Capítulo 3

Algoritmos de Decodificação

A utilidade de um sistema de criptografia seria muito diminuída se o processo de decifração fosse muito demorado. A seguir apresentaremos alguns algoritmos de decodificação para códigos de Goppa, os quais são o suporte matemático para os algoritmos de decifração e assinatura de sistemas criptográficos como McEliece (1978) e Niederreiter (1986).

Dentro dos algoritmos para decodificação de códigos de Goppa existem aqueles que usam o polinômio síndrome, veja seção 2.2.2. Os algoritmos para decodificação de códigos de Goppa variam em função dos recursos tempo e memória. Normalmente, os melhores algoritmos, em função do tempo, para decodificação são aqueles projetados para tipos específicos de códigos.

Os algoritmos para decodificar códigos alternantes, a excepção dos de força bruta, são conhecidos. Por exemplo, temos os apresentados em (MacWilliams e Sloane, 1997, cap. 12.9). Estes podem corrigir até $\lfloor \delta/2 \rfloor$ erros. No entanto, o algoritmo proposto por Guruswami e Sudan (1999), pode corrigir mais de $\lfloor \delta/2 \rfloor$ erros, especificamente $n - n(n-\delta) \approx \sqrt{\delta/2 + (\delta/2)2/(2n-\delta)}$ erros. Já, o algoritmo de Patterson, como veremos na seção 3.1, pode corrigir δ erros num código de Goppa binário. O algoritmo proposto por Bernstein (2008), é um pouco melhor, e alcança uma capacidade de correção de $n - \sqrt{n(n-2\delta-2)} \approx \delta + 1 + (\delta+1)2/2(n-\delta-1)$ erros para códigos de Goppa binário irredutíveis. Técnicas similares podem corrigir $n - \sqrt{n(n-r\delta)} \approx r\delta/2 + (r\delta/2)2/(2n-r\delta)$ erros para códigos de Wild

como apresentados Bernstein et al. (2011a), veja Barreto et al. (2011).

Os algoritmos de decodificação para os códigos de Goppa binário, de forma geral, têm os seguintes passos:

- Cálculo do polinômio síndrome;
- Solução da *equação chave*;
- Localização e correção de erros.

O primeiro passo é feito através da multiplicação da matriz de teste de paridade pelo vetor recebido no decodificador. O segundo passo, que é a parte essencial dos algoritmos de decodificação que apresentamos, pode ser calculado mediante os algoritmos apresentados neste trabalho. O terceiro passo é o calculo das raízes $\alpha_i = 1/L_i$ do polinômio localizador $\sigma(X)$. Estas raízes indicam a posição i onde ocorreu um erro. Para a correção dos erros, devido a fato que F tem característica dois, basta com inverter o valor na posição i do vetor recebido pelo decodificador.

Iremos analisar alguns algoritmos que lidam com a família de códigos de Goppa binário e códigos Alternantes.

3.1 Algoritmo de Patterson

Este algoritmo foi proposto por N. J Patterson em 1974, desde aquele tempo até agora existem investigações e modificações deste. Berlekamp em 1973 escreveu:

"*Unfortunately, no method of comparable simplicity is know for solving* (7) *in the case of more general* $g(z)$"

Motivado nesse comentário, Patterson desenvolveu um algoritmo cuja a ideia principal, na forma original, é usar um LFSR (*Linear Feedback Shift Register*) para encontrar $h(X)$ (veja o passo 4 no **Algoritmo** 1) e, portanto, mudar o problema da equação chave (2.3) para uma outra equação chave onde $g(X) = X^{\delta}$, que é o caso que Berlekamp resolveu. O LFSR proposto é o seguinte.

$$\begin{aligned}\mathbf{f}^{(i+1)} &= (f_0^{(i+1)}, f_1^{(i+1)}, \cdots, f_{n-1}^{(i+1)}) \\ &= (-f_{n-1}^{(i)} g_0, f_0^{(i)} - f_{n-1}^{(i)} g_1, \cdots, f_{n-2}^{(i)} - f_{n-1}^{(i)} g_{n-1}),\end{aligned}$$

onde as condições inicias são $f_i^0 = 0$, para $i = 0, ..., n-1$, e a saída, h_i, é o conteúdo da célula que encontra-se mais à direita, i.e. $h_i = f_{n-1}^{(i)}$. Vejamos um exemplo para um melhor entendimento.

Exemplo. Sejam $K = \mathbb{F}_2$ e $F = \mathbb{F}_{2^4} = K[X]/(X^4 + X + 1)$. Sejam a o elemento primitivo de F e os polinômios $g(X) = X^4 + (a+1) \cdot X^3 + (a^3 + a) \cdot X^2 + a \cdot X + 1$ e $f(X) = (a^3 + a^2 + 1) \cdot X + a^2 + a + 1 \in F[X]$, então temos: e por tanto

Iteração	**célula 1**	**célula 2**	**célula 3**	**célula 4**
0	0	0	0	0
1	$a^2 + a + 1$	$a^3 + a^2 + 1$	0	0
2	0	$a^2 + a + 1$	$a^3 + a^2 + 1$	0
3	0	0	$a^2 + a + 1$	$a^3 + a^2 + 1$
4	$a^3 + a^2 + 1$	$a^3 + 1$	$a^3 + a + 1$	$a + 1$

Tabela 3.1: Valores de cada célula no circuito LFSR.

$h_0 = 0,\ h_1 = 0,\ h_2 = a^3 + a^2 + 1$ e $h_3 = a + 1$.

Em **Algoritmo 1** apresentamos o algoritmo de Patterson. Para a demonstração desse são necessários alguns resultados preliminares que apresentamos agora.

Denotaremos como $R[[X]]$ o anel de séries de potências formais com coeficientes no corpo K, ou seja, se $g(X) \in R[[X]]$, então $\sum_{i=-\infty}^{\infty} a_i X^i$, onde para cada $i \geq 0$ $a_i = 0$ a menos de uma quantidade finita. Um resultados simples é que $K[X] \subset R[[X]]$.

Algoritmo 1 Algoritmo de Patterson Original

Entrada: Polinômio síndrome $S(X)$.

Assegura: $\sigma(X)$ com $\deg(\sigma(X)) \leq \lfloor n/2 \rfloor$ e $\omega(X)$ com $\deg(\omega(X)) \leq \lfloor (n-1)/2 \rfloor$

1: **for** i=0 **to** n-1 **do**
2: a_i = coeficiente associado ao termo de grau $n-1$ em $X^i \cdot S(X)$.
3: **end for**
4: Criar $h(Y) = \sum_{i=0}^{n-1} a_i Y^i$.
5: **if** n é par **then**
6: Usar Algoritmo 3 com entradas: h, Y^n para encontrar σ_1, ω_1 e N em $h(Y)\sigma_1(Y) \equiv \omega_1(Y) \bmod Y^n$ onde $N = \max(\deg(\sigma_1), \deg(\omega_1) + 1)$
7: **else**
8: Usar Algoritmo 3 com entradas: h, Y^{n-1} para encontrar σ_1, ω_1 e N em $h(Y)\sigma_1(Y) \equiv \omega_1(Y) \bmod Y^{n-1}$ onde $N = \max(\deg(\sigma_1), \deg(\omega_1) + 1)$
9: **end if**
10: Suponha $\sigma_1(X) = \sum_{i=0}^{r} c_i Y^i$, $r \leq N$, então faça $\sigma(X) = \sum_{i=0}^{r} c_i Y^{N-i}$ e $\omega(X) = S(X)\sigma(X) \bmod g(X)$

Algoritmo 2 Algoritmo de Euclides Estendido Modificado

Entrada: Dois polinômios não nulos $s(X), o(X) \in F[X]$.

1: A ⟵ $s(X)$
2: B ⟵ $o(X)$
3: U ⟵ 1
4: G ⟵ A
5: V1 ⟵ 0
6: V3 ⟵ B
7: **while** TRUE **do**
8: Q,R ⟵ G.quocienteResiduo(V3) (1)
9: T ⟵ U - V1*Q
10: U ⟵ V1
11: G ⟵ V3
12: V1 ⟵ T
13: V3 ⟵ R
14: **if** deg(V3) $< \delta/2$ **then**
15: break
16: **end if**
17: **end while**
18: V ⟵ quociente((G-A*U),B)(2)
19: lc ⟵ G.coefficenteLider()(3)
20: **return** G/lc, U/lc, V/lc

Teorema 3.1.1 (Patterson (1974)). *Sejam* $\alpha, \beta \in R[[X]]$ *polinômios, com* $\deg(\beta) = n$ *e* $\alpha = q\beta + r$*, onde* r *é um polinômio de grau menor que* n*. Suponha* $r \neq 0$*. Se:*

$$\alpha(X) = (q'(X) + \sum_{i=1}^{\infty} b_i X^{-i})\beta(X),$$

onde $q'(X)$ *é um polinômio em* X*,* $b_1 = b_2 = ... = b_s = 0$ *e* $b_{s+1} \neq 0$*, então* $\deg(r) = n - s - 1$*.*

Demonstração. Para $a = 0$, temos $X^a\alpha(X) = q_a(X)\beta(X) + r_a(X)$, onde $\deg(r_a(X)) < n$. Supondo que o $s = 0$, temos que

$$\alpha(X) = q'(X)\beta(X) + b_1 X^{-1}\beta(X)$$

e, por tanto, dado que o quociente e o resíduo numa divisão de polinômios são únicos, $\deg(r) = n - 1$. Agora se $a > 0$

$$X^a\alpha(X) = (q'(X)X^a + \textstyle\sum_{j=i}^{a} b_j X^{a-j} + \sum_{i=1}^{\infty} b_{i+a}X^{-i})\beta(X).$$

Fazendo $q''(X) = q'(X)X^a + \sum_{j=i}^{a} b_j X^{a-j}$ então:

$$\begin{aligned} q_a(X)\beta(X) &= X^a\alpha(X) - r_a(X) \\ &= \beta(X)q''(X) + \beta(X) \cdot \sum_{i=1}^{\infty} b_{i+a}X^{-i} - r_a(X). \end{aligned}$$

Portanto, $(q_a(X) - q''(X))\beta(X) - \beta(X)\sum_{i=1}^{\infty} b_{i+a}X^{-i} + r_a(X) = 0$. Tomando os coeficientes de X^k, para $k \geq n$, temos que $(q_a(X) - q''(X)) = 0$, dado que nos outros termos o grau é menor do que n, por hipótese. Assim $r_a(X) = \beta(X)\sum_{i=1}^{\infty} b_{i+a}X^{-i}$, onde b_{a+1} é o coeficiente do termo de maior grau (X^{n-1}). Já que $r_a(X) = r(X)X^a$ então $\deg(r) = n - a - 1$. □

Proposição 3.1.2 (Patterson (1974)). *Seja a serie* $A = \sum_{n=-\infty}^{m} a_{m-n}X^n \in R[[X]]$*, se* a_0 *é inversível em* K *então* A *é inversível em* $R[[X]]$*.*

Demonstração. Se a_0 é inversível em K, então existe a inversa $B = b_0X^{-m} + b_1X^{-m-1} + \cdots$, onde seus coeficientes, b_i, estão dados pela formula recursiva, a seguir.

$$b_0 = \frac{1}{a_0} \text{ e } b_n = \frac{-1}{b_0} \sum_{i=1}^{n} a_i b_{n-i}, \text{ para } n \geq 1$$

□

Após os resultados preliminares apresentados, justificaremos o **Algoritmo** 1.

Configurando $g(X) = \sum_{n=-\infty}^{\delta} a_{\delta-n} X^n \in R[[X]]$, onde $a_i = 0$ para $i < 0$ e seja $f(X) \in R[[X]]$, então $f(X) = f(X)g(X)^{-1}g(X)$, e pela proposição 3.1.2 temos $f(X) = ((a_0X^{-1} + a_1X^{-2} + \ldots + a_{n-1}X^{-n}) + b_nX^{-(n-1)} + \ldots)g(X)$. Fazendo uma troca de variável ($Y = X^{-1}$) no polinômio $h(Y)$ do passo 4 no **Algoritmo** 1 temos:

$$(a_0 + a_1X^{-1} + ... + a_{n-1}X^{-(n-1)})\sigma'(X^{-1}) \equiv \omega'(X^{-1}) \bmod X^{-M}, \tag{3.1}$$

com M satisfazendo Caso A e Caso B do **Algoritmo** 1. Multiplicando $f(X)$ por $\sigma'(X^{-1})$ temos:

$$\begin{aligned}
f(X)\sigma'(X^{-1}) &= ((a_0X^{-1} + \cdots + a_{n-1}X^{-n}) + b_nX^{-(n=1)} + \cdots \\
&\quad + \cdots)\sigma'(X^{-1})g(X) \\
&= ((a_0X^{-1} + \cdots + a_{n-1}X^{-n})\sigma'(X^{-1}) + (b_nX^{-(n-1)} + \cdots \\
&\quad + \cdots)\sigma'(X^{-1}))g(X) \\
&= ((a_0X + \cdots + a_{n-1}X^{-(n-1)})\sigma'(X^{-1})X^{-1} + (b_nX^{-(n-1)} + \cdots \\
&\quad + \cdots)\sigma'(X^{-1}))g(X) \\
&= (X^{-1}\omega'(X^{-1}) + X^{-(M+1)}(d_0 + d_1X^{-1} + \cdots))g(X) \text{ (usando (3.1)).}
\end{aligned}$$

Seja $\sigma(X) = X^N\sigma'(X^{-1})$ e lembrando do **Algoritmo** 1, passo 5, que

$$N = \max(\ \deg(\sigma'(X)),\ \deg(\omega'(X)) + 1),$$

então temos que: $f(X)\sigma'(X^{-1}) = (\lambda(X) + X^{-(M+1-N)}(d_0 + d_1X^{-1}\ldots))g(X)$. E considerando o Teorema 3.1.1, sabemos que $(f(X)\sigma(X) \bmod g(X)) = \omega(X)$ tem grau $\leq n - 1 - (M - N) = N - 1$ se n é par e grau $\leq N$ se n é ímpar. Portanto $\deg(\sigma(X)) \leq N$ e $\deg(\omega(X)) \leq N - 1$ se n é par e $\deg(\omega(X)) \leq N$ se n é

ímpar. Pelo Algoritmo 3 em Patterson (1974) temos que: $N \leq n/2$, se n é par e $N \leq (n-1)/2$, se n é impar. Logo a condição dos polinômios $\sigma(X)$ e $\omega(X)$, no passo 10 do **Algoritmo** 1, é satisfeita.

