Melhorando o ataque de reação contra o QC-MDPC McEliece

Thales Areco Bandiera Paiva

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Programa: Ciência da Computação
Orientador: Prof. Dr. Routo Terada

O autor recebeu auxílio financeiro da CAPES

São Paulo, novembro de 2017

Melhorando o ataque de reação contra o QC-MDPC McEliece

Esta é a versão original da dissertação elaborada pelo candidato Thales Areco Bandiera Paiva, tal como submetida à Comissão Julgadora.

Sumário

Capítulo 1

Introdução

O esquema de chave pública de McEliece [McE78] foi o primeiro baseado em códigos corretores de erros, e também o primeiro a usar um procedimento aleatório para a encriptação [MVOV96]. Tanto a encriptação quanto a decriptação são computacionalmente mais eficientes que as respectivas de outros esquemas baseados no problema de fatoração ou no problema do logaritmo discreto, considerando os correspondentes níveis de segurança oferecidos [OS09]. Além disso, o esquema é baseado no problema $\mathcal{NP}$-difícil da decodificação por síndrome [BMVT78], que é um problema conjecturado difícil para instâncias típicas [BBD09]. Isso sugere que o esquema de McEliece pode oferecer uma segurança maior do que os algoritmos baseados no problemas de fatoração de inteiros ou no cálculo do logaritmo discreto, que por pertencerem à interseção das classe $\mathcal{NP}$ e co$\mathcal{NP}$, conjectura-se não serem $\mathcal{NP}$-difíceis [AB09].

Esse esquema, como o RSA [RSA78], foi publicado na década de 70, mas não foi bem recebido por dois motivos. O primeiro é que as suas chaves públicas são muito grandes. Por exemplo, para níveis de segurança maiores do que 100 bits, as chaves têm tamanho maiores do que 100 quilobytes [BCS13, Tabela 1.1]. O segundo motivo é que acreditava-se ser impossível usar o problema da decodificação por síndrome para desenvolver esquemas de assinatura. Tal crença foi justificada por McEliece [McE78] pela dificuldade de se decodificar qualquer mensagem usando certos códigos corretores de erros, um passo importante para gerar uma assinatura. Apesar disso, em 2001, Courtois, Finiasz, e Sendrier [CFS01] mostraram como construir um tal esquema de assinatura. Este esquema de assinatura é significativamente mais lento do que os esquemas baseados na fatoração e no logaritmo discreto, mas tem uma boa redução de segurança, e um algoritmo rápido para a verificação da assinatura.

Depois de anos sem atenção, o esquema de McEliece passou a ser estudado pois, ao contrário do RSA [RSA78] e das Curvas Elípticas [Mil86], esse esquema resiste a ataques quânticos baseados no algoritmo de Shor [Sho97]. Em particular, acredita-se que computadores quânticos não são capazes de resolver problemas $\mathcal{NP}$-difíceis eficientemente [AB09], oferecendo apenas uma melhora quadrática no tempo de busca por uma solução, por exemplo, com o algoritmo de Grover [Gro96]. Dessa forma, problemas $\mathcal{NP}$-difíceis podem ser bons candidatos a suportar algoritmos criptográficos pós-quânticos. O uso desses problemas deve ser feito com com certo cuidado, pois as classes de complexidade dizem respeito à dificuldade dos problemas considerando as suas instâncias mais difíceis, podendo ser fácil resolver suas instâncias típicas [BT+06, Imp95].

Com os avanços da computação quântica, os principais algoritmos de chave pública em uso ficam cada vez mais vulneráveis, o que justifica a busca por algoritmos criptográficos resistentes a ataques quânticos [CCJ+16]. Essa busca dá origem à área chamada Criptografia Pós-Quântica, que, além de esquemas baseados em códigos corretores de erros, estuda também esquemas baseados em *hashes*, reticulados, equações quadráticas multivariadas, e isogenias de curvas elípticas [BBD09].

Os principais esquemas de chave pública baseados em códigos corretores de erros são o de McEliece [McE78] e o de Niederreiter [Nie86]. Os esquemas têm segurança equivalente quando o adversário não tem nenhuma informação sobre a mensagem transmitida, ou quando são usa-

das conversões de segurança como as de Kobara e Imai [KI01] para deixar os textos encriptados indistinguíveis de textos aleatórios mesmo contra ataques adaptativos de texto encriptado escolhido (IND-CCA2) [BDPR98]. Tanto por ser mais antigo quanto por ser mais facilmente descrito, os resultados apresentados nas próximas seções são baseados no esquema de McEliece.

No esquema de McEliece, a chave secreta de Alice é um código linear $\mathcal{C}$ com capacidade de correção de t erros para o qual Alice conhece um decodificador eficiente. A chave pública de Alice, é uma matriz geradora de $\mathcal{C}$, possivelmente embaralhada de forma que seja difícil usá-la para recuperar um decodificador para $\mathcal{C}$. Para enviar uma mensagem $\mathbf{m}$ a Alice, codifique $\mathbf{m}$ usando a chave pública de Alice e adicione t erros à codificação, obtendo $\mathbf{c}$. Então Alice, que é a única portadora de um decodificador eficiente para $\mathcal{C}$, decodifica $\mathbf{c}$ e obtém $\mathbf{m}$.

Uma das principais linhas de pesquisa em criptografia com códigos corretores de erros é diminuir o tamanho das chaves do esquema de McEliece através da escolha adequada da família de códigos associada. Originalmente, McEliece sugeriu o uso de códigos de Goppa [Gop70, Ber73] binários e irredutíveis, que, apesar de não sofrerem ataques críticos, resultam em grandes chaves. O motivo principal por que códigos de Goppa, de taxa[1] não muito alta [FGUO+13], ainda resistem aos ataques é que não se conhece algoritmo eficiente para distinguir matrizes geradoras de códigos de Goppa de matrizes geradoras de códigos aleatórios. Algumas outras famílias de códigos foram propostas, como as dos códigos BCH quase cíclicos [Gab05], alternantes quase cíclicos [BCGO09], LDPC [SMR00], e de Goppa quase diádicos [MB09]. Apesar de obterem uma boa redução do comprimento da chave, todas foram mostradas inseguras depois de alguns anos [OTD10, FOPT10, FOP+16].

Em 2013, foi proposta por Misoczki et al. [MTSB13] uma nova variante do Esquema de McEliece, chamada QC-MDPC McEliece que usa códigos que possuem uma matriz de paridade quase cíclica e de densidade moderada, do inglês *quasi-cyclic moderate-density parity-check codes* (QC-MDPC). Esta variante promete chaves públicas de 4801 bits para um nível de segurança de 80 bits, e tem uma boa redução de segurança. A iniciativa europeia PQCRYPTO, para o desenvolvimento de criptografia pós-quântica, na revisão de 2015 de seu documento com recomendações de esquemas pós-quânticos [ABB+15], considerou esse esquema como um forte candidato, junto com a variante Stehlé-Steinfeld [SS11] do esquema NTRU [HPS98], a ser um padrão adotado por aplicações com alto nível de segurança.

Até meados de 2016, o QC-MDPC McEliece não sofrera ataques críticos. Porém, na Asiacrypt de 2016, Guo, Johansson, e Stankovski [GJS16] mostraram um ataque eficiente para a recuperação de chave contra esse esquema. O ataque se baseia no fato de que os decodificadores de códigos QC-MDPC são iterativos e podem falhar, e, quando isso ocorre, o destinatário pede para o remetente reenviar a mensagem com outro erro inserido. O ataque é classificado como um ataque de reação [HGS99] pois o atacante precisa saber como Alice, a portadora da chave secreta, reage para cada texto encriptado enviado. A observação dos autores é que a probabilidade de o decodificador falhar para um erro $\mathbf{e}$ é significativamente menor quando $\mathbf{e}$ e a chave secreta compartilham certas propriedades, chamadas de espectros.

Suponha que uma atacante Eva queira atacar Alice. O ataque tem duas fases. Na primeira, Eva gera uma grande quantidade de textos encriptados e os envia para Alice, gravando quais encriptados Alice conseguiu e quais não conseguiu decodificar. Usando estas informações, Eva tenta então estimar o espectro da chave secreta de Alice. Na segunda fase, sem mais se comunicar com Alice, Eva usa as informações coletadas sobre a chave secreta para reconstruí-la.

1.1 Motivação

Foram identificados três problemas do algoritmo de reconstrução que Guo et al. sugerem para a segunda fase de seu ataque, destacados a seguir.

1. Para que o algoritmo funcione eficientemente, o espectro estimado pela primeira fase tem

[1] A taxa de um código linear de dimensão k e comprimento n é igual a k/n.

que estar quase totalmente correto. Segundo os autores, para o nível de segurança de 80 bits, isso em geral é possível fazendo-se em torno de 200 milhões de testes de decodificação na primeira fase.

2. O algoritmo não escala bem para níveis de segurança mais altos. Pela sua análise assintótica, e pelos nossos experimentos, parece impraticável usá-lo para o nível de segurança de 256 bits. A saber, sua complexidade é $\mathcal{O}\left(r^{\phi(\lambda)}\right)$, onde r é a codimensão[2] do código QC-MDPC, e $\phi(\lambda)$ é uma função que cresce, embora não muito rapidamente, de acordo com o nível de segurança λ.

3. O algoritmo tem natureza recursiva, e não é fácil escolher uma boa estratégia de paralelização. Em outras palavras, certas estratégias de paralelização são boas para certas chaves, mas não há como saber a priori como quais estratégias obtêm melhores *speedups* ao permitir o uso de mais processadores.

Encontrar algoritmos mais eficientes para a reconstrução de chave a partir do espectro é importante para que possamos estimar qual a vida útil de uma chave do esquema QC-MDPC McEliece, ou mesmo quais parâmetros de segurança desse esquema são seguros contra esse ataque de reação.

1.2 Objetivo

O objetivo deste trabalho é melhorar o ataque de reação de Guo et al. [GJS16] contra o QC-MDPC McEliece. Estamos interessados em algoritmos que possibilitem reconstruir a chave com um menor número de testes de decodificação, isto é, observando um menor número de reações, e também em algoritmos que reconstruam a chave em um menor número de passos do que o algoritmo propostos por Guo et al. Para avaliar os algoritmos propostos, consideramos tanto a análise probabilística assintótica dos algoritmos como os seus desempenho na prática, usando implementações em linguagem C.

1.3 Trabalhos relacionados

Destacam-se dois trabalhos relacionados, ambos apresentados em conferências de 2017, que também se basearam no ataque de reação de Guo et al. [GJS16].

O primeiro é um trabalho apresentado na conferência PQCRYPTO por Fabšič et al.[FHS+17] que estende o ataque de reação para o QC-LDPC McEliece. Este ataque usa um algoritmo de reconstrução de chave diferente do de Guo et al. modelando o espectro como um grafo esparso e buscando cliques de certo tamanho neste grafo. Apesar de ser uma nova interpretação, a complexidade do algoritmo não parece ser significativamente diferente da do algoritmo original de Guo et al..

O segundo é um trabalho apresentado na conferência CHES por Rossi et al. [RHHM17], que mostra um ataque por canal lateral a uma implementação do QC-MDPC McEliece, chamada QcBits [Cho16]. Depois de recuperar a informação sobre a chave usando um canal lateral, os autores mostraram como recuperar a chave usando uma relação linear entre os blocos da chave privada. Este trabalho tem certas similaridades com o nosso, em particular com um dos nossos algoritmos propostos. Porém, nosso algoritmo é significativamente diferente, sendo eficiente mesmo com menos informação sobre o espectro. Além disso, nossas análises teórica e prática do desempenho do algoritmo de reconstrução são mais detalhadas. Note que o foco do trabalho de Rossi et al. era mostrar um ataque ao QcBits por canal lateral, então é natural que a ênfase não tenha sido dada ao algoritmo de reconstrução.

[2] A codimensão de um código é a dimensão de sua matriz de paridade.

1.4 Contribuições originais

Este trabalho apresenta duas contribuições originais. Ambas são algoritmos para a reconstrução da chave secreta a partir do espectro, que são mais eficientes, tanto assintoticamente quanto na prática, do que as apresentadas por Guo et al. Os dois algoritmos apresentados são probabilísticos, e ambos podem ser paralelizáveis trivialmente, resultando em bons *speedups*.

O primeiro algoritmo apresentado é uma simples extensão probabilística do algoritmo de Guo et al.. Apesar de simples, ele é bem mais eficiente do que o algoritmo original. Um fato observado ilustrativo disso é que, quando há informação completa sobre o espectro, o algoritmo proposto faz menos operações para o nível de segurança de 256 bits do que o original faz para o nível de 80 bits. Mesmo assim, o algoritmo não é capaz de trabalhar eficientemente com baixa quantidade de informação sobre o espectro. A sua complexidade é melhor do que a do algoritmo original, embora similar à complexidade deste, sendo $\mathcal{O}\left(r^{\tau(\lambda)}\right)$, onde τ é uma função do nível de segurança λ que parece crescer mais lentamente do que o expoente $\phi(\lambda)$ do algoritmo original, como apresentado no item 2 da Seção 1.1.

O segundo algoritmo, que consideramos ser a nossa principal contribuição, é significativamente diferente do algoritmo de Guo et al., usando uma relação linear entre os componentes da chave secreta. Mesmo com uma quantidade de informação entre 50 e 55% menor do que o necessário para o funcionamento do algoritmo de Guo et al, o algoritmo tem complexidade $\mathcal{O}\left(r^4\right)$. Esta complexidade é uma melhora significativa em relação ao expoente crescente em função do nível de segurança do algoritmo original de Guo et al. e também do algoritmo probabilístico de busca. A proposta desse algoritmo foi submetido à revista *Transactions on Fundamentals of Electronics, Communications and Computer Sciences* do IEICE [PT17].