3.2 Algoritmo de Patterson no caso binário

Para o caso binário, faz-se $\omega(X) = \sigma'(X)$ e usa-se a equação (2.8) como entrada no **Algoritmo 1**, ou pode-se usar o algoritmo de Euclides estendido modificado, **Algoritmo 2**, com entradas $g(X)$ e $\sqrt{X + 1/S(X)}$ para encontrar os valores $a_0(X)$ e $a_1(X)$ na equação (2.13). Um pré-cálculo, antes de usar o **Algoritmo 1** ou **Algoritmo 2**, é encontrar o valor de $\sqrt{X + 1/S(X)}$. Para fazer isto de um modo mais simples que no passo 2 do Algoritmo 4 em Patterson (1974), vamos descrever em detalhe o método proposto em Hubber (1996).

Da equação (2.13) tem-se que existe $R(X)$, tal que $R(X)^2 \equiv t(X) \bmod g(X)$, com $t(X) = X + 1/S(X)$. Desdobrando $g(X)$ na parte par e ímpar, temos:

$$\begin{aligned} & g_0(X)^2 + Xg_1(X)^2 = g(X) \\ \Leftrightarrow\ & g_0(X)^2 + 2 \cdot Xg_1(X)^2 = g(X) + Xg_1(X)^2 \\ \Leftrightarrow\ & g_0(X)^2 = g(X) + Xg_1(X)^2. \end{aligned}$$

Logo, $g_0(X)^2 \equiv Xg_1(X)^2 \bmod g(X)$. Devido a $g(X)$ ser irredutível então $1 = mdc(X, g(X))$. Por tanto, existe $w(X)$ tal que: $w(X)^2 \equiv X \bmod g(X)$, o que nos leva a concluir que $g_0(X) \equiv w(X)g_1(X) \bmod g(X)$. Assumindo que $g_1(X)$ e $g(X)$ são coprimos, então $w(X) \equiv g_0(X)g_1(X)^{-1} \bmod g(X)$. Desdobrando $t(X)$ na parte par e ímpar temos $t(X) = t_0(X)^2 + Xt_1(X)^2$ e, por tanto, $R^2(X) \equiv t \equiv (t_1(X)+w(X)t_2(X))^2 \equiv \bmod g(X)$. Finalmente, $R(X) \equiv t_1(X)+w(X)t_2(X) \bmod g(X)$. No **Algoritmo** 4, adicionando os passos do método proposto por Hubber (1996).

3.3 Algoritmo de Barreto

Este algoritmo foi proposto por Barreto et al. (2011) como uma generalização do algoritmo de decodificação proposto por Patterson, visto na seção anterior.

É um algoritmo para decodificar códigos de Goppa livres de quadrado, no corpo $\mathbb{F}_p$, para algum p primo. O algoritmo tem uma capacidade de correção de $(2/p)t$. Especificamente, no caso binário, a capacidade de correção é a mesma que o algoritmo de Patterson e com uma complexidade de $O(p^3t^2)$ passos no pior dos casos. Antes de fazer uma descrição do algoritmo veremos alguns conceitos que serão usado aqui.

Definição 3.3.1. *Dado a matriz polinomial $A \in \mathbb{F}_q[X]^{n\times m}$, com posto de linha r, denomina-se reticulado polinomial $\Lambda(A)$ sobre $\mathbb{F}_q[X]$, gerado pelas linhas de A à seguinte expressão:* $\Lambda(A) = \{(u_0, ..., u_n)A \in \mathbb{F}_q[X]^m | (u_0, ..., u_n) \in \mathbb{F}_q[X]^n\}$

Definição 3.3.2. *Seja o vetor de polinômios $v \in \mathbb{F}_q[X]^k$, define-se como norma do máximo grau a* $|v| = \max(\deg(v_i))$.

Nos inteiros o problema de encontrar o menor vetor, em relação à norma Euclideana, num reticulado é um problema NP-difícil. Já nos reticulados polinomiais este problema, em relação à norma da Definição 3.3.2, em $\Lambda(A)$ pode-se resolver num tempo polinomial.

Teorema 3.3.1 (Barreto et al. (2011))**.** *Existe um algoritmo o qual encontra o menor vetor em $\Lambda(A)$ com $O(mnrd^2)$ operações em $\mathbb{F}_q$, onde*

$$d = \max\left\{\deg(A_{i,j}) | 1 \leq i \leq n, \leq j \leq m\right\}.$$

O algoritmo indicado no Teorema 3.3.1, é o **Algoritmo** 5, denominado WeakPovovForm veja (veja Apêndice A em Barreto et al. (2011))
No caso binário o algoritmo de Barreto, **Algoritmo** 6, justifica-se do fato que a equação (2.13) pode ser vista como uma equação diofântica com variáveis a_0 e a_1 e pode-se resolver com a restrição (2.12). Para isto constrói-se a matriz

$$A = \begin{bmatrix} g(X) & 0 \\ -v & 1 \end{bmatrix},$$

e calcula-se os polinômios de graus menores no reticulado gerado pelas filas da matriz A. Isso devido ao fato de que $(\lambda, a_0(X), a_1(X))A = (\lambda \cdot g(X) - a_1(X)v, a_1) = (a_0, a_1)$. Pode-se usar o **Algoritmo** 5 para encontrar a_0 e a_1. Com isso, então pode-se calcular o polinômio localizador usando a equação (2.11).

É importante indicar que infelizmente a capacidade de encontrar os menores vetores no reticulado $\Lambda(A)$, usando **Algoritmo** 5, não garante a solução de uma equação chave, quando o corpo tem característica diferente de dois.

3.4 Algoritmo Alternante

Descreveremos, na continuação, um algoritmo para decodificação de códigos alternantes. Para isto consideramos primeiro o estudo do algoritmo da divisão de polinômios.

3.4.1 Divisão de Polinômios

Dados os polinômios $a(X)$, $b(X)$ e $d(x)$ no anel $K[X]$. Suponha que para todo $f \in K[X]$, se $f(X)|a(X)$ e $f(X)|b(X)$, então $f(X)|d(X)$. Logo, $d(X)$ é chamado de máximo divisor comum de $a(X)$ e $b(X)$ e sera denotado como $mdc(a(X), b(X))$.

Teorema 3.4.1. *O máximo divisor comum, $d(X)$, de dois polinômios, não simultaneamente nulos, $a(X)$ e $b(X)$ em $K[X]$ existe e pode ser escrito na forma.*

$$d(X) = \lambda(X) \cdot a(X) + \mu(X) \cdot b(X)$$

É fácil demostrar que máximo divisor comum, $d(X)$, é associado a um único polinômio mônico que também é um mdc de $a(X)$ e $b(X)$.

Consideremos o problema de determinar $\lambda(X)$ e $\mu(X)$, no teorema acima. Para isto descreveremos o algoritmo de Euclides estendido. Dividendo $a(X)$ por $b(X)$ temos:

$a(X) = b(X) \cdot q(X) + r(X)$, com $r(X) = 0$ ou $\deg(r(X)) < \deg(g(X))$ Fazemos repetidas divisões para obter a serie de equações:

$a(X) = b(X)q_1(X) + r_1(X)$; $r_1(X) = 0$ ou $\deg(r_1(X))<\deg(a(X))$

$b(X) = r_1(X)q_2(X) + r_2(X)$; $r_2(X) = 0$ ou $\deg(r_2(X))<\deg(r_1(X))$

$r_1(X) = r_2(X)q_3(X) + r_3(X)$; $r_3(X) = 0$ ou $\deg(r_3(X))<\deg(r_2(X))$

...

$r_{t-2}(X) = r_{t-1}(X)q_t(X) + r_t(X)$; $r_t(X) = 0$ ou $\deg(r_t(X))<\deg(r_{t-1}(X))$

$r_{t-1}(X) = r_t(X)q_{t+1}(X)$

Assim o último resto não nulo $r_t(X)$ no processo de divisão será o *mdc* de $a(X)$ e $b(X)$.

3.4.2 Método de Decodificação Alternante

Seja $r_{-1} = X^\delta$ e $r_0 = s(X)$, então a serie de equações da Seção anterior podem ser determinadas mediante a seguinte expressão:

$$r_{i-1}(X) = q_{i+1}r_i(X) + r_{i+1}(X) \tag{3.2}$$

com $r_{i+1}(X) = 0$ ou $\deg(r_{i+1}(X)) < \deg(r_i(X))$. Seja t o primeiro índice tal que $r_t(X) \neq 0$ e $r_{t+1} = 0$. Definimos recursivamente para $i = 1, ..., t$ os polinômios.

$$z_{-1}(X) = 0,\ z_0(X) = 1$$
$$z_i(X) = z_{i-2}(X) - q_i(X) + z_{i-1}(X)$$

Proposição 3.4.2 (Hefez e Villela (2008))**.** *As seguintes afirmações são válidas:*
(i) $r_i(X) \equiv z_i(X)s(X) \mod X^\delta$, *para* $i = -1, 0, ...t$;
(ii) $\deg(z_i(X)) = \delta - \deg(r_{i-1}(X))$, *para* $i = 0, ...t$;

Proposição 3.4.3 (Hefez e Villela (2008))**.** *O polinômio localizador de erros* $l(X)$ *e o polinômio avaliador de erros* $a(X)$*, no Teorema 2.5.3, são dados por:*

$$l(X) = z_k(0)^{-1}z_k(X);$$
$$a(X) = z_k(0)^{-1}r_k(X)$$

onde k *é o primeiro índice para o qual* $\deg(r_k(X)) < \delta/2$.

Demonstração. Da Equação (2.15), sabemos que $\deg(l(X)) \leq \delta/2$ e $\deg(a(X)) < \delta/2$. Seja $d(X)$ o *mdc* de X^δ e $s(X)$, logo $d(X)\lambda(X) = X^\delta$ e $d(X)\mu(X) = s(X)$. Do Teorema 2.5.3 temos que: $a(X) = \eta(X)X^\delta + s(X)l(X)$, assim $a(X) = \eta(X)d(X)\lambda(X) + d(X)\mu(X)l(X) \iff a(X) = d(X)(\eta(X)\lambda(X) + \mu(X)l(X))$, por tanto $d(X)|a(X)$. Dado que $r_t(X)$ também é um *mdc* de X^δ e $s(X)$, tem-se que $r_t(X)$ difere $d(X)$ pelo produto de uma constante não nulo em F. Logo $\deg(r_t(X)) = \deg(d(X)) \leq \deg(a(X)) \leq \delta - 1$. Da Equação (3.2) temos que:

$$\deg(r_t(X)) < \deg(r_{t-1}(X)) < ... < \deg(r_1(X)) < \deg(r_0(X)) < \deg(r_{-1}(X)).$$

Logo existe $h \in \mathbb{N}$, com $0 \leq h \leq t$, tal que:

$$\deg(r_h(X)) \leq \deg(a(X)) \text{ e } \deg(r_{h-1}(X)) \geq \deg(a(X)) + 1. \tag{3.3}$$

A expressão do lado direito, acima, é igual $-\deg(r_{h-1}(X)) \leq -\deg(a(X)) - 1$. Por tanto Proposição 3.4.2 (ii) tem-se que

$$\deg(z_h(X)) = \delta - \deg(r_{h-1}(X)) \leq \delta - \deg(a(X)) - 1. \tag{3.4}$$

Da Proposição 3.4.2 (i) e da Proposição 2.5.3 tem-se que:

$$r_h(X) \equiv z_h(X)s(X) \bmod X^\delta \tag{3.5}$$

Multiplicando por $l(X)$ à (3.5) e por $z_h(X)$ à (2.14) temos

$$a(X)z_h(X) \equiv (s(X)l(X))z_h(X) \equiv r_h(X)l(X) \bmod X^\delta,$$

aplicando transitividade de congruências. Da Equação (3.4) tem se que $\deg(a(X)) + \deg(z_h(X)) \leq \delta - 1$. Da Equação (3.3), temos que $\deg(r_h(X)) + \deg(l(X)) \leq \deg(a(X)) + \deg(l(X)) \leq \deg(a(X)) + \lfloor \delta/2 \rfloor \leq 2\lfloor \delta/2 \rfloor - 1 \leq \delta - 1$. Por tanto

$$a(X)z_h(X) = r_h(X)l(X) \tag{3.6}$$

Assim, $a(X)|r_h(X)l(X)$, mas como $l(X)$ e $a(X)$ são primos entre si, $a(X)|r_h(X)$.

Mas como $\deg(r_h(X)) \leq \deg(a(X)) \Longrightarrow \deg(r_h(X)) = \deg(a(X))$, existe α, não nulo, que pertence F, tal que $a(X) = \alpha r_h(X)$. Por tanto, de (3.6) temos que $\alpha z_h(X) = l(X)$ e de $l(0) = 1$, substituindo $X = 0$ temos $\alpha = z_h(0)^{-1}$. Resta ainda mostrar que $h = k$, para que a Proposição seja demostrada. De $\deg(r_{h-1}(X)) \leq \deg(r_k(X)) < \delta/2$ conclui-se que $h = k$. □

3.5 Algoritmo Berlekamp-Massey

Em 1968 E. Berlekamp publico um algoritmo para decodificação de códigos BCH, Um ano depois J Massey (1969) publica uma variação deste algoritmo, que pode ser usado para decodificar vários tipos de códigos algébricos.

Como vimos no **Problema 1**, o algoritmo de Berlekamp utiliza uma equação chave para introduzir um número conhecido de coeficientes, num polinômio $\lambda(X)$ e, em seguida determininese os coeficientes restantes deste polinômio, mediante a solução de um sistema de equações lineares. Tanto E. Berlekamp como J. Massey reformulam o problema de maneira que evita-se pensar nas matrizes de n por n de forma explícita, já que o trabalho e o volume de armazenamento deste tipo de operação é muito grande.

As aplicações e implementação deste algoritmo foram estendidos por J. Massey que usou a interpretação física de um LFSR como uma ferramenta para entender melhor o algoritmo. A interpretação do algoritmo em sua forma usual é apresentada no **Algoritmo** 8 e a prova do algoritmo pode ser encontrada em Berlekamp (1984)

A seguir apresentamos uma tabela comparativa das complexidades assintóticas dos algoritmos descritos neste capítulo.

[PAT]	[ALT]	[BMA]	[BAR]
$O(\delta^2)$	$O(\delta^2)$	$O(\delta^2)$	$O(p^3\delta^2)$

Tabela 3.2: Complexidades assintóticas dos algoritmos de decodificação apresentados no Capítulo 3.