Capítulo 2

Fundamentos

2.1 Notações e definições básicas

Conjuntos

Denotamos por $[r]$ o conjunto $\{1, 2, \ldots, r\}$. Se A é um conjunto, denotamos o número de elementos de A por $|A|$ ou por $\#A$. O conjunto dos números naturais é denotado por $\mathbb{N}$. O símbolo $\mathbb{F}_2$ denota o corpo finito com dois elementos. Se b é um elemento de $\mathbb{F}_2$, $\overline{b}$ denota o seu complemento. Então $\overline{1} = 0$ e $\overline{0} = 1$.

Vetores e Álgebra Linear

Letras em negrito minúsculas como $\mathbf{u}$ representam vetores, enquanto maiúsculas como $\mathbf{H}$ representam matrizes. Vetores são considerados como linhas. O T sobrescrito denota a transposição de matrizes ou vetores, então o vetor $\mathbf{u}^T$ é um vetor coluna. Índices referentes a posições de vetores começam em 1, de forma que se $\mathbf{v} = [0\,0\,1\,0\,0\,0]$, então a posição de índice 3 de $\mathbf{v}$ tem o elemento 10. Apesar disso, em geral começamos a numerar blocos de matrizes ou vetores por 0, para ser condizente com a notação estabelecida por Misoczki et al. [MTSB13]. O suporte de um vetor $\mathbf{v}$, em geral binário, denotado por $\sup(\mathbf{v})$ é o conjunto das posições de suas entradas não nulas. Por exemplo, se $\mathbf{v} = [0\,0\,1\,1\,0\,1\,0]$, então $\sup(\mathbf{v}) = \{3, 4, 6\}$. Se $\mathbf{v}$ é um vetor, então $\mathbf{v}[i]$ denota a sua i-ésima entrada. Usamos $|$ para separar blocos de uma matriz como $\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1]$, ou de vetores como $\mathbf{e} = [\mathbf{e}_0|\mathbf{e}_1]$. O símbolo $\mathbb{F}_2^n$ denota espaço vetorial binário de comprimento n. Isto é, $\mathbb{F}_2^n$ é o conjunto de cadeias de n bits, onde a soma entre duas cadeias é feita bit a bit. Denotamos por $\dim \mathbf{M}$ a dimensão do espaço gerado por uma matriz $\mathbf{M}$.

Algoritmos

Em geral, algoritmos são escritos em *CamelCase* e versalete como em QUICKSORT. Classifica-se como algoritmo probabilístico um algoritmo que usa um gerador de números aleatórios para tomar decisões. Se f e g são duas funções de $\mathbb{N}$ em $\mathbb{N}$, dizemos que $f = \mathcal{O}(g)$ se existe uma constante c e um inteiro n_0 tal que $f(n) \leq cg(n)$ para todo $n > n_0$. Note que o sinal de igualdade em $f = \mathcal{O}(g)$ não implica que $g = \mathcal{O}(f)$. Um algoritmo de tempo polinomial é aquele que executa $\mathcal{O}(n^c)$ passos até terminar, para alguma constante c, onde n é o número de bits usados para se armazenar entrada. Similarmente, um algoritmo de tempo exponencial é aquele cujo o número de passos até o fim da execução é $\mathcal{O}(k^n)$, para alguma constante $k > 1$. Dizemos que um algoritmo tem tempo médio polinomial se o tempo esperado de execução do algoritmo é $\mathcal{O}(n^c)$, para certa distribuição sobre as entradas. Tipicamente consideramos a distribuição uniforme sobre as entradas. É comum dizer simplesmente algoritmo polinomial no lugar de algoritmo de tempo polinomial, ou similarmente para tempo exponencial. Em complexidade computacional, classificam-se como problemas fáceis aqueles que podem ser resolvidos por algum algoritmo polinomial. Chamam-se difíceis aqueles para que não se conhece algoritmo polinomial de resolução. É importante notar que estas definições de fácil e difícil são assintóticas e de pior caso, então de-

vem ser um pouco refinadas para usá-las em criptografia. Um oráculo é um algoritmo que resolve um problema em um passo computacional.

2.2 Teoria de Códigos

O esquema de McEliece é baseado em códigos corretores de erros, que são meios eficientes de transmitir mensagens, que são conjuntos ordenados de símbolos, de uma fonte a um receptor, através de um canal possivelmente ruidoso [Sha48]. Quando um canal de transmissão não é ruidoso, a meta é codificar uma mensagem de modo que a mensagem transmitida tenha o mínimo possível de símbolos, e a mensagem original possa ser recuperada a partir da transmitida. Por sua vez, numa transmissão feita por um canal ruidoso, a meta é codificar a mensagem de modo que esta possa ser recuperada, mesmo que alguns de seus símbolos sejam alterados por conta de ruídos causados pela imperfeição do canal.

A Figura 2.2.1 exemplifica a transmissão de uma mensagem por um canal ruidoso. Vamos seguir o fluxo definido pelos arcos. Uma fonte escolhe uma mensagem $\mathbf{m}$ do conjunto $\mathcal{M}$ de mensagens possíveis. O algoritmo codificador E codifica a mensagem $\mathbf{m}$ gerando a palavra $\mathbf{c}$. Ao passar pelo canal ruidoso, a palavra $\mathbf{c}$ sofre um erro $\mathbf{e}$ bit por bit se tornando a palavra $\mathbf{c}'$. O algoritmo decodificador D é aplicado sobre $\mathbf{c}'$ resultando na palavra $\hat{\mathbf{m}}$. Idealmente, queremos que a mensagem recuperada $\hat{\mathbf{m}}$ seja igual à mensagem original $\mathbf{m}$ com alta probabilidade. Porém, se o erro $\mathbf{e}$ fizer $\mathbf{c}'$ ser muito diferente de $\mathbf{c}$, pode ser que o algoritmo D não seja capaz de recuperar a mensagem $\mathbf{m}$ corretamente. Naturalmente, espera-se que E e D rodem em tempo polinomial. E é tipicamente um algoritmo determinístico, enquanto D pode ser probabilístico.

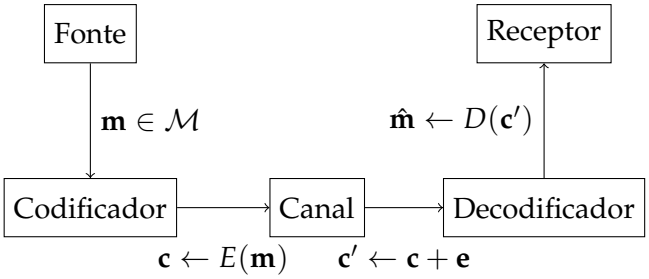

Figura 2.2.1: *Transmissão de uma palavra* $\mathbf{m}$ *através de um canal ruidoso.*

É comum em teoria de códigos chamar de decodificação de uma mensagem transmitida tanto a correção de seus erros, quanto a recuperação da mensagem original, embora estas ações sejam fundamentalmente diferentes. Em geral, para códigos usados na prática, em sua grande maioria códigos lineares, uma vez que os erros da mensagem foram corrigidos, recuperar a mensagem original é simplesmente resolver um sistema linear. Neste texto, algoritmos de decodificação são sempre algoritmos corretores de erros, omitindo o procedimento de recuperação da mensagem original a partir de sua codificação.

Os códigos mais estudados e usados em sistemas reais são os códigos lineares. Um código linear é um subespaço vetorial, o que lhes garante certa estrutura.

Definição 2.2.1 (Código binário linear)**.** Um código binário $[n, k]$-linear é um subespaço vetorial de dimensão k do espaço $\mathbb{F}_2^n$.

Por serem subespaços vetoriais, códigos lineares têm representações compactas através da imagem de matrizes, ou equivalentemente, através do núcleo de matrizes. Destas representações, seguem naturalmente as seguintes definições.

Definição 2.2.2 (Matrizes geradora e de paridade)**.** Seja $\mathcal{C}$ um código binário $[n, k]$-linear. Se $\mathcal{C}$ for igual ao espaço gerado por combinações lineares das linhas de uma matriz $\mathbf{G}$ de $\mathbb{F}_2^{k\times n}$, dizemos que $\mathbf{G}$ é uma matriz geradora de $\mathcal{C}$.

Analogamente, se $\mathcal{C}$ é o núcleo de uma matriz $\mathbf{H}$ de $\mathbb{F}_2^{n-k\times n}$, dizemos que $\mathbf{H}$ é uma matriz de paridade de $\mathcal{C}$. Neste trabalho, estamos interessados em códigos que são definidos por sua matriz de paridade. Então, para a notação ficar menos carregada, é útil definir a codimensão de um código como $r = n - k$, e então $\mathbf{H}$ pertence a $\mathbb{F}_2^{r\times n}$.

Assim, se $\mathbf{G}$ e $\mathbf{H}$ são matrizes geradoras quaisquer de um código $[n,k]$-linear $\mathcal{C}$, então

$$\mathcal{C} = \{\mathbf{mG} : \mathbf{m} \in \mathbb{F}_2^k\} = \{\mathbf{c} \in \mathbb{F}_2^n : \mathbf{cH}^T = 0\}.$$

Uma matriz de paridade $\mathbf{H} = (h)_{ij}$ tem esse nome pois, para um vetor $\mathbf{c} = [c_1 \dots c_n]$ pertencer a um código que admite matriz de paridade $\mathbf{H}$, as seguintes equações devem ser satisfeitas:

$$\begin{array}{ccccccccc} c_1h_{11} & + & c_2h_{12} & + & \dots & c_nh_{1n} & + & = & 0 \\ c_1h_{21} & + & c_2h_{22} & + & \dots & c_nh_{2n} & + & = & 0 \\ \vdots & & \vdots & & \dots & & + & = & \vdots \\ c_1h_{r1} & + & c_2h_{r2} & + & \dots & c_nh_{rn} & + & = & 0 \end{array}$$

que por serem equações sobre $\mathbb{F}_2$, são ditas equações de paridade.

Para verificar se um dado vetor $\mathbf{v}$ pertence a um código, multiplique $\mathbf{v}$ à esquerda por $\mathbf{H}^T$ e verifique se foi obtido o vetor $\mathbf{0}$. Se sim, todas as equações de paridade foram satisfeitas, e $\mathbf{v}$ pertence ao código. Para a decodificação, muitas vezes é útil saber quais as equações de paridade que não são satisfeitas por um dado vetor. Chama-se síndrome de $\mathbf{v}$ o vetor binário $\mathbf{s}$ tal que, $\mathbf{s}[i] = 1$ se e somente se a i-ésima equação de paridade não foi satisfeita para $\mathbf{v}$. Formalmente, a síndrome de um vetor é definida a seguir.

Definição 2.2.3 (Síndrome)**.** A síndrome de um vetor $\mathbf{v}$, denotada por $s(\mathbf{v})$, em relação a um código que admite matriz de paridade $\mathbf{H}$, é

$$s(\mathbf{v}) = \mathbf{vH}^T.$$

Então a síndrome de um vetor é $\mathbf{0}$ se e somente se este vetor pertence ao código.

Dois conceitos importantes para o estudo de códigos são o de peso e distância, que fazem de $\mathbb{F}_2^n$ um espaço métrico.

Definição 2.2.4 (Peso)**.** O peso de Hamming, ou simplesmente peso, de um vetor $\mathbf{v}$, denotado por $\mathrm{w}(\mathbf{v})$, é o número total de suas coordenadas não nulas.

Definição 2.2.5 (Distância)**.** A distância de Hamming, ou simplesmente distância, entre dois vetores $\mathbf{u}$ e $\mathbf{v}$, denotada por $\mathrm{dist}(\mathbf{u},\mathbf{v})$, é o número total de coordenadas em que $\mathbf{u}$ e $\mathbf{v}$ diferem. Para códigos binários, vale que $\mathrm{dist}(\mathbf{u},\mathbf{v}) = \mathrm{w}(\mathbf{u}+\mathbf{v})$.

Com isso pode ser definido um problema fundamental em teoria de códigos, que é o problema $\mathcal{NP}$-difícil[BMVT78] em que é baseado o esquema de McEliece.

Definição 2.2.6 (Problema da decodificação)**.** O problema da decodificação por distância mínima é: dado um código $[n,k]$-linear $\mathcal{C}$, e um vetor $\mathbf{c}'$ de $\mathbb{F}_2^n$, encontre o vetor $\mathbf{c}$ de $\mathcal{C}$ tal que $\mathrm{dist}(\mathbf{c},\mathbf{c}')$ é mínima.

Códigos de Goppa

O esquema de McEliece [McE78] original usa códigos de Goppa [Gop70, Ber73]. Estes são códigos algébricos que são bons candidatos ao uso criptográfico, mas não são muito interessantes para chaves compactas. Como não são fundamentais para entender o QC-MDPC McEliece, optamos por não descrevê-los detalhadamente.

Algumas de suas propriedades importantes para o esquema original de McEliece são listadas a seguir. Cada código é gerado por um polinômio irredutível $g(x)$ de $\mathbb{F}_{2^m}[x]$. A capacidade de

correção é igual ou superior ao grau t de $g(x)$. É fácil construir decodificadores de t erros eficientes conhecendo $g(x)$ [Pat75]. Não se conhece algoritmo eficiente para construir decodificadores sem conhecer $g(x)$. Parece ser difícil em geral distinguir matrizes geradoras de códigos de Goppa e matrizes geradoras de códigos aleatórios [BCS13, BBD09, BLP08], embora seja possível quando k é próximo de n [FGUO+13].