Algoritmo 3 Algoritmo Berlekamp

Entrada: Polinômio síndrome $S(X) = \sum_{i}^{0} X^i \in F[X], 0 < N$, $g(X)$ com $\deg(g(X)) = \delta$.
Assegura: Polinômio localizador $\sigma(X)$; Polinômio avaliador $\omega(X)$;

if $S_0 = 1$ **then**
  $\sigma_0(X) = 1$; $\tau_0(X) = 1$; $\omega_0(X) = 1$; $\gamma_0(X) = 0$.
  $D(0) = 0$; $B(0) = 0$.
end if
if $S_0 = 0$ **then**
  $\sigma_0(X) = 1$; $\tau_0(X) = 1$; $\omega_0(X) = 0$; $\gamma_0(X) = -1$.
  $D(0) = 0$; $B(0) = 1$.
end if
for $k = 0$ to $\delta - 2$ **do**
  Δ_k = coeficiente de X^{k+1} em $S(X)\sigma_k(X)$.
  $\sigma_{k+1} = \sigma_k - \Delta_k X \tau_k$.
  $\omega_{k+1} = \omega_k - \Delta_k X \gamma_k$
  if ($\delta_k = 0$ or $D(k) > (k+1)/2$) or ($D(K) = (k+1)/2$ and $B(k) = 0$) **then**
    $D(k+1) = D(K)$; $B(k+1) = B(K)$.
  else
    if ($B(k-1) = 0$ and $D(k-1) \leq k - \lfloor \delta/2 \rfloor - 1$) or ($B(k-1) = 1$ and $D(k-1) \leq k - \lfloor (\delta-1)/2 \rfloor - 1$) **then**
      $\hat{\sigma} = X^{\delta-k}\sigma_{k-1}$; $\hat{\omega} = X^{\delta-k}\omega_{k-1}$; $\hat{N} = \delta - k + D(k-1) + 1 - B(k-1)$.
    end if
    $D(k+1) = k + 1 - D(k)$; $B(k+1) = 1 - B(k)$.
    $\tau_{k+1} = \sigma_k/\Delta_k$; $\gamma_{k+1} = \omega_k/\Delta_k$
  end if
end for
return $l(X)$

Algoritmo 4 Algoritmo de Patterson Modificado

Entrada: Polinômio síndrome $S(X) \in F[X]$.
Assegura: $\sigma(X)$ com $\deg \sigma(X) \leq \lfloor n/2 \rfloor$

Inicializar $w(X)$ com $w^2(X) \equiv X \bmod g(X)$
$T(X) \longleftarrow 1/S(X) \bmod g(X)$
if $T(X) = X$ **then**
  $\sigma(X) = X$
else
  $T_0^2(X) + XT_1^2(X) \longleftarrow T(X) + X$
  $R(X) \longleftarrow T_0(X) + w(X)T_1(X)$
  Usando **Algoritmo 2** com entradas $g(X)$ e $R(X)$ calcular $a(X)$ e $b(X)$ tal que a equação 2.10 seja satisfeita.
  return $\sigma(X) = a_0(X)^2 + Xa_1^2(X)$
end if

Algoritmo 5 WeakPopovForm

Entrada: $A \in F_q[X]^{p\times p}$.

Assegura: WeakPopovForm de A.

1: **for** $j \longleftarrow 1$ **to** p **do**
2: **if** $\deg(A_j, 1) > 0$ **then**
3: $I_j^A \longleftarrow 1$
4: **else**
5: $I_j^A \longleftarrow j$
6: **end if**
7: **end for**
8: **while** $p(I^A) > 1$ **do**
9: **for** $k \longleftarrow 1$ to p tal que $I_k^A \neq 0$ **do**
10: **for** $l \longleftarrow 1$ to p tal que $l = k$ **do**
11: **while** $\deg(A_{l,I_k^A}) \geq \deg(A_{k,I_k^A})$ **do**
12: $c \longleftarrow \text{lider}(A_{l,I_k^A})/\text{lider}(A_{l,I_k^A})$
13: $e \longleftarrow \deg(A_{l,I_k^A})\text{-}\deg(A_{l,I_k^A})$
14: $A_l \longleftarrow A_l - cx^e A^k$
15: **end while**
16: $d \longleftarrow \max\{\deg(A_{l,j})|j = 1, ..., p\}$
17: $I_l^A \longleftarrow \max\{j|\deg(A_{l,j}) = d\}$
18: **end for**
19: **end for**
20: **end while**
21: **return** A

Algoritmo 6 Algoritmo de Barreto

Entrada: $\Gamma(L, g(X))$, código Goppa sobre $\mathbb{F}_p$, onde $g(X)$ é livre de quadrados. A matriz de teste de paridade $H \in F_q^{r \times n}$. O vetor recebido $c' = c + e \in F_q^n$.

Assegura: vetor corrigido $c \in \Gamma(L, g(X))$.

1: $s^T \leftarrow Hc'^T \in \mathbb{F}_q^n$, $s(X) \longleftarrow \sum_i^{s_i} X^i$
2: **if** $\nexists s^{-1}(X) \bmod g(X)$ **then**
3: **return** vazio
4: **end if**
5: $S \longleftarrow$ vazio
6: **for** $\phi = 1$ **to** $p - 1$ **do**
7: **for** $k \longleftarrow 1$ **to** $p - 1$ **do**
8: $u_k(X) \longleftarrow X^k + \phi k X^{k-1}/s(X) \bmod g(X)$
9: **if** $\nexists \sqrt[p]{u_k(X)} \bmod g(X)$ **then**
10: provar o seguinte ϕ, isto só se $g(X)$ é composto.
11: **end if**
12: $v_k \longleftarrow \sqrt[p]{u_k(X)} \bmod g(X)$
13: **end for**
14: Construir o reticulado base A.
15: Aplicar *WeakPovovForm* para reduzir a base de $\Lambda(A)$
16: **for** $i = 1$ **to** p **do**
17: $a \longleftarrow A_i$; com entradas a_j onde j tem valores $0...p-1$
18: **for** $j = 1$ **to** $p - 1$ **do**
19: **if** $\deg(a_j) > \lfloor (t - j)/p \rfloor$ **then**
20: provar seguinte i; se não fosse solução.
21: **end if**
22: **end for**
23: $\sigma(X) \longleftarrow \sum_j X^j a_j(X)^p$
24: Computar o conjunto J tal que $\sigma(L_j) = 0$, $\forall j \in J$
25: **for** $j = 1$ **to** $p - 1$ **do**
26: Computar a multiplicidade μ_j de L_j
27: $e_j \longleftarrow \phi\mu_j$
28: **end for**
29: **if** $He^T = s^T$ **then**
30: $S \longleftarrow S \cup c' - e$
31: **end if**
32: **end for**
33: **end for**
34: **return** S

Algoritmo 7 Algoritmo Alternante

Entrada: **w** palavra recebida.

Assegura: vetor corrigido $c \in \Gamma(L, g(X))$.

1: Constrói-se o polinômio síndrome $s^T \leftarrow Hc'^T \in \mathbb{F}_q^n$, $s(X) \longleftarrow \sum_i s_i X^i$
2: **if** $s(X) = 0$ **then**
3: **return** $c = w$.
4: **end if**
5: Inicializar $r_{-1} = X^\delta$, $r_0 = S(X)$
6: Executar o algoritmo estendido de Euclides até encontrar $r_k(X)$ tal que: grau$(r_{k-1}(X)) \geq (1/2)\delta$ e grau$(r_k(X)) \leq (1/2)\delta - 1$.
7: $l(X) \longleftarrow z_k(0)^{-1} z_k(X)$ e
8: $a(X) = z_k(0)^{-1} r_k(X)$
9: Aplicar um algoritmo para encontrar as raízes, $\alpha_i = 1/L_i$, do polinômio localizador $l(X)$.
10: Inverter o valor na posição do vetor **w** recebido pelo decodificador.

Algoritmo 8 Algoritmo Berlekamp-Massey

Entrada: Polinômio síndrome $S(X) \in F[X], 0 < N$.

Assegura: Polinômio localizador $l(X)$

1: Inicializar $l_{-1}(X) = 0$; $l_0(X) = 1$;
2: $a_{-1}(X) = -X^{-1}$; $a_0(X) = 0$;
3: $\mu = -1$; $\delta_{-1} = 1$
4: **for** $i = 0$ to N **do**
5: $\delta_i =$ coeficiente de X^i em $l_i(X)b(X)$
6: $l_{i+1}(X) = l_i(X) - (\delta_i/\delta_\mu)X^{i-\mu} l_\mu(X)$
7: $a_{i+1}(X) = a_i(X) - (\delta_i/\delta_\mu)X^{i-\mu} a_\mu(X)$
8: **if** $\delta_i \neq 0$ and $2ord(l_i, a_i) \leq 1$ **then**
9: $\mu = i$
10: **end if**
11: **end for**
12: **return** $l(X)$

Capítulo 4

Sistemas Criptográficos Clássicos e Pós-Quânticos

Em 1976 Diffie e Hellman no seu trabalho "*New directions in cryptography*", mudaram os rumos da criptografia, criando os sistemas criptográficos de chave pública ou assimétricos, os quais utilizam um par de chaves: uma chave pública e uma chave privada como na Figura 4.1. Apareceram, logo, outros sistemas criptográficos assimétricos práticos, eficientes e seguros como: RSA, sistemas baseados em curvas elípticas, etc. Em 1994 e 1997 apareceram os algoritmos quânticos, de Shor e de Grover, respectivamente, os quais quebram alguns destes sistemas criptográficos.

Na obra "*Post-Quantum Cryptography*", de (Bernstein, Dahmen, e Buchmann, 2009), classifica-se os sistemas de criptografia em clássicos e pós-quânticos. Esta classificação é feita em função da aparente resistência, destes, ao ataque proveniente de algoritmos clássicos ou quânticos. Segundo esta classificação temos, por exemplo, que dentro dos sistemas de criptografia clássicos estão: RSA, Diffie-Hellman, sistemas baseados em curvas elípticas, etc, e dentro dos sistemas criptográficos, candidatos, pós-quânticos estão: McEliece, *N-th degree Truncated Polynomial Ring*-NTRU, etc. Estes sistemas têm suporte em problemas matemáticos difíceis de resolver, a saber:

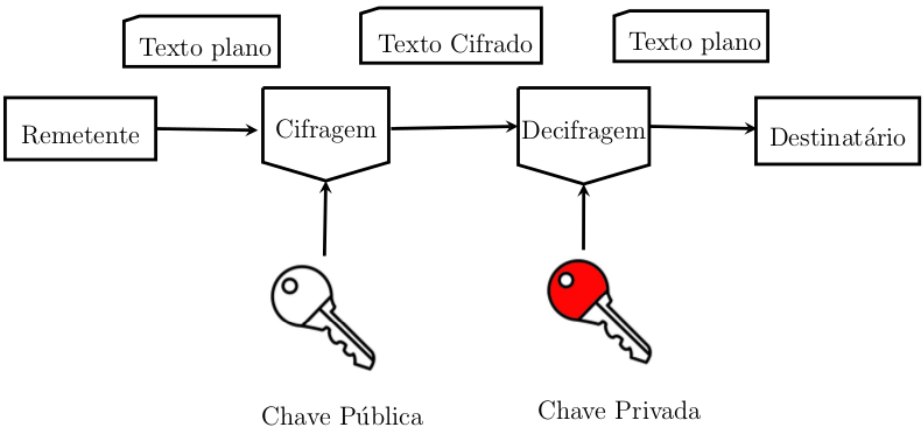

Figura 4.1: Criptografia de Chave Pública.

- IFP: Do inglês *Integer Factorizating Problem.* Tem sua segurança baseada no problema da fatoração de números inteiros;

- DLP: Do inglês *Discrete Logarithm Problem.* Tem sua segurança baseada no problema da determinação do logaritmo discreto;

- ECDLP: Do inglês *Elliptic Curve Discrete Logarithm Problem.* Tem sua segurança baseada no problema da determinação do logaritmo discreto em curvas elípticas;

- DSP: Do inglês *Decoding Syndrome Problem.* Tem sua segurança baseada no problema da decodificação num código linear randômico;

- SVP: Do inglês *Shortest Vector Problem.* Tem sua segurança baseada no problema de encontrar o vetor, não nulo, de menor peso num reticulado dado (veja Problema 2);

- CRHF: *Collision Resistance of that Hash Function.* Sua segurança depende da resistência de colisão das funções Hash.

Na continuação apresentamos duas tabelas, nestas mostramos os sistemas de criptografia mais representativos e o problema matemático no qual baseiam-se. A Tabela 4.1 para sistemas criptográficos clássicos e a Tabela 4.2 para os pós-quânticos.

IFS	DLS	ECDLS
RSA	ElGamal	ElGamal (curvas elípticas)
Rabin	Diffie-Helman	Diffie-Helman (curvas elípticas)
	Schnorr	

Tabela 4.1: Sistemas Criptográficos Clássicos e seus respectivos problemas matemáticos no qual baseiam-se.

DSP	SVP	CRHF
McEliece	NTRU	Merkle's Hash-tree
Niederreiter	CTRU	

Tabela 4.2: Sistemas criptográficos pós-quânticos e seus respectivos problemas matemáticos no qual baseiam-se.

O sistema criptográfico RSA é um dos de maior destaque dentro dos sistema clássicos, visto que até hoje é usado por ser eficiente e seguro. Na seção 4.1 apresentaremos este sistema e na seção 4.2 o McEliece, que é motivo interesse principal de nosso trabalho.

4.1 Criptografia RSA

O RSA, sistema criptográfico assimétrico foi desenvolvido por Ronald Rivest, Adi Shamir e Leonard Adleman, sendo que as inicias dos seus sobrenomes definem o nome do sistema. É dos mais populares e estudados devido ao fato que foi um dos primeiros sistemas criptográficos de chave pública inventados, e que ainda contínua em vigência, visto ser considerados um dos mais adaptados para uso cotidiano, devido à eficiência e segurança.

Basei-se no fato da dificuldade, aparente, de conseguir fatorar o produto de números primos, grandes. A seguir veremos os algoritmos que são usados neste sistema criptográfico, além de uma breve discussão da segurança deste sistema, tendo como referência Coutinho (1997).