Códigos LDPC

Os códigos LDPC foram introduzidos por Gallager [Gal62] na década de 60. Sua característica determinante é que possuem uma matriz de paridade de baixa densidade. Intuitivamente, isso significa que o número de entradas não nulas cresce linearmente no número de colunas da matriz. Códigos LDPC não são necessariamente binários [RL09], porém apenas os binários interessam para este trabalho.

Definição 2.2.7 (LDPC)**.** Um código binário e linear com matriz de paridade $\mathbf{H}$ pertencente a $\mathbb{F}_2^{r\times n}$ é dito um código $[n,r]$-LDPC [Gal62] se o número de elementos não nulos de $\mathbf{H}$ é $\mathcal{O}(n)$.

Para entender o funcionamento do algoritmo de decodificação, é útil considerar uma interpretação gráfica para códigos LDPC.

Definição 2.2.8 (Grafo de Tanner)**.** Seja $\mathbf{H} = (h)_{ij}$ uma matriz de um código $[n,r]$-LDPC. Considere os conjuntos de nós $V = \{v_1, v_2, \ldots, v_n\}$, chamado de nós de variáveis, e $C = \{c_1, c_2, \ldots, c_r\}$, chamado de nós de verificação. Considere também o conjunto E de arestas entre V e C, dado por

$$E = \{(i,j) : h_{ij} = 1 \text{ para algum par de vértices } (c_i, v_j)\}.$$

Construa o seguinte grafo $G = (V \cup C, E)$, em $V \cup C$ é o conjunto de nós, e E é o conjunto de arestas. Chamamos G de grafo de Tanner da matriz $\mathbf{H}$.

Em geral consideram-se uma matriz binária indissociável de seu grafo de Tanner. De forma que, uma vez que a matriz foi definida, pode-se referenciar o seu grafo de Tanner sem explicitamente construí-lo.

Exemplo 2.2.9. Considere $\mathbf{H}$ a matriz de paridade de um código LDPC definida como

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

Esta matriz tem como grafo de Tanner o grafo mostrado na Figura 2.2.2.

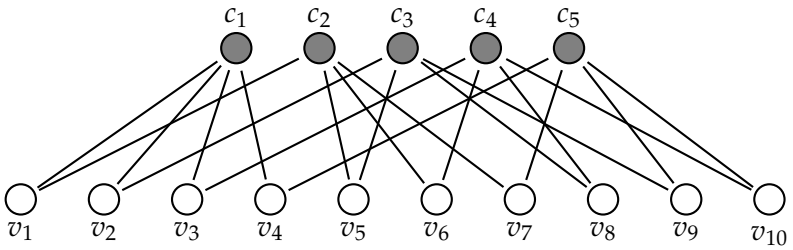

Figura 2.2.2: *Grafo de Tanner da matriz* $\mathbf{H}$.

Definição 2.2.10 (Nó de verificação insatisfeito)**.** Seja $\mathbf{H}$ uma matriz de um código $[n,r]$-LDPC, e seja $\mathbf{c}$ um vetor qualquer em $\mathbb{F}_2^n$. Chame de $\mathbf{s} = [s_1, s_2, \ldots, s_r]$ a síndrome de $\mathbf{c}$, ou seja $\mathbf{s} = \mathbf{c}\mathbf{H}^T$. Se $s_i = 0$, dizemos que o nó de verificação c_i está satisfeito. Caso contrário, $s_i = 1$ e dizemos que c_i está insatisfeito.

Para corrigir erros em códigos LDPC, usam-se algoritmos iterativos de otimização local para a decodificação que localizam as possíveis posições de erros usando o fato de a matriz de paridade ser esparsa. Estes algoritmos são eficientes, e conseguem boas capacidades de correção de erros [RL09] porém têm dois problemas.

O primeiro é que todos os algoritmos conhecidos de decodificação para códigos LDPC têm uma probabilidade, potencialmente pequena mas não negligenciável, de não conseguir decodificar algumas mensagens. Este problema não é muito sério quando o objetivo é integridade na comunicação, pois basta que o destinatário, ao não conseguir decodificar a mensagem, peça para que o remetente reenvie a mensagem. Mas quando o objetivo é a confidencialidade da mensagem, esse problema possibilita um ataque ao QC-MDPC McEliece [GJS16].

O segundo problema, que tem relação com o primeiro, é que não é fácil estimar precisamente a probabilidade de o decodificador falhar ao decodificar uma mensagem usando somente técnicas teóricas [Gal62, RL09]. Em geral a avaliação de decodificadores LDPC é feita através de simulações [RL09, MOG15].

Os algoritmos de decodificação para códigos LDPC são classificados como baseados em decisões abruptas ou baseados decisões suaves. Ambos os tipos são iterativos, e tentam recuperar a mensagem original fazendo alterações locais[3] nas entradas da mensagem recebida. Idealmente, estas alterações são cada vez mais confiáveis a cada iteração. Num algoritmo baseado em decisões abruptas, a cada iteração, a mudança local que pode ocorrer em um bit é a inversão dele. Por este motivo, são chamados de algoritmos de *bit-flipping*. Em contraste, nas iterações de um algoritmo baseados em decisões suaves, considera-se a probabilidade de cada bit da mensagem recebida ser 0 ou 1.

O primeiro algoritmo de *bit-flipping* para decodificação foi introduzido por Gallager [Gal62]. Como este foi o algoritmo atacado que Guo et al. [GJS16] escolheram para formular seu ataque, sua descrição foi deixada para a Seção 4. Para dar ao leitor um exemplo de algoritmo decodificador, consideramos uma variante do algoritmo de Gallager sugerida por Huffman e Pless [HP10], que é descrita no Algoritmo 1. A cada iteração, é calculado o número f_j de nós de verificação vizinhos a v_j que estão insatisfeitos. Pode-se descrever alternativamente f_j como o número de equações de paridade às quais o bit j pertence e que estão insatisfeitas. No fim da iteração, cada bit j é invertido se e somente se $f_j \geq \max_i f_i$. O algoritmo realiza esses passos iterativamente até que tenha extraído os erros da mensagem, ou o número de iterações tenha excedido o máximo permitido `max_it`.

O exemplo a seguir pode ajudar a entender o funcionamento do algoritmo e a por que é útil a interpretação gráfica de códigos LDPC.

Exemplo 2.2.11. Considere o código $[10, 5]$-LDPC denotado por $\mathcal{C}$ que tem $\mathbf{H}$, a mesma considerada no exemplo anterior, como sua matriz de paridade esparsa

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

Seja $\mathbf{c} = [0\,1\,1\,0\,0\,0\,0\,0\,1\,1]$. Note que $\mathbf{c}$ pertence a $\mathcal{C}$ pois $\mathbf{c}\mathbf{H}^T = \mathbf{0}$. Suponha que o vetor de erro adicionado a $\mathbf{c}$ seja $\mathbf{e} = [0\,0\,0\,0\,0\,0\,0\,1\,0\,0]$, resultando $\mathbf{c}' = \mathbf{c} + \mathbf{e} = [0\,1\,1\,0\,0\,0\,0\,1\,1\,1]$.