4.1.1 Pré-codificação

Antes de começar com os processos de cifração e decifração tem-se que converter a mensagem numa lista de números. Para isso usaremos a seguinte tabela de conversão:

A	B	C	D	E	F	G	H	I	J	K	L	M
10	11	12	13	14	15	16	17	18	19	20	21	22

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
23	24	25	26	27	28	29	30	31	32	33	34	35

O espaço entre duas palavras será substituído pelo número 99, quando for feita a conversão. Por exemplo, a frase "Trujillo lindo" é convertida no número:

$$292730191821212499211823 1324$$

Antes de continuar precisamos determinar os parâmetros do sistema RSA que vamos usar. Estes parâmetros são dois números primos distintos, que vamos denotar por p e q. Ponha $n = pq$. A última fase do processo de pré-codificação consiste em quebrar o longo número produzido anteriormente em blocos. Estes blocos devem ser números menores que n. Por exemplo, se escolhermos $p = 11$ e $q = 13$, então $n = 143$. Neste caso, a mensagem, cuja conversão numérica foi feita acima, pode ser quebrada nos seguintes blocos:

$$25 - 102 - 7 - 102 - 93 - 49 - 91 - 49 - 92 - 118 - 23 - 13 - 10.$$

A maneira de escolher os blocos não é única.

4.1.2 Cifração e Decifração

Para codificar a mensagem precisamos de n, que é o produto dos primos, e de um inteiro positivo, e, que seja inversível módulo $\phi(n)$ [1] . Em outras palavras, $\text{mdc}(e, \phi(n)) = 1$. O $\phi(n)$ é fácil de calcular se conhecermos p e q, de fato

$$\phi(n) = (p-1)(q-1).$$

Chamaremos o par (n, e) de **chave pública** do sistema RSA. Tendo submetido a mensagem à pré-codificação, temos uma sequência de números ou blocos.

[1] $\phi(n)$ é o número de inteiros positivos menores que n e relativamente primos com n.

Codificaremos cada bloco separadamente e a mensagem codificada será a sequência dos blocos codificados. Vamos denotar o bloco codificado, b, por $C(b)$, sendo seu cálculo da seguinte forma:

$$b^e \equiv C(b)(\text{ mod } n).$$

Vejamos o que acontece no exemplo que estamos considerando. Temos $p = 11$ e $q = 13$, logo $n = 143$ e $\phi(n) = 120$. Ainda precisamos escolher o e. Neste exemplo, o menor valor possível para e é 7, que é o menor primo com 120 que não o divide. Assim, se tomarmos o segundo bloco da mensagem anterior, esse é codificado como o resto da divisão de 102^7 por 143, ou seja, $102^7 \equiv C(102)(mod\ 143)$. Fazendo as contas teremos:

$$102^7 \equiv 119(mod\ 143)$$

Codificando toda a mensagem, obtemos a seguinte sequência de blocos:

$$64 - 119 - 6 - 119 - 102 - 36 - 130 - 36 - 277 - 79 - 23 - 117 - 10$$

O processo necessário para a decodificação consiste de encontrar dois números: n e o inverso de e em $\phi(n)$, que denotaremos por d. Chamaremos o par (n, d) de **chave de decodificação (ou chave privada)**. Seja a um bloco da mensagem codificada, então $D(a)$ será o resultado do processo de decodificação. O cálculo de $D(a)$ é:

$$a^d \equiv D(a)(mod\ n).$$

Voltando ao exemplo, temos que $n = 143$ e $e = 7$. Aplicaremos o **algoritmo euclidiano estendido** para calcular o d então teremos

	17	7
120	7	1
1	0	

Assim,

$$1 = 120 + (-17) \cdot 7.$$

Ou seja, $7 \cdot (-17) \equiv 1(mod\ 120)$, logo o inverso de 7 módulo 120 é -17. Como vamos usar d como expoente de potências, precisamos que d seja positivo, para isso basta usar o menor inteiro positivo congruente a -17 módulo 120, 103.

Assim, para decodificar um bloco de conteúdo 119 da mensagem codificada teremos, $119^{103} \equiv D(119)(mod\ 143)$. Usando um sistema de computação algébrica, podemos verificar, que, de fato, $119^{103} \equiv 102(mod\ 143)$.

4.1.3 Segurança

Um ponto muito importante para a segurança, refere-se à escolha dos primos p e q. Se esses fossem pequenos, o sistema seria fácil de quebrar, também não basta serem grandes já que se p e q são grandes, e $|p - q|$ é pequeno então é fácil fatorar $n = pq$ usando o algoritmo de Fermat (veja capítulo 2 em Coutinho (1997)).

Em 1995 dois estudantes de uma universidade americana quebraram uma versão do RSA em uso público. Isso só foi possível porque a escolha dos números primos usados neste sistema era inadequada. Mas se os primos forem bem escolhidos, o sistema mostra-se bastante seguro.

Suponha que desejamos implementar o RSA com chave pública (n, e), de modo que n seja um inteiro com aproximadamente r algarismos. Para construir n, escolha um primo p, que tenha, entre $4r/10$ e $45r/100$ algarismos e, em seguida escolha q próximo de $10^r/p$. O tamanho da chave recomendado para uso pessoal é 768 bits (aproximadamente 231 algarismos). Para construir um n precisaremos de dois primos de 104 e 127 algarismos. Note que esses primos estão longe o bastante um do outro pra fazer com que a fatoração de n pelo algoritmo de Fermat seja impraticável. Contudo, precisamos saber se os números $p-1$, $q-1$, $p+1$ e $q+1$ não tem fatores primos pequenos, porque isto faria de n uma presa fácil para alguns algoritmos de fatoração conhecidos. Vejamos então como obter primos grandes.

Definição 4.1.1. *Seja $x \in \mathbb{N}$ denote como $\pi(x)$ o número de primos positivos*

menores ou iguais a x.

De acordo com o Teorema dos Números Primos, se x é suficientemente grande, então $\pi(x)$ é aproximadamente igual a $x/\ln x$. Seja agora x um número muito grande e ϵ um número positivo qualquer. Queremos obter uma aproximação para a quantidade de primos entre x e $x+\epsilon$, isto é, $\pi(x+\epsilon)-\pi(x)$. Segue, do teorema dos números primos e das propriedades do logaritmo, que $\pi(x+\epsilon)-\pi(x)$ é aproximadamente igual a

$$\frac{x+\epsilon}{\ln x + \ln(1+x^{-1}\epsilon)} - \frac{x}{\ln x}$$

Supondo que $x^{-1}\epsilon$ é suficientemente pequeno, podemos substituir $\ln(1+x^{-1}\epsilon)$ por zero e ainda assim obter uma aproximação razoável para $\Pi(x+\epsilon)-\Pi(x)$. Concluímos que o número de primos entre x e $x+\epsilon$ é aproximadamente igual $\epsilon/\ln x$. É claro que, quanto maior for x, menor será $x^{-1}\epsilon$, e melhor será a aproximação.

Suponha, agora, que desejamos achar um primo próximo de um inteiro positivo x, para tornar a discussão mais concreta, digamos que x é da ordem de 10^{127}. Procuraremos por este primo no intervalo que vai de x a $x+10^4$. Precisamos saber quantos primos há nesse intervalo, é aqui que o resultado do parágrafo anterior vem em nosso auxílio. Nesse exemplo, $x^{-1}\epsilon$ é da ordem de 10^{-123} e, portanto, é bem pequeno. Assim, usando a fórmula acima, concluímos que no intervalo deve haver aproximadamente 34 primos.

$$\lfloor 10^4/\ln(10^{127}) \rfloor = 34$$

Para saber se um dado inteiro ímpar n é primo podemos proceder da seguinte maneira:

(1) Verifique se n é divisível por um primo menor que 5000;

(2) Supondo que n não é divisível por nenhum destes primos, aplique o teste de Miller (veja seção 3 Capítulo 6 de Coutinho (1997)) a n usando como bases os primeiros 10 primos;

(3) Supondo que o teste de Miller teve uma saída inconclusiva para todas essas bases, aplique o teste de primalidade a n.

Adaptaremos esta estratégia para achar um número primo no intervalo que vai de x a $x + 10^4$. Em primeiro lugar, elimine do intervalo dado os inteiros ímpares que são divisíveis por primos menores que 5000. Em seguida, aplique (2) aos números remanescentes até que um primo seja encontrado.

Para descobrir quanto trabalho teremos para obter este primo, podemos tentar determinar quantos inteiros sobraram depois que eliminamos aqueles que são divisíveis por primos menores que 5000. Seja m um inteiro positivo. Se $x \leq km \leq x + 10^4$, então

$$\left\lfloor \frac{x}{m} \right\rfloor \leq k \leq \left\lfloor \frac{x + 10^4}{m} \right\rfloor .$$

Assim, há

$$\left\lfloor \frac{x + 10^4}{m} \right\rfloor - \left\lfloor \frac{x}{m} \right\rfloor$$

múltiplos de m no intervalo que vai de x a $x+10^4$. Este número é aproximadamente igual a $\lfloor 10^4/m \rfloor$, que é aproximadamente o número de inteiros que são múltiplos de primos positivos menores que $5 \cdot 10^3$ nos intervalos $[x, x + 10^4]$ e $[0, 10^4]$. Não é fácil calcular o último desses números. Em primeiro lugar, um número composto menor que 10^4 tem que ser múltiplo de um primo menor que $\sqrt{10^4} = 100$. Assim, um inteiro no intervalo $[0, 10^4]$ é múltiplo de um primo menor que $5 \cdot 10^3$ se é composto, ou se é ele próprio um primo menor que $5 \cdot 10^3$. Levando em conta que 2 é o único primo par, temos que o número de **inteiros compostos e ímpares** menores que 10^4 é $5000 - \pi(10^4) + 1$. Dessa forma, o número total de inteiros ímpares que restam no intervalo de x a $x + 10^4$ após a eliminação daqueles que são divisíveis por primos menores que $5 \cdot 10^3$ é aproximadamente igual a

$$5000 - (5000 - (\pi(10^4) - 1)) - (\pi(5 \cdot 10^3) - 1) = 560.$$

Observe que $\pi(10^4)$ e $\pi(5 \cdot 10^3)$ são facilmente calculados usando o crivo de

Eratóstenes.

Assim, esperaríamos encontrar, em média 34 primos de um total de 560 inteiros deixados após o processo inicial de eliminação.

4.2 Sistema Criptográfico McEliece

Este sistema criptográfico assimétrico foi inventado por McEliece (1978), e usa os códigos corretores de erros. A proposta original sugere os códigos de Goppa binário como o suporte matemático para seus algoritmos. Sua segurança vem determinada pelo dificuldade de decodificar um código linear aleatório, que é um problema NP-difícil (como indicado na Tabela 4.2). Uma das características mais importantes deste sistema criptográfico é o baixo custo computacional no que se refere a seus algoritmos de cifração e decifração em comparação com outros sistemas consolidados, por exemplo o sistema RSA. Outra de suas características fortes é sua segurança. Mas tem uma desvantagem, que o faz impraticável, esta é o tamanho de suas chaves, a qual é consideravelmente grande em comparação com RSA, sistemas baseados em curvas elípticas, etc. Daí, que muitos pesquisadores voltaram-se a estudar a redução destas, como por exemplo Misoczki e Barreto (2009).

4.2.1 Descrição

Como vimos no sistema criptográfico RSA, descreveremos aqui os algoritmos de geração de suas chaves, cifração e decifração.

Geração de suas Chaves

Aqui veremos como são gerados o conjunto de chaves, publica e privada, os quais são usados no processo de cifração e decifração.

(1) Alicia seleciona um código de Goppa binário Γ com parâmetros (n, k) e capacidade de correção de δ erros;

(2) Alicia gera a matriz geradora G, de dimensões $k \times n$, do código Γ;

(3) Seleciona uma matriz binária não singular S, de dimensões $k \times k$;

(4) Seleciona uma matriz de permutação P, de dimensões $n \times n$;

(5) Computa a matriz $\hat{G} = S \cdot G \cdot P$;

(6) A chave pública de Alicia é $(\hat{G}, \delta)$ e sua chave privada é (S, G, P).

Cifração

Suponha que Bob envia um mensagem x para Alicia cuja chave pública é $(\hat{G}, \delta)$:

(1) Faz uma pre-codificação da mensagem $x = x_1, x_2, \ldots$, como na Seção 4.1.1 com comprimento do bloco x_i igual a k;

(2) Computa o vetor $c' = x_i\hat{G}$, para cada i;

(3) Gera um vetor, e, aleatorio de comprimento n; $w(e) = \delta$;

(4) Computa o texto criptografado como $c = c' + e$.

Decifração

Neste ponto, Alicia com a chave privada e o vetor recebido faz:

(1) Computa o inverso de P, isto é P^{-1};

(2) Computa $\hat{c} = cP^{-1}$;

(3) Usa um algoritmo decodificador para o código Γ, para decodificar $\hat{c}$ em $\hat{x}_i$;

(4) Computa $x_i = \hat{x}_i S^{-1}$, para cada i.

Do passo (2) temos que, $\hat{c} = cP^{-1} = (c' + e)P^{-1} = x_i\hat{G}P^{-1} + eP^{-1} = x_i S \cdot G \cdot P \cdot P^{-1} + eP^{-1} = x_i S \cdot G + eP^{-1}$. Veja que $w(eP^{-1}) = \delta$, devido a que P é só uma matriz que permuta as entradas de e. Assim aplicando o algoritmo decodificador para o código Γ, na expressão anterior, temos que $\hat{c}$ se decodifica em $\hat{x}_i = x_i S$. Multiplicando por S^{-1}, nos dos lados da expressão anterior temos que $\hat{x}_i S^{-1} = x_i$.

4.2.2 Segurança

Existem basicamente dois tipos de ataque contra dois aspectos de segurança neste sistema criptográfico. Estes aspectos baseiam-se em problemas difíceis dos códigos lineares, a saber: a Indistinguibilidade dos códigos de Goppa e o problema da decodificação de um código linear binário (NP-difícil). A seguir apresentamos estes aspectos de segurança e logo um dois ataques teóricos mais relevantes contra o McEliece.

Indistinguibilidade dos códigos de Goppa.

Este aspecto refere-se, a segurança na estrutura das chaves do sistema. Existem algumas estrategias para disfarçar a estrutura de um código de Goppa. Na continuação, pode-se ver as diferentes propostas de modificação do sistema McEliece, para este fim (veja seção 2.1 em Bernstein et al., 2009):

(1) *Row Scrambler*: Multiplicar G, pela direita, por uma matriz S inversível. Assim, $\langle G\rangle = \langle SG\rangle$.

(2) *Column Scrambler / Isometry*: Multiplicar G, pela esquerda, por uma matriz, $T^{n\times n}$, inversível aleatória, onde T preserva a norma (peso de Hamming), obviamente pode-se corrigir erros, nesta norma até δ em GT, se G e T são conhecidos.