A Figura 2.2.3 ilustra as iterações do algoritmo decodificador para recuperar $\mathbf{c}$ a partir de $\mathbf{c}'$. O primeiro grafo mostra os nós de variáveis assumindo os valores dos bits em $\mathbf{c}'$. O segundo grafo mostra as variáveis de verificação somando $(\mathrm{mod}\ 2)$ os valores dos nós de variáveis vizinhos. As que foram satisfeitas estão pintadas verdes, e as que não foram estão pintadas de vermelho. O terceiro grafo mostra as variáveis contando quantos de seus nós de verificação vizinhos não estão

[3] Dizemos que as alterações são locais pois, para decidir se uma alteração deve ou não ocorrer, não são levados em conta os valores de todas as posições do vetor, mas considera-se apenas um subconjunto delas.

Algoritmo 1: Variante do algoritmo de *bit-flipping* sugerido por Huffman e Pless [HP10]

Entrada: **H** matriz de paridade esparsa $r \times n$ de um código LDPC
y uma palavra transmitida a ser decodificada
`max_it` o número máximo de iterações

Saída: A decodificação estimada de **y**, ou $\perp$ se o número máximo de iterações for excedido

```
1  início
2      it ← 0
3      enquanto yH^T ≠ 0 e it < max_it faça
4          para cada j = 1, 2, ..., n faça
5              f_j ← Número de vizinhos de v_j insatisfeitos
6          max_upc ← max{f_j}
7          para cada j = 1, 2, ..., n faça
8              se f_j ≥ max_upc então
9                  y_j ← ȳ_j
10         it ← it + 1
11     se it ≥ max_it então
12         devolva ⊥
13     senão
14         devolva y
```

satisfeitos. Nota-se que a variável v_8 é a que está com mais vizinhos insatisfeitos, portanto seu valor é invertido. Finalmente no quarto grafo, os nós de verificação fazem novamente a soma de suas variáveis vizinhas, e nota-se que não há mais nós insatisfeitos. Então o algoritmo termina obtendo $[0,1,1,0,0,0,0,0,1,1]$ como vetor recuperado, que exatamente **c**.

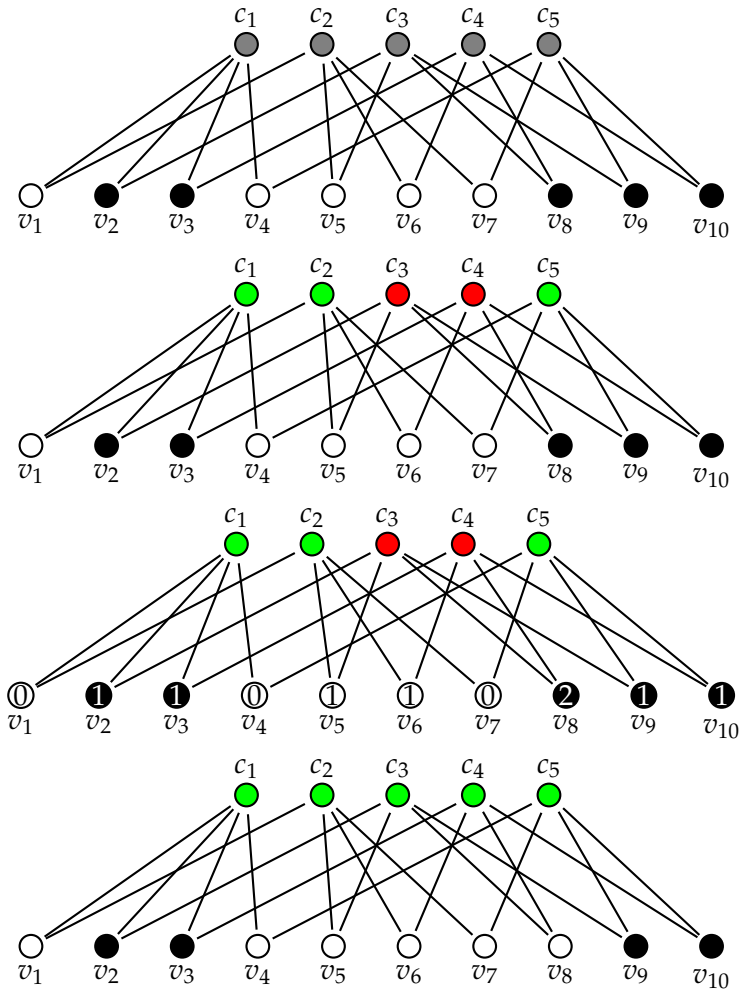

Figura 2.2.3: *Funcionamento do algoritmo de bit-flipping*

2.3 Criptografia de chave pública

Criptografia é o conjunto de técnicas que possibilitam a comunicação segura num meio compartilhado com adversários maliciosos. Pode-se definir vários conceitos de segurança para a comunicação, sendo que os principais objetivos de segurança relacionados à criptografia de chave pública são enumerados a seguir.

1. Confidencialidade: garante que somente o destinatário da mensagem pretendido pelo remetente consiga lê-la.

2. Autenticação: permite que um ou mais destinatários de uma mensagem consigam comprovar que esta foi gerada por um dado remetente.

3. Integridade: permite que um ou mais destinatários de uma mensagem consigam comprovar que esta não foi alterada na transmissão.

4. Não-repúdio: garante que o real remetente de uma mensagem não pode negar tê-la criado.

Note que o termo mensagem aqui é tomado num sentido amplo. Por exemplo, um documento estático numa página da web pode ser entendido como uma mensagem de um remetente para vários destinatários.

A confidencialidade da comunicação é conseguida por esquemas de encriptação, enquanto as outras três são conseguidas por esquemas de assinatura de mensagens. Neste trabalho estamos interessados somente em esquemas de encriptação, visto que este é o ponto forte do esquema de McEliece.

Esquemas de encriptação

Todo esquema de encriptação usa chaves, que são informações secretas que devem ser combinadas ao texto legível para encriptá-lo e ao texto encriptado para decriptá-lo. Os esquemas de encriptação se dividem em dois tipos: esquemas de chave secreta e esquemas de chave pública. Em esquemas de chave secreta, remetente e destinatário compartilham uma mesma chave, que é usada tanto na encriptação quanto na decriptação. Estes esquemas têm a desvantagem de exigir que remetente e destinatário devam, ao menos uma vez, ter acesso a um canal seguro de comunicação para combinar uma chave[4]. Apesar de serem conceitualmente bem mais antigos do que os de chave pública, há hoje esquemas de chave privada bem sofisticados, sendo o DES [DES77] e o AES [DR13] os dois principais representantes desta classe de esquemas.

Nos esquemas de chave pública, cada usuário possui duas chaves, uma chave secreta e outra pública, que são matematicamente relacionadas de forma que, idealmente, seja computacionalmente impraticável para um atacante obter a chave privada de um usuário a partir da sua chave pública. Para enviar uma mensagem para uma destinatária Alice, usa-se a chave pública de Alice para encriptar a mensagem, e Alice usa a sua chave secreta para decriptá-la. Esquemas de chave pública foram idealizados por Diffie e Hellman [DH76], que mostraram, supondo a dificuldade problema do logaritmo discreto, como usuários de uma rede poderiam decidir uma chave secreta comum sem a necessidade de um canal seguro, mas não mostraram como montar um esquema de encriptação. Os principais esquemas de encriptação de chave pública são o RSA de Rivest, Shamir, e Adleman [RSA78], que foi publicado poucos anos depois do trabalho de Diffie e Hellman, e os baseados em curvas elípticas [Mil86]. A seguir é dada uma definição formal de esquemas de encriptação de chave pública.

Definição 2.3.1 (Esquema de encriptação [Gal12])**.** Seja λ um inteiro positivo. Defina os seguintes espaços, todos dependentes de λ. $\mathcal{M}_\lambda$ é o conjunto de mensagens possíveis. $\mathcal{PK}_\lambda$ é o conjunto de possíveis chaves públicas. $\mathcal{SK}_\lambda$ é o conjunto das possíveis chaves secretas. $\mathcal{C}_\lambda$ é o conjunto de todos os textos encriptados possíveis. Diz-se que a tupla de algoritmos

$$(\text{GERACHAVES}, \text{ENCRIPTA}, \text{DECRIPTA})$$

é um esquema de encriptação de chave pública sobre os espaços definidos acima se todos os algoritmos são de tempo esperado polinomial em λ e as seguintes condições são obedecidas.

1. O algoritmo gerador de chaves GERACHAVES é um algoritmo probabilístico, toma λ como parâmetro, e devolve K_{SEC} e K_{PUB}, tais que $K_{\text{SEC}} \in \mathcal{SK}_\lambda$ e $K_{\text{PUB}} \in \mathcal{PK}_\lambda$.

2. O algoritmo de encriptação ENCRIPTA é um algoritmo probabilístico, que toma $\mathbf{m} \in \mathcal{M}_\lambda$ e $K_{\text{PUB}} \in \mathcal{PK}_\lambda$ como parâmetros, e devolve $\mathbf{c} \in \mathcal{C}_\lambda$.

3. O algoritmo de decriptação DECRIPTA é um algoritmo[5], que toma $\mathbf{c} \in \mathcal{C}_\lambda$ e $K_{\text{SEC}} \in \mathcal{SK}_\lambda$ como parâmetros, e devolve um valor $\mathbf{m} \in \mathcal{M}_\lambda$, ou o símbolo $\perp$. O símbolo $\perp$ em geral significa uma falha ao decriptar por conta de um texto encriptado inválido passado como argumento.

4. Os três algoritmos são relacionados da seguinte forma. Se K_{PUB} e K_{SEC} são saídas do algoritmo GERACHAVES(λ) para um certo λ, então vale que

$$\text{DECRIPTA}(\text{ENCRIPTA}(\mathbf{m}, K_{\text{PUB}}), K_{\text{SEC}}) = \mathbf{m},$$

para todo $\mathbf{m} \in \mathcal{M}$.

[4] A menos que sejam usadas primitivas típicas de esquemas de chave pública, como esquemas de compartilhamento de chave [DH76].

[5] Nada impede o algoritmo de decriptação de ser probabilístico, mas não é comum que seja.

Às vezes, como é o caso do QC-MDPC McEliece, a quarta condição é muito restritiva. Então podemos exigir apenas que esta condição valha com alta probabilidade, e também que, se

$$\text{DECRIPTA}(\text{ENCRIPTA}(\mathbf{m}, K_{\text{PUB}}), K_{\text{SEC}}) \neq \perp,$$

então

$$\text{DECRIPTA}(\text{ENCRIPTA}(\mathbf{m}, K_{\text{PUB}}), K_{\text{SEC}}) = \mathbf{m}.$$

Ou seja, se o algoritmo de decriptação não falhar, então ele decripta o texto encriptado para a mensagem original.

Os espaços e algoritmos associados ao esquema dependem de λ da seguinte forma: o melhor ataque conhecido ao esquema esquema deve terminar num número de passos maior ou igual a 2^{λ}. Por este motivo, λ é conhecido como o nível de segurança. Tipicamente, os tamanhos dos espaços crescem exponencialmente em função de λ, enquanto os algoritmos de encriptação, decriptação, e geração de chaves têm tempo de execução polinomial em λ.

Complexidade computacional

Como a chave pública é disponível para qualquer atacante, é fundamental para a segurança de um esquema criptográfico de chave pública que seja difícil encontrar a chave privada de um usuário a partir de sua pública. Para criar esquemas em que resolver este problema seja de fato difícil, uma ideia bem aceita na criptografia moderna é usar classes de problemas cujos melhores ataques não sejam algoritmos polinomiais.

Em teoria da complexidade, problemas computacionais são separados em classes que representam a sua dificuldade. As classes mais usadas são descritas em termos de problemas de decisão, ou seja, aqueles cuja resposta é sim ou não. Em geral, problemas que não são de decisão podem induzir facilmente problemas de decisão. Por exemplo, o problema de encontrar o produto entre dois inteiros a e b não é um problema de decisão. Porém um problema de decisão induzido por ele é decidir se $ab = k$ para um certo k. Algumas das principais classes de problemas são dadas a seguir. Note que a dificuldade de um problema é reduzida ao número máximo de passos que o melhor algoritmo possível dá para resolver a instância mais difícil deste problema.

1. $\mathcal{P}$ é a classe dos problemas de decisão que podem ser resolvidos por algoritmos polinomiais. Um exemplo simples de problema em $\mathcal{P}$ é decidir se um inteiro k pertence a uma lista com n inteiros.

2. $\mathcal{NP}$ contém problemas de decisão tais que cada uma de suas instâncias cuja resposta é sim admite um certificado compacto que permite que um algoritmo de tempo polinomial comprove que a resposta é de fato sim. Um certificado compacto é simplesmente uma cadeia de bits de tamanho $\mathcal{O}(n^c)$, onde n é o tamanho da instância do problema. Um exemplo de problema em $\mathcal{NP}$ é o problema de satisfazibilidade de fórmulas booleanas (SAT). Um certificado compacto de que uma fórmula é satisfazível é uma valoração de suas variáveis que tornam a fórmula satisfeita. A primeira definição da classe $\mathcal{NP}$ foi dada como o conjunto dos problemas de decisão que podem ser resolvidos polinomialmente por máquinas de Turing não determinísticas, do inglês *nondeterministic polynomial time*. Preferimos para este texto a primeira definição pois não usa máquinas não determinísticas.