(3) Sub-código: Seja $0 < l < k$. Multiplicar G por uma matriz aleatória $S \in \mathbb{F}^{l\times k}$ de posto de linha completa. Isto é $SG \subseteq G$, assim um algoritmo de correção de erros, conhecido, pode ser usado.

(4) Sub-código Sub-corpo: Tomar o sub-código sub-corpo, $C|_F$, de um código secreto para um subcorpo $\mathbb{F}_m$ de F, veja Definição 2.4.1. Assim, pode-se corrigir erros com um algoritmo de decodificação para o código secreto.

(5) Concatenação de Matriz: Tomar o código $\langle [G|SG]\rangle$, onde $S \in F^{k\times k}$ é uma matriz inversível. Na norma de Hamming, o dono da chave privada pode

corrigir erros $2\delta + 1$ erros neste código, já que ele pode corrigir erros na primeira, n colunas ou nas demais.

(6) Erros artificiais: Pode-se optar por modificar a matriz G num pequeno número de posições. Essas posições serão tratados como rasuras na decifração e, assim, mudar a capacidade de correção de do código.

(7) Códigos redutíveis: Escolha as matrizes $Y \in \mathbb{F}^{k \times n}$ e $S \in \mathbb{F}^{k \times n}$ com $l \leq k$. Depois, tomar o código gerado por

$$\begin{bmatrix} SG & 0 \\ Y & G \end{bmatrix}.$$

A correção de erros é possível se corrige-se erros em seções, começando pela direita. No entanto, para a correção de erros na métrica de Hamming, esta abordagem não parece ser adequada.

A proposta para disfarçar o código de Goppa usado no sistema criptográfico McEliece original, é uma combinação de (1), (2) e (4). Todos os ataques conhecidos, neste aspecto, tem uma complexidade exponencial e são denominados de ataques estruturais. O melhor ataque estrutural conhecido é *support splitting algorithm*, apresentado por Sendrier (2000). Os ataques estruturais nunca são melhores que os ataques que baseiam-se no problema a seguir.

Problema de decodificação de um código linear binário.

O Problema 1, decodificação de um código linear binário, está fortemente relacionado com o problema de encontrar palavras de peso mínimo, Problema 2:

Problema 1. Dada uma matriz binária $H^{r \times n}$ e uma palavra $\mathbf{s} \in \mathbb{F}_2^r$, encontrar uma palavra $\mathbf{x}$; $w(\mathbf{x}) \leq r/log_2 n$ na classe lateral de $\mathbf{s}$. Onde $r = \delta \cdot m$ e $n = 2^m$.

Problema 2. Dada uma matriz binária $H^{r \times n}$ e um inteiro $\omega > 0$, encontrar uma palavra $\mathbf{x}$, não nula; $w(\mathbf{x}) \leq \omega$ na classe lateral de $\mathbf{0}$. Onde $r = \delta \cdot m$ e $n = 2^m$.

Os algoritmos para a solução do problema acima (com algumas modificações), podem ser usados para solucionar o **Problema 1**. Esses algoritmos são

considerados como algoritmos de ataque para o sistema McEliece (na mensagem). Os melhores algoritmos desse tipo de ataque baseiam-se no algoritmo "*Information Set Decoding*". Lembre que *Information Set* é definido como:

Definição 4.2.1. *Dado um código linear C com parâmetros (n, k) e sejam as k posições $i_1, i_2, \ldots, i_k$ com a propriedade que ao restringir as palavras códigos a estas posições, obtêm-se as q^k palavras de comprimento k. O conjunto $\{i_1, i_2, \ldots, i_k\}$ é denominado Information Set.*

Exemplo. Seja a mensagem $[0, 1, 2, 3] \in \mathbb{Z}_4^4$ codificado como $[0, 3, 2, 0, 3, 1]$, então o conjunto de informação é $\{4, 6, 3, 2\}$, com $q = 4$, $n = 6$, e $k = 4$.

O algoritmo "*Information Set Decoding*" consiste basicamente em selecionar, iterativamente, k-bits que denominaremos de c_k, num vetor c de n-bits, transmitido, tendo como esperança que nos k-bits selecionados não exista nenhum bit de erro. Se não existe erro em c_k, então $c_k\hat{G}_k^{-1}$ é a mensagem x_i, na Subseção **cifração** da Seção 4.2. A matriz $\hat{G}_k$, de dimensões $k \times k$, foi obtida selecionando k colunas de $\hat{G}$, de acordo com a seleção de c_k. O custo total do algoritmo, é dado por:

$$W = \alpha k^3 \frac{\binom{n}{k}}{\binom{n-\delta}{k}}$$

onde α é uma constante pequena que usualmente é 1. O custo total de algoritmos deste tipo é chamado de *work factor*. O custo, em operações binárias, de uma iteração dividida pelo *work factor*, denomina-se de *binary work factor*.

Em 1989 Lee and Brickell apresentam uma modificação do algoritmo anterior, além de um modo de verificar se a palavra de k-bits encontrada é ou não a mensagem original transmitida.

A modificação feita consiste em selecionar randomicamente, em cada iteração, um *Information Set* e examinar todas as palavras código de peso j, com j pequeno, no vetor c_k dentro do *Information Set*; j é usualmente 1 ou 2. Reduzindo assim, significativamente, o *binary work factor*, $W \approx 2^{84}$, do algoritmo "*Information Set Decoding*", para $W \approx 2^{73}$, caso o código de Goppa tenha parâmetros

$n = 1024$ e $t = 50$. O *work factor* para este algoritmo esta dado pela seguinte expressão:

$$W_j = T_j(\alpha k^3 + N_j \beta k);\ \text{com}\ N_j = \sum_{i=0}^{j} \binom{k}{i}\ \text{e}\ T_j = \frac{\binom{n}{k}}{\sum_{i=0}^{j} \binom{\delta}{i}\binom{n-\delta}{k-i}}$$

onde α e β são duas constantes ''pequenas".

Na variante de Leon (1988), em vez de computar todos os pesos das palavras-código selecionadas, o autor sugere procurar as palavras código, c, com peso ω (onde ω é uma constante ''pequena"), tais que para um parâmetro, l, as sub-palavras código de comprimento l em c tenham peso 0. Com esta modificação consegui-se melhorar o *work factor* significativamente.

Stern (1989) apresenta outra variante do algoritmo "*Information Set Decoding*", parecida à proposta por Leon. Mas a proposta que obtém destaque é a apresentada por Canteaut e Chabaud (1998), sendo um dos melhores ataques conhecidos até agora.

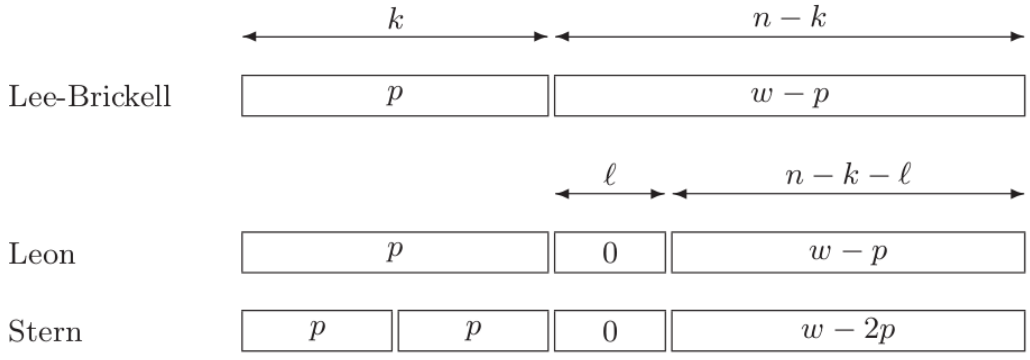

Figura 4.2: Perfil de peso das palavras-código procurado pelos algoritmos propostos por Lee-Brickell, Leon e Stern (os valores dentro das caixas são os pesos de Hamming)(Figura extraída de Bernstein et al. (2009)).

Algoritmo de Canteaut-Chabaud

Esta variante foi proposta por Canteaut-Chabaud e basicamente é o algoritmo de Stern com uma melhoria proposta por Tilburg (1990).

Ataque do texto cifrado escolhido.

Antes de descrever este ataque precisamos conhecer o que é maleabilidade. A maleabilidade é uma propriedade de alguns esquemas criptográficos que consiste em obter um texto cifrado válido a partir de outro texto cifrado conhecido. Por exemplo, o sistema McEliece tem esta propriedade. De fato a partir do teste cifrado $c = mG + e$ podemos obter o texto cifrado válido $c' = mG + m'G + e$.

Suponha que um atacante quer descifrar o texto cifrado $c_1 = m_1G+e$. Devido à maleabilidade e supondo que ele, ou ela, tem um oráculo aleatório que consegue descifrar textos cifrados (sem ser c) o atacante, ou a atacante, pode recuperar a mensagem m_1, ou equivalentemente e da seguinte forma. Primeiro o atacante, ou a atacante, precisa adicionar uma combinação linear m_2G a c_1, ou seja ele, ou ela, vai obter $c_2 = m_1G + m_2G + e$. Devido à maleabilidade c_2 é um texto cifrado válido. Usando o oráculo o atacante pode descifrar c_2 obtendo $m_1 \oplus m_2$. Após isso o atacante pode aplicar o ataque *Related-Message Attack* (**?**) para descobrir m_1.

4.3 Sistema Criptográfico Niederreiter

Niederreiter (1986) propos uma variação, dual ao sistema criptográfico McEliece, o qual mostrou-se, em Ding et al. (2007), que tem segurança equivalente. Este sistema criptográfico tem como vantagem a redução do tamanho da chave pública, contudo seus algoritmos de cifração e decifração são muito demorados. Este sistema criptográfico apresenta como novidade a redução do tamanho das chaves, quando usa-se os códigos generalizados de Reed Salomoon (GRS) como suporte matemático. Em 1992 Sidelnikov e Shestakov, no seu trabalho Sidelnikov e Shestakov (1992), mostraram que aquela novidade traiz como consequência a insegurança do sistema.

4.3.1 Descrição

Os algoritmos de geração de chaves, cifração e decifração são parecidos aos do McEliece, como veremos a continuação.

Geração de suas Chaves

Aqui, veremos como são geradas as chaves publica e privada, as quais são usadas no processo de cifração e decifração.

(1) Usuário seleciona um código de Goppa binário Γ, com parâmetros (n, k) e capacidade de correção de δ erros;

(2) Alicia gera a matriz de teste de paridade H, de dimensões $k \times n$, do código Γ;

(3) Seleciona uma matriz binária não singular S, de dimensões $k \times k$;

(4) Seleciona uma matriz de permutação P, de dimensões $n \times n$;

(5) Computa a matriz $\hat{H} = S \cdot H \cdot P$;

(6) A chave pública de Alicia é $(\hat{H}, \delta)$ e sua chave privada é (S, H, P).

Cifração

Suponha que Bob envia um mensagem x para Alicia cuja chave pública é $(\hat{H}, \delta)$:

(1) Bob faz uma pre-codificação da mensagem $x = x_1, x_2, \ldots$, como na seção 4.1.1, com o comprimento do bloco x_i igual a n;

(2) Bob codifica cada mensagem x_i como um vetor binário de comprimento n e $w(x_i) = \delta$;

(3) Bob computa o vetor $c = \hat{H}x_i^T$, para cada i.

Decifração

(1) Alicia computa o inverso de S, isto é S^{-1};

(2) Alicia computa $S^{-1}c = HPx_i^T$;

(3) Alicia aplica um algoritmo de decodificação para códigos de Goppa gerados pela matriz G, para recuperar Px_i^T;

(4) Alicia computa a mensagem x_i via $x_i^T = P^{-1}Px_i^T$.

4.4 Assinaturas Digitais

Por definição, um sistema de assinatura digital deve fornecer uma maneira de assinar qualquer documento de maneira que o autor seja identificado de forma única, além de dispor de um algoritmo público eficiente de verificação de assinatura.

Existiram algumas tentativas para fazer assinaturas, baseadas em teoria de códigos praticas, eficientes e seguras, como amostrados em Stern (1993, 1994); Courtois et al. (2001). O problema destas tentativas, tinha a ver, especialmente, no passo (1) da seção Assinatura abaixo. Isto é, porque nem sempre s era uma síndrome decodificável, já que uma síndrome gerada randômica, usualmente, corresponde a um vetor com peso de Hamming maior que δ. Em (Courtois, Finiasz, e Sendrier, 2001) se propõe um esquema, baseado no sistema Niederreiter, tal que seja possível utilizar na pratica. Para solucionar o problema, da geração randômica da síndrome, este sistema corrige um numero, fixo, adicional de erros λ. Assim, para decodificar uma síndrome correspondente a um vetor com peso de Hamming $\delta + \lambda$ adiciona-se randomicamente λ colunas na matriz de teste de paridade H, e tenta-se decodificar outra vez a nova síndrome. Se conseguir decodificar a nova síndrome, então esta é correspondida com uma palavra de peso δ. Se não tem que gerar randomicamente outras colunas até encontrar uma síndrome decodificável.

Um esquema de assinatura, baseado em códigos de Goppa, como mencionado acima, pode-se descrever nos seguintes passos.

Geração de suas chaves

(1) Escolher um código de Goppa $\Gamma(L, g(X))$;

(2) Obter G e H, a matriz geradora $k \times n$ e a matriz $(n-k) \times n$ de verificação de paridade para este código, respectivamente;

(3) Calcular $V = SHP$, onde S é uma matriz binária inversível $(n-k)\times(n-k)$ aleatória e P uma matriz de permutação aleatória $n \times n$. Assim, a chave privada seria G e a chave pública (V, δ).

Assinatura

(1) Encontrar o menor $i \in \mathbb{N}$ tal que, para $c = h(m, i)$ e $s = S^{-1}c$ seja uma síndrome decodificável do código Γ;

(2) Usando o algoritmo de decodificação de Γ, obter o vetor de erros e, cuja síndrome seja s , ou seja $s = He^T$;

(3) Obter $e^T = P^{-1}e'^T$. Assim, a assinatura e o par: (e, i).

Verificação da assinatura

(1) Obter $\mathbf{c} = V\mathbf{e}^T$;

(2) Aceite somente se $\mathbf{c} = h(m, i)$.

No pior dos casos a assinatura, é feita usando $O(t!)$ passos.

Capítulo 5

Implementações e Resultados

Esta Capítulo está dividido em duas partes. Na Seção 5.1 apresentamos os dados, métodos, funções e classes que implementam a geração randômica de códigos de Goppa binário irredutível e alternante; e os algoritmos descritos no Capítulo 3. Na Seção 5.2 apresentamos os experimentos, as comparações e os resultados conclusivos do nosso trabalho. A menos que seja indicado, neste capítulo, quando mencionamos códigos de Goppa binário, nos referiremos a códigos de Goppa binário irredutíveis.

5.1 Implementação

SAGE é um sistema algébrico computacional (cuja sigla em inglês é CAS) escrito no Python e uma versão modificada de Pyrex (chamada inicialmente SageX e posteriormente Cython).

As implementações encontram-se feitas de forma híbrida, usando o software HyMES, desenvolvido por (Biswas, 2010) e o CAS SAGE para a implementação dos algoritmos descritos no Capítulo 3. Usamos algumas estruturas e alguns métodos modificados do software HyMES para a geração randômica dos códigos de Goppa binário e alternante, os quais são o suporte para o sistema McEliece. O software HyMES nos fornece, também, a implementação de um parâmetro importante, o *binary work factor* para um dos melhores ataques conhecidos, o algoritmo Canteaut-Chabaud (Biswas, 2010, Seção 2.4), para as comparações detalhadas na

Seção 5.2. Para levar a cabo a integração, foi preciso compilar o código escrito na linguagem C, criando um módulo, de nome *randomGoppa.so*, para Python versão 2.6, o qual pode ser usado no SAGE fazendo *from randomGoppa import **.

O computador usado, para fazer os experimentos, têm sistema operacional Suse 11.1, arquitetura $x86\_32$, processador de $2500MHz$ com 7 gigabytes de memoria RAM.

Do ponto de vista da implementação do sistema McEliece, é mais adequado trabalhar com uma permutação do suporte L, e assim transformar a matriz G, da Seção 4.2, na forma padrão. Como mencionado em (Hoffmann, 2011, Seção 4), S não tem nenhuma função em esconder o polinômio secreto $g(X)$. Portanto, na pratica que podemos nos livrar das matrizes S e P. Logo temos:

- Chave Privada: (L, g), onde L é uma permutação do suporte L e $g(X)$, com $\deg(g(X)) = \delta$, é o polinômio de Goppa, do código descrito pela matriz geradora G' com parâmetros $[n, k, d]$, $d \geq 2\delta + 1$.

- Chave Pública: (G, δ), onde G é a matriz geradora de um código equivalente, na forma padrão, do código gerado por G'.

Apresentamos a seguir os métodos mais importantes usados na implementação do sistema criptográfico McEliece, usando códigos de Goppa binário e alternante, gerados de forma randômica.

5.1.1 Implementação Randômica dos Códigos de Goppa Binário e Alternante

Para a implementação dos códigos de Goppa binário e alternante, de forma randômica, nos baseamos no software HyMES. Esta implementação consta de duas partes. Na Seção 5.1.1.1 apresentamos a geração do corpo F_{2^m} e na Seção 5.1.1.2 a geração do código de Goppa binário e alternante de forma randômica.

5.1.1.1 Geração do Corpo Finito

A documentação da implementação do corpo finito $F = K[X]/p(X)$, onde $p(X) \in K[X]$ é um polinômio irredutível, pode ser encontrada na Seção 6 em (Hoffmann, 2011).

5.1.1.2 Geração do Código de Goppa

O primeiro passo para gerar o código de Goppa binário é descrevê-lo mediante sua matriz geradora e de teste de paridade. A seguir especificaremos a implementação das principais funções usadas para a geração destas matrizes.

Para as notações, estruturas de dados e outras definições que aparecem a seguir veja (Hoffmann, 2011).

binmat_t key_genmat(gf_t * L, poly_t g, int ** perm, int * $\hat{H}$)**

Função que gera o polinômio de Goppa $g(X)$, de forma randômica, o suporte $L = \mathbb{F}|_0$. Com isto a função gera, também, as matrizes H, $\hat{H}$ e matriz G na forma padrão.

poly_t keypair(poly_t * g, gf_t **L)

Uma vez gerado L, $g(X)$ e G, na forma padrão, esta função armazena as chaves, pública e privada, segundo o ponto de vista da implementação mencionada anteriormente.

int * generateError()

Função que gera um vetor, **e**, error aleatório com entradas em $\mathbb{F}_2$.

int * encrypt(int * cw, int * e, binmat_t R)

Função que cifra uma mensagem, **cw**, adicionando um vetor erro, **e**, com a chave pública **R**.

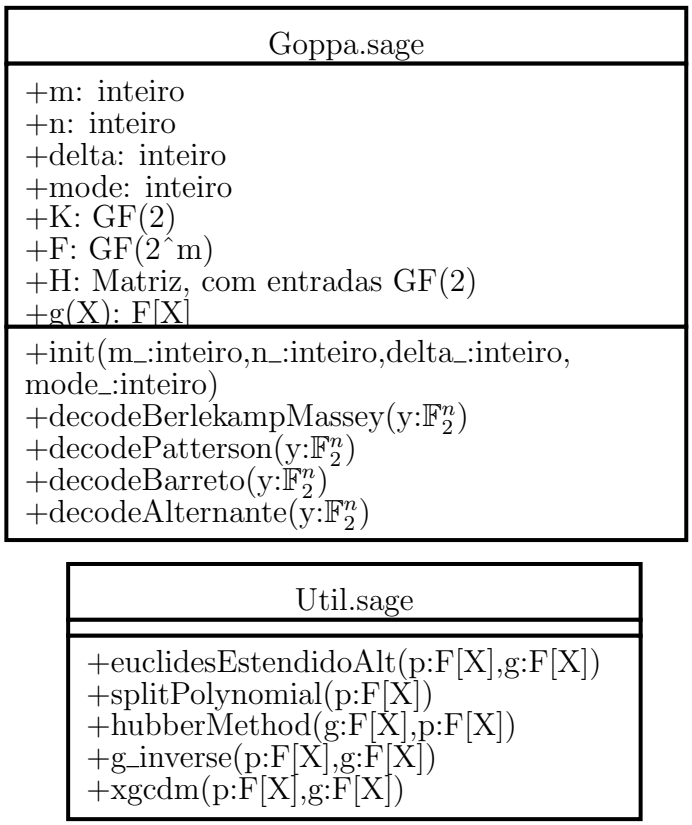

Figura 5.1: Diagrama de Classes, para a Implementação dos Algoritmos de Decodificação segundo o Capitulo 3.

5.1.2 Implementação dos Algoritmos de Decodificação para Códigos de Goppa Binário e Alternantes

Para a implementação dos algoritmos descritos no Capítulo 3, desenvolvemos a classe Goppa. Esta classe, feita no SAGE, implementa também o corpo finito $F = \mathbb{F}_2$ e o anel de classes residuais de polinômios $F[X]$, segundo os parâmetros a seguir: n, comprimento das palavras, m, grau do polinômio irredutível, $p(X)$, usado para gerar a extensão $F = K[X]/p(X)$ e δ, o grau do polinômio de Goppa. Estes parâmetros são os mesmos usados para a geração do código de Goppa randômico visto na seção anterior.

***Class* Goppa(self, n_,m_,delta_,mode_)**

__init__(self, n_,m_,delta_,mode_)

Instancia um objeto para gerar um código de Goppa e/ou alternante aleatório.

Parâmetros: **n_** – (*integer*) comprimento das palavras código

m_ – (*integer*) grau do polinômio irreduzivel $p(X)$ em $K[X]/p(X)$

delta_ – (*integer*) grau do polinômio de Goppa

mode – (*integer*) *flag* igual ao 1 ou 0(1, Alternante; 0, Goppa)

Exemplo.

```
>>gCode = Goppa(self, n_,m_,delta_,mode_)
```

decodeBerlekampMassey(self,y)

Decodifica o vetor y que chega ao decodificador segundo Berlekamp-Massey (1969), **Algoritmo** 8.

decodePatterson(self,y)

Decodifica o vetor y que chega ao decodificador segundo Patterson (1974), **Algoritmo** 4.

decodeBarreto(self,y)

Decodifica o vetor y que chega ao decodificador segundo Barreto et al. (2011), **Algoritmo** 6.

decodeAlternante(self,y)

Decodifica o vetor y que chega ao decodificador segundo o algoritmo alternante apresentado por MacWilliams e Sloane (1997), **Algoritmo** 7.

***Class* Útil**

Esta é uma classe estática, que implementa diversos algoritmos auxiliares, que são usados pelos algoritmos implementados na classe Goppa.

simulation()

Método para configurar nossos experimentos.

hubberMethod(g,t)

Implementa o método proposto por Hubber (1996), para resolver o radical da Equação (2.13), modulo g (veja seção 3.2).

split(p)

Método para obter a parte par e parte ímpar de um polinômio $p(X)$.

g_inverse(p,g)

Obtêm o inverso multiplicativo do polinômio $p(X)$ modulo $g(X)$.

xgcdm(p, q)

Dado dois polinômios $p, q \in F[X]$, este método implementa o **Algoritmo** 2, ou seja, ela retorna os polinômios d, s, t, os quais satisfazem $d = s{\cdot}p{+}t{\cdot}q$. Normalmente este algoritmo é executado até que a variável interna, V3, seja igual a zero, e o inverso de $p(X)$ é retornado. Mas neste caso, o algoritmo para quando $\deg(V3) < \lfloor \deg(p(X))/2 \rfloor$.

euclidesEstendidoAlternante(p, q)

Este método implementa o passo 6. do **Algoritmo** 7.

5.2 Experimentos

Dois tipos de experimentos foram feitos neste trabalho. O primeiro consiste num estudo comparativo das complexidades assintóticas do algoritmo de decifração do sistema McEliece, que resulta do uso dos algoritmos propostos no Capítulo 3. O segundo consiste num estudo comparativo dos desempenhos das complexidades mencionadas tendo como parâmetro importante o *work factor* do algoritmo de Canteaut e Chabaud (1998). É importante mencionar que este último experimento é uma extensão do experimento feito em (Biswas, 2010, Seção 2.4), com 3 algoritmos mais, alternante (MacWilliams e Sloane, 1997, cap. 12.9), Barreto et al. (2011) e Berlekamp-Massey (1969), além do Patterson.

Nestes experimentos limitamos a capacidade de correção dos algoritmos Patterson (1974) e Barreto et al. (2011) para poder igualar a capacidade de correção dos algoritmos Berlekamp-Massey (1969) e o algoritmo alternante (MacWilliams e Sloane, 1997, cap. 12.9), os quais tem capacidade $\lfloor \delta/2 \rfloor$. As siglas *bwf* abaixo é o logaritmo em base 2, do *work factor* do algoritmo de Canteaut e Chabaud (1998).

Nos experimentos das Tabelas 5.1 e 5.2 (cujos respectivos gráficos são as Figuras 5.2 e 5.3), os valores nas colunas [PAT], [BAR], [BMA] e [ALT] são o logaritmo em base 2, do tempo medido em segundos dos algoritmos Patterson (1974), Barreto et al. (2011), Berlekamp-Massey (1969) e o algoritmo alternante (MacWilliams e Sloane, 1997, cap. 12.9) respectivamente (os tempos aqui considerados é uma media entre 5 execuções). Nessas tabelas, a coluna denominada tamanho, contém a quantidade de *kilobytes* necessários para armazenar a chave pública do sistema McEliece. Nessa coluna, os valores que aparecem com '-' é porque o software HyMES não fornece as chaves do sistema com parâmetros (m, δ).

Na Tabela 5.3 apresentamos a complexidade assintótica do algoritmo de decifração do sistema McEliece e do algoritmo de decifração do sistema RSA. O experimento da Figura 5.2, confirma o primeiro valor dessa tabela, quando o *binary work factor*, *(bwf)*, cresce ordenadamente como na Tabela 5.1. No experimento da Figura 5.3 ocorre um comportamento anômalo, como pode ser visto na linha 7.

Assim, as complexidades assintóticas dos algoritmos de decifração, usando [ALT], [BAR] e [PAT], do sistema McEliece não coincidem com a complexidade assintótica apresentada na Tabela 5.3. As complexidades assintóticas mencionadas anteriormente são aquelas, que resultam da implementação do algoritmo de decifração usando os algoritmos de decodificação apresentados no Capítulo 3.

Nas Figuras 5.4, 5.5, 5.6 e 5.7 plotamos o tempo do algoritmo de decifração, medido em *cpb* para valores de $m = 11, 12, 13$ e 14, respectivamente, *versus* o *binary work factor* do algoritmo de Canteaut e Chabaud (1998). Nas Figuras 5.8, 5.9, 5.10 e 5.11 plotamos o tempo do algoritmo de decifração, medido em segundos, para os mesmos valores de m acima *versus* o grau do polinômio de Goppa.

Muitos valores de δ foram provados para um corpo de extensão $11 \leq m \leq 14$. Como pode-se observar nas Figuras 5.4 - 5.7, para valores fixos de $m = 11, 12, 13$, o algoritmo alternante proposto em (MacWilliams e Sloane, 1997, cap. 12.9) tem o melhor desempenho, seguido de Patterson (1974), Berlekamp-Massey (1969) e Barreto et al. (2011), nesta ordem. De igual forma acontece quando é fixado um nível de segurança aceitável [2] . Já no caso de $m = 14$ a ordem mencionada anteriormente só acontece para valores de segurança acima de 160. Sendo o algoritmo de melhor desempenho, para valores de segurança entre 40 e 80, o proposto por Berlekamp-Massey (1969). Especificamente nas Tabelas 5.4 e 5.5 podemos encontrar, o quanto mais rápido é um algoritmo respeito a outro. Por exemplo na Tabela 5.4 fixando o *bwf* em 59, o algoritmo alternante é 4.93 vezes mais rápido que o algoritmo de Barreto.

Na Figura 5.12, quando é fixado um nível de segurança, pode-se observar que o melhor desempenho não é obtido para valores de m pequenos. E sim, o sistema sistema trabalha melhor para valores de m grandes.

[2] Isto é para valores de segurança maiores que os valores decorrentes dos parâmetros sugeridos na proposta original do sistema McEliece.