3. *co*$\mathcal{NP}$ é a classe dos problemas de decisão tais que cada uma de suas instâncias cuja resposta é não admite um certificado compacto. Um exemplo de problema em co$\mathcal{NP}$ é o problema de decidir se uma fórmula booleana é uma tautologia. Um certificado compacto de que uma fórmula não é uma tautologia é uma valoração de suas variáveis que tornam a fórmula insatisfeita. Cada problema em $\mathcal{NP}$ induz um problema em co$\mathcal{NP}$, e vice-versa. Porém note que *co*$\mathcal{NP}$ não é o conjunto complementar da classe $\mathcal{NP}$.

Destas definições, temos que $\mathcal{P} \subseteq \mathcal{NP}$ e $\mathcal{P} \subseteq$ co$\mathcal{NP}$, já que, mesmo sem qualquer certificado, pode-se resolver um problema em $\mathcal{P}$ polinomialmente. Um dos maiores problemas abertos em

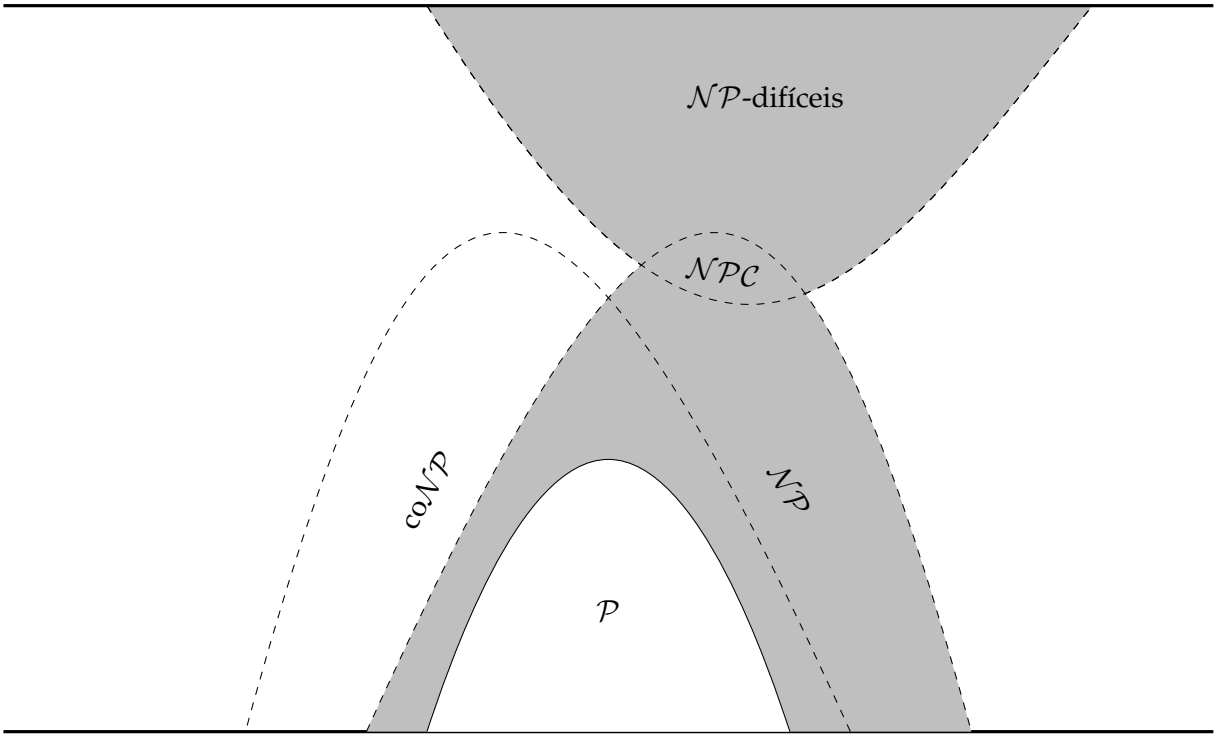

Figura 2.3.1: *Relações entre algumas das principais classes de problemas, supondo que $\mathcal{P} \neq \mathcal{NP}$ e $\mathcal{NP} \neq co\mathcal{NP}$, com destaque para a região onde está a maior parte dos problemas usados em criptografia*

teoria da computação é decidir se $\mathcal{P} = \mathcal{NP}$ ou não. Inclusive não se sabe nem se $\mathcal{NP} = co\mathcal{NP}$. Mas em geral, supõe-se que $\mathcal{P} \neq \mathcal{NP}$, e que $\mathcal{NP} \neq co\mathcal{NP}$. Se $\mathcal{P} = \mathcal{NP}$, então $\mathcal{NP} = co\mathcal{NP}$, porém se $\mathcal{NP} = co\mathcal{NP}$, pode ser o caso de $\mathcal{P} \neq \mathcal{NP}$.

Um importante subconjunto da classe $\mathcal{NP}$ é o dos problemas $\mathcal{NP}$-completos, denotado por $\mathcal{NPC}$. Cada problema deste conjunto tem a propriedade de que, se um dia for descoberto um algoritmo polinomial para resolvê-lo, então $\mathcal{P} = \mathcal{NP}$. Isso faz com que os problemas $\mathcal{NP}$-completos sejam considerados os problemas teoricamente mais difíceis em $\mathcal{NP}$. Uma classe que capta problemas, agora não só de decisão, pelo menos tão difíceis quanto os problemas mais difíceis em $\mathcal{NP}$ é a classe dos problemas $\mathcal{NP}$-difíceis. Esta classe contém os problemas $\mathcal{NP}$-completos, mas também contém outros problemas tais que, se for possível resolver algum deles eficientemente, então pode-se resolver problemas $\mathcal{NP}$-completos eficientemente. A menos que $\mathcal{P} = \mathcal{NP}$, a classe dos problemas $\mathcal{NP}$-difíceis contém problemas bem mais difíceis do que problemas $\mathcal{NP}$ - completos [AB09].

A Figura 2.3.1 mostra como a relação entre as classes de problemas apresentadas, supondo que $\mathcal{P} \neq \mathcal{NP}$ e $\mathcal{NP} \neq co\mathcal{NP}$. A área pintada de cinza contém a grande maioria, senão todos, dos problemas usados para a elaboração de esquemas criptográficos, em suas versões de decisão.

Por exemplo, considere a versão de decisão do problema da fatoração de inteiros, usado pelo RSA [RSA78]. Neste problema, nos é dado um número inteiro positivo grande n, e um outro inteiro positivo $m < n$. O problema é decidir se n tem algum divisor d tal que $1 < d \leq m$. Este problema está em $\mathcal{NP}$, pois