	(m, δ)	[PAT]	[ALT]	[BMA]	[BAR]	*bwf*	Tamanho
1	(5, 3)	-6.369	-5.971	-5.748	-4.764	8.75	-
2	(6, 5)	-5.721	-5.729	-5.019	-4.265	11.84	-
3	(7, 8)	-4.911	-5.092	-4.158	-3.584	16.47	-
4	(8, 13)	-3.967	-4.44	-3.346	-2.731	22.34	-
5	(9, 30)	-2.467	-3.049	-1.59	-1.026	37.83	9.7kB
6	(10, 50)	-1.277	-2.055	-0.512	0.25	59.28	33.0kB
7	(11, 52)	-0.915	-1.716	-0.318	0.456	80.74	106kB
8	(12, 55)	-0.493	-1.232	-0.078	0.746	103.38	302.4kB
9	(13, 56)	-0.082	-0.758	0.023	0.971	126.28	716.6kB
10	(14, 57)	0.641	0.278	0.206	1.395	145.5	1638.4kB
11	(15, 58)	1.367	1.152	0.284	1.824	167.98	3686.4kB

Tabela 5.1: McEliece: *Benchmarks* para decifração. Logaritmo em base 2 do tempo medido em segundos.

	(m, δ)	[PAT]	[ALT]	[BMA]	[BAR]	*bwf*	Tamanho
1	(5, 3)	-9.682	-7.559	-6.922	-6.201	8.75	-
2	(6, 5)	-7.266	-7.804	-5.683	-4.858	11.84	-
3	(7, 8)	-6.041	-6.351	-4.873	-4.559	16.47	-
4	(8, 13)	-4.513	-6.311	-4.308	-2.964	22.34	-
5	(9, 22)	-2.877	-4.121	-2.084	-1.148	34.4	10.1kB
6	(10, 39)	-1.286	-2.325	-0.283	0.879	54.07	35.5kB
7	(11, 63)	3.153	2.907	1.634	3.907	87.44	119.2kB
8	(12, 70)	0.571	-0.79	1.648	3.033	120.05	364.7kB
9	(13, 79)	2.6	1.494	2.123	4.032	157.13	974.4kB
10	(14, 116)	2.393	0.509	3.249	4.703	244.30	3174.4kB
11	(15, 129)	3.17	1.493	3.808	5.387	306.63	-

Tabela 5.2: McEliece: *Benchmarks* para decifração, com comportamento anômalo. Logaritmo em base 2 do tempo medido em segundos.

[MCELIECE]	[RSA]
$O(n^2)$	$O(\hat{n}^3)$

Tabela 5.3: Complexidades assintóticas dos algoritmos de decifração para os sistemas: McEliece e RSA. Onde $\hat{n}$ é o numero de bits do texto plano.

bwf	59,24.8,33.5			80,70.5,106.3			103,181.1,302.4,		
	[ALT]	[PAT]	[BMA]	[ALT]	[PAT]	[BMA]	[ALT]	[PAT]	[BMA]
[PAT]	1.71	-	-	1.74	-	-	1.67	-	-
[BMA]	3.12	1.82	-	2.85	1.64	-	2.47	1.48	-
[BAR]	4.93	2.88	1.58	4.50	2.59	1.58	3.94	2.36	1.6

Tabela 5.4: Comparação da velocidade dos algoritmos apresentados no Capítulo 3, para valores de $bwf = 59, 80, 103$.

bwf	126,375.8,716.6			145,895.6,1.6MB			167,1.8MB,3.6MB		
	[ALT]	[PAT]	[BMA]	[ALT]	[PAT]	[BMA]	[ALT]	[PAT]	[BMA]
[PAT]	1.6	-	-	1.28	-	-	1.16	-	-
[BMA]	2.1	1.26	-	1.28	1.00	-	0.95	0.82	-
[BAR]	3.31	2.07	1.65	2.17	1.69	1.69	1.59	1.37	1.68

Tabela 5.5: Comparação da velocidade dos algoritmos apresentados no Capítulo 3, para valores de $bwf = 126, 145, 167$.

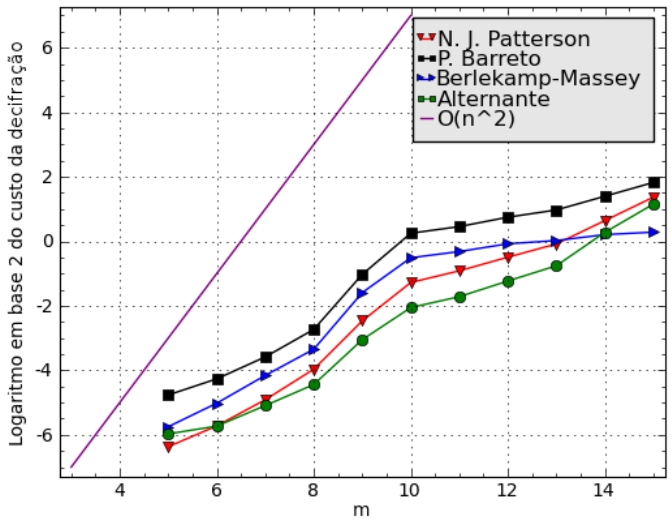

Figura 5.2: Complexidade assintótica dos algoritmos de decodificação apresentados no Capítulo 3. onde o parâmetro *bwf* encontra-se ordenado em forma crescente.

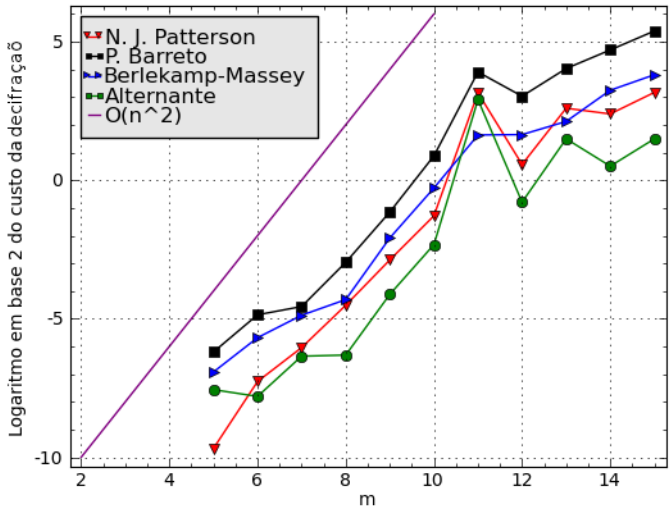

Figura 5.3: Complexidade assintótica dos algoritmos de decifração usando os algoritmos apresentados no Capítulo 3, onde o parâmetro *bwf* encontra-se não ordenado.

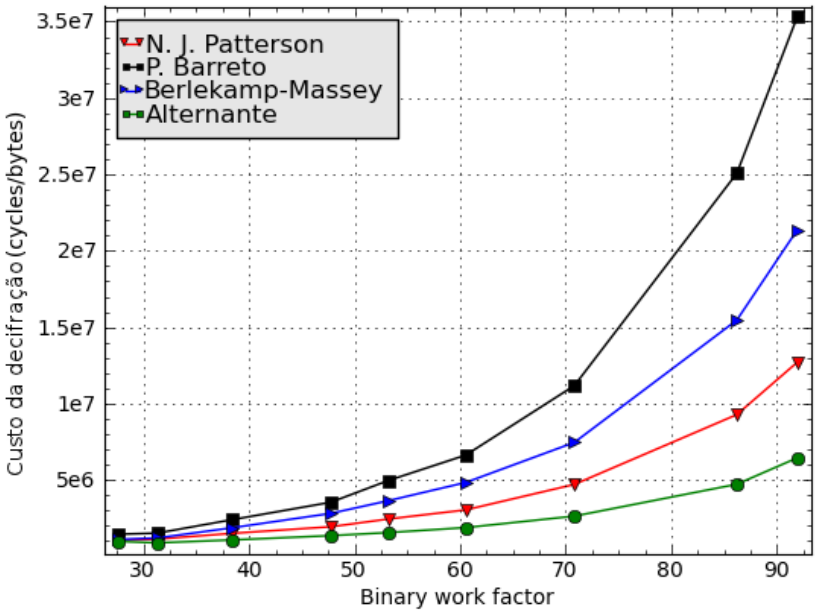

Figura 5.4: Custo de decifração (cpb) vs. *binary work factor* para extensão de grau $m = 11$.

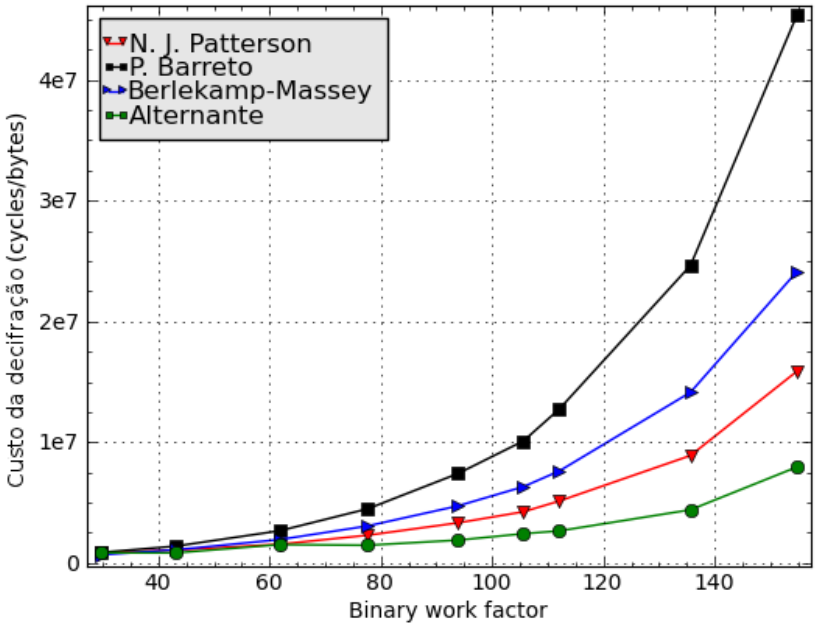

Figura 5.5: Custo de decifração (cpb) vs. *binary work factor* para extensão de grau $m = 12$.

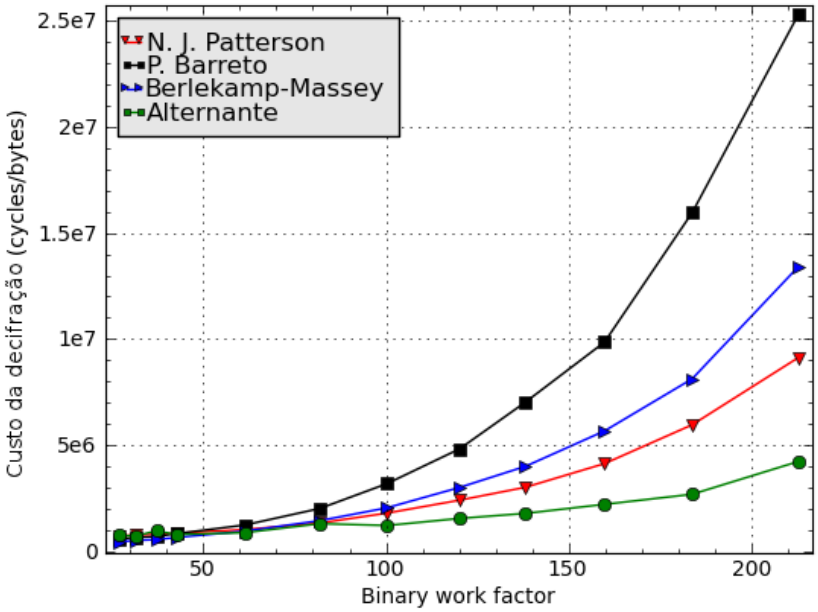

Figura 5.6: Custo de decifração (cpb) vs. *binary work factor* para extensão de grau $m = 13$.

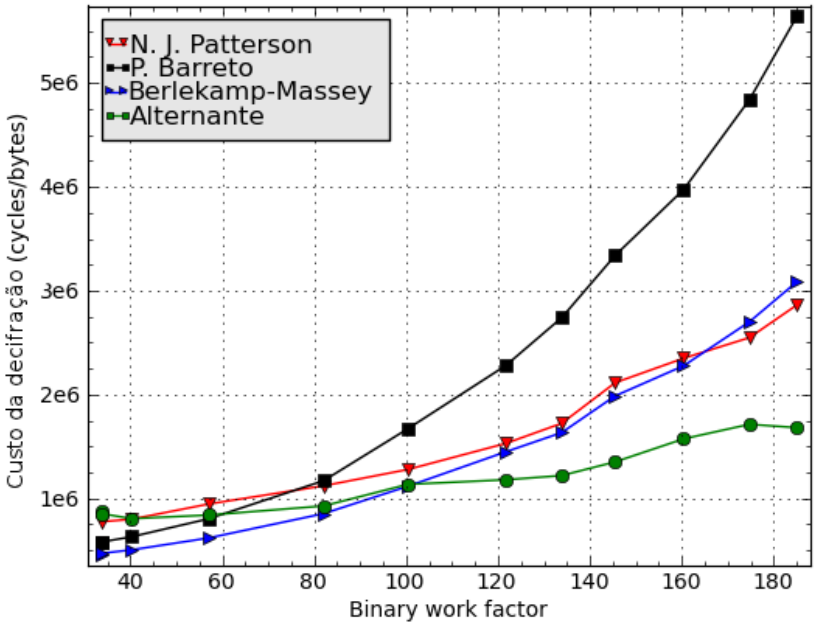

Figura 5.7: Custo de decifração (cpb) vs. *binary work factor* para extensão de grau $m = 14$.

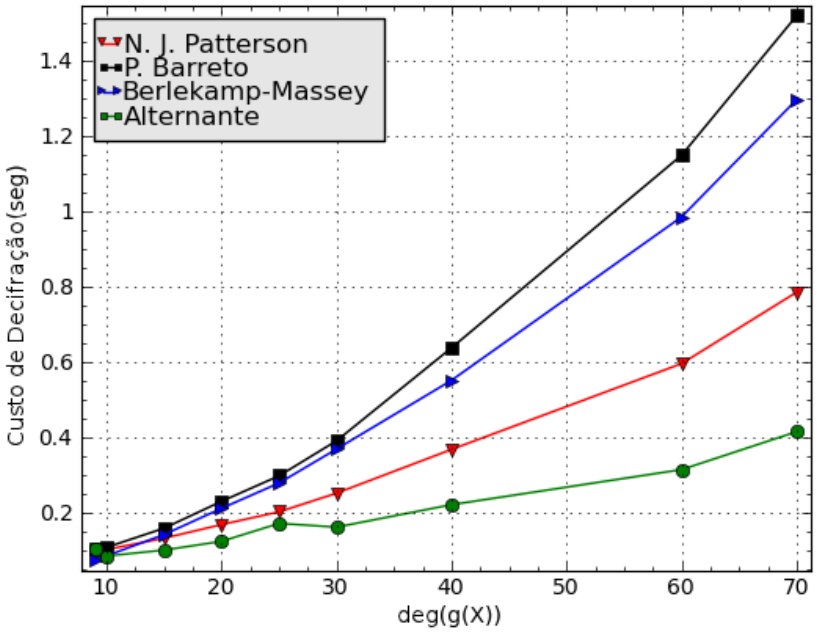

Figura 5.8: Custo de decifração (seg) vs. o grau do polinômio de Goppa para extensão de grau $m = 11$.

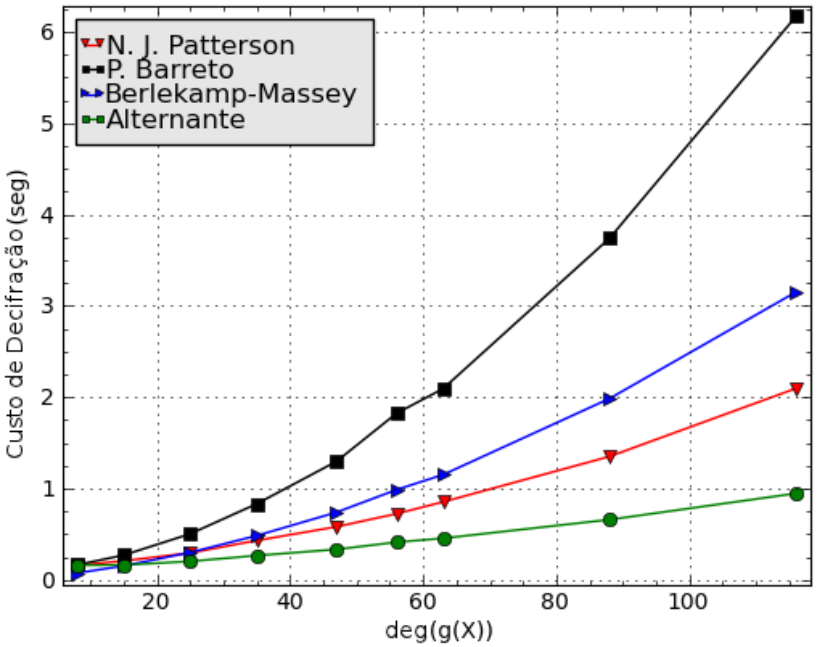

Figura 5.9: Custo de decifração (seg) vs. o grau do polinômio de Goppa para extensão de grau $m = 12$.

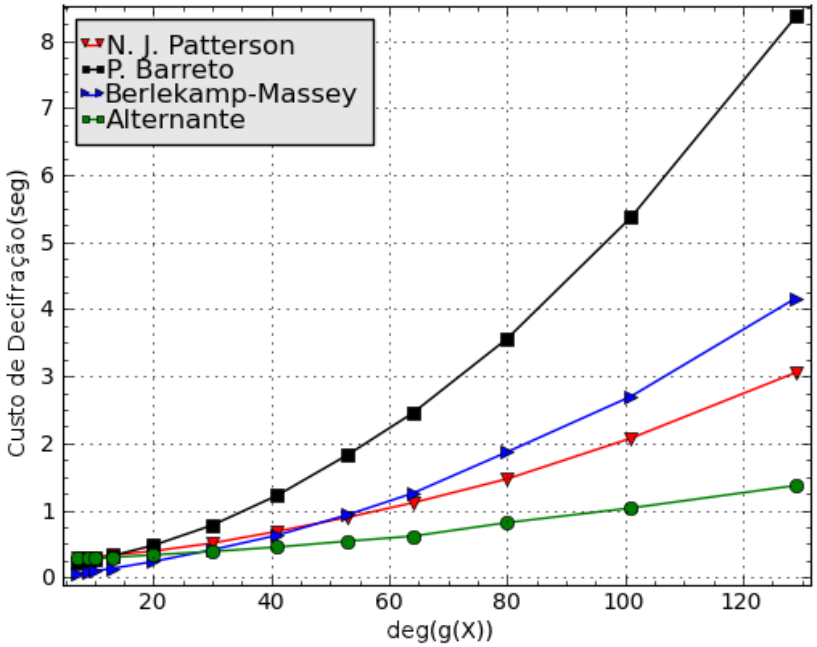

Figura 5.10: Custo de decifração (seg) vs. o grau do polinômio de Goppa para extensão de grau $m = 13$.

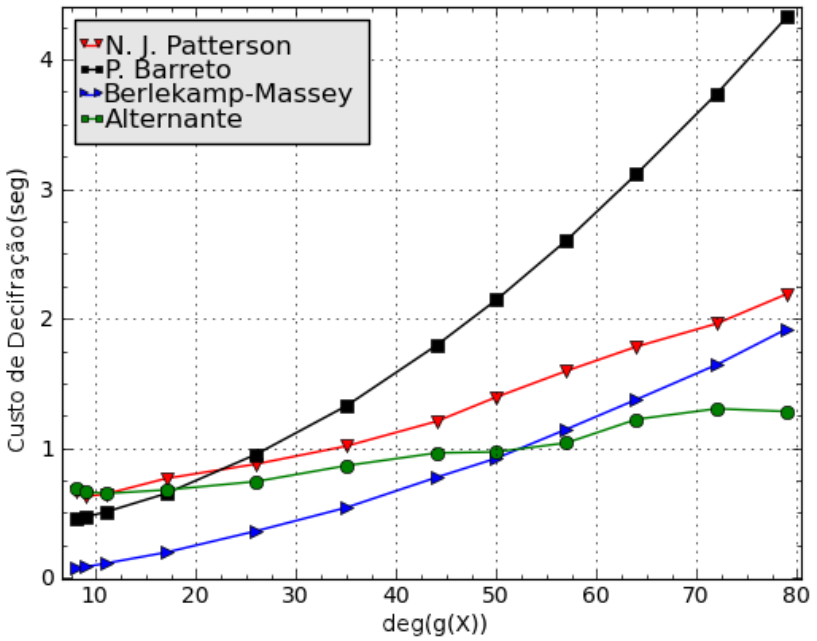

Figura 5.11: Custo de decifração (seg) vs. o grau do polinômio de Goppa para extensão de grau $m = 14$.

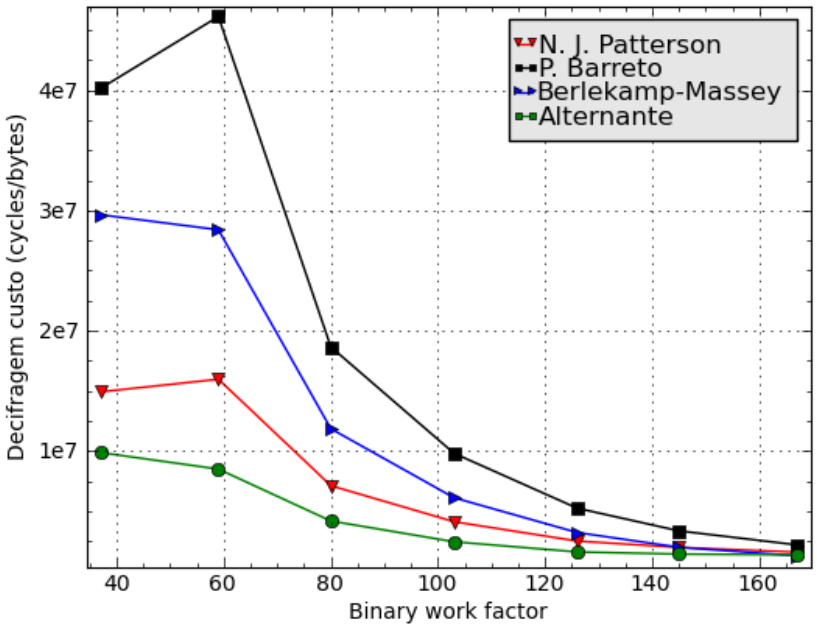

Figura 5.12: Custo de decifração (cpb) vs. *binary work factor*.

Capítulo 6

Conclusão

Neste trabalho, apresentamos uma revisão de como gerar códigos de Goppa e alternantes binários com foco nos algoritmos de decodificação. Apresentamos o algoritmo de Canteaut e Chabaud (1998), o qual é de muita importância em nossos experimentos, como já foi visto. Dentro dos algoritmos apresentados no Capítulo 3, os quais tem capacidade de correção $\lfloor \delta/2 \rfloor$, o algoritmo (MacWilliams e Sloane, 1997, cap. 12.9) tem um melhor desempenho seguido de Patterson (1974), Berlekamp-Massey (1969) e Barreto et al. (2011), nesta ordem. A fim de investigar, a complexiddade computacional do algoritmo de assinatura Courtois et al. (2001), $O(\delta!)$, nossos experimentos sugerem a implementação do algoritmo de decifração do sistema McEliece, usando o método alternante apresentado neste trabalho. Do experimento, respeito à complexidade assintótica do sistema McEliece, concluímos que esta última não só está afetada pelo comprimento das palavras-códigos, se não, também pela segurança do sistema McEliece.

Uma linha futura de nossa investigação é usar o algoritmo *Ball-collision* proposto por Bernstein, Lange, e Peters., substituindo o algoritmo de Canteaut e Chabaud (1998) como parâmetro para nossas comparações feitas. Além disso, nosso estudo pode ser enriquecido, adicionando mais algoritmos de decodificação com capacidade de correção δ.

Referências Bibliográficas

P. S. L. M. Barreto, R. Lindner, e R. Misoczki. Decoding square-free goppa codes over f_p. **Code-based Cryptography**, 2011.

E. R. Berlekamp. Goppa codes. **IEEE Transactions on Information Theory**, IT-19(5):590–592, 1973.

E. R. Berlekamp. **Algebraic Coding Theory**. No. M-6. Aegean Park Press, 1984.

D. J. Bernstein. List decoding for binary goppa codes. 2008.

D. J. Bernstein, E. Dahmen, e J. Buchmann. **Post-Quantum Cryptography**. Springer-Verlag, Berlin, 2009.

D. J. Bernstein, T. Lange, e C. Peters. Wild mceliece. **SAC 2010**, páginas 143–158, 2011a.

Daniel J. Bernstein, Tanja Lange, e Christiane Peters. Smaller decoding exponents: ball-collision decoding. **In CRYPTO 2011, Lecture Notes in Computer Science,**, 6841:743–760, 2011b.

B. Biswas. **Implementational aspects of code-based cryptography**. Tese de Doutorado, L'Ecole Polytechnique, 2010.

B. Biswas e V. Herbert. Efficient root finding of polynomials over fields of characteristic 2. **WEWoRK**, 2009.

A. Canteaut e F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to mcelieceś cryptosystem and to narrow-sense bch codes of length 511. **IEEE Transactions on Information Theory**, 44, 1998.

I. Castillo. **Geometría Algebraica, notas de aula, www.uv.es/ivorra/Libros/Geometria.pdf**. 2011.

N. Courtois, M. Finiasz, e N Sendrier. How to achieve a mceliece-based digital signature scheme. **Lecture Notes in Computer Science**, páginas 157–174, 2001.

S. C. Coutinho. **Números Inteiros e Criptografía RSA**. Instituto de Matemática Pura e Aplicada IMPA e Sociedade Brasileira de Computação SBM, Rio de Janeiro, 1997.

W. Diffie e M.E. Hellman. New directions in cryptography. **IEEE Transactions on Information Theory,**, IT-22(6):644–654, 1976.

J. Ding, C. Wolf, e B.Y. Yang. l-invertible cycles for multivariate quadratic public cryptography. **Lecture Notes in Computer Science**, 4450:266–281, 2007.

V. D. Goppa. A new class of linear error-correcting codes. **Problemy Peredachi Informatsii**, 6(3):24–30, 1970.

J. Grossman. Coding theory: Introduction to linear codes and applications. **River Academic Journal**, 4(2):–, 2008.

V. Guruswami e M. Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. **IEEE Transactions on Information Theory**, 45(6):1757–1767, 1999.

A. Hefez e M. L. T. Villela. **Códigos Corretores de Erros**. Instituto de Matemática Pura e Aplicada IMPA, Rio de Janeiro, 2008.

G. Hoffmann. Implementation of mceliece using quasi-dyadic goppa codes. Relatório Técnico 2, Technical University of Darmstadt, 2011.

K. Hubber. Note on decoding binary goppa codes. **Electronics Letters**, 32(2): 102–103, 1996.

R. E. Klima, N. Sigmon, e E. Stitzinger. **Applications of Abstract with MAPLE**. CRC Press LLC, U.S.A., 2000.

Kaoru Kurosawa, Toshiya Itoh, e Masashi Takeuchi. Public key cryptosystem using a reciprocal number with the same intractability as factoring a large number. **CRYPTOLOGIA**, 12:225–233, 1994.

P. Lee e E. Brickell. An observation on the security of mcelieceś public key cryptosystem. **Advances in Cryptology-EUROCRYPT'88, LNCS**, 330:275–280, 1989.

J. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. **IEEE Transactions on Information Theory**, 34(5):1354–1359, 1988.

F. J. MacWilliams e N. J. A. Sloane. **The Theory of Error Correcting Codes**, volume 16. North-Holland Publishing Company, U.S.A, 1997.

J. L. Massey. Shift- register synthesis and bch decoding. **IEEE Trans. Information Theory**, IT-15:122–127, 1969.

R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. **The Deep Space Network Progress Report**, páginas 114–116, 1978.

R. Misoczki e P. Barreto. Compact mceliece keys from goppa codes. **Lecture Notes in Computer Science**, 5867:376–392, 2009.

H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. **Problems of Control and Information Theory**, 15(2):159–166, 1986.

N. J. Patterson. The algebraic decoding of goppa codes. **IEEE Transactions on Information Theory**, IT-21(2):203–207, 1974.

R.L. Rivest, A. Shamir, e L. M. Adleman. Method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, 21(2):120–126, 1978.

N. Sendrier. Finding the permutation between equivalent codes: the support splitting algorithm. **IEEE Transactions on Information Theory**, 46(4):1193–1203, 2000.

V. M. Sidelnikov e S. O. Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. **Discrete Mathematics and Applications**, 2(4):439–444, 1992.

J. Stern. A method for finding codewords of small weight. **Coding Theory and Applications**, 388:106–133, 1989.

J. Stern. A new identification scheme based on syndrome decoding. **Lecture Notes in Computer Science**, (973):13–21, 1993.

J. Stern. Can one design a signature scheme based on error-correcting codes ? **Lecture Notes in Computer Science**, (917):424–426, 1994.

Van Tilburg. On the mceliece cryptosystem. **Lecture Notes in Computer Science**, (403):119–131, 1990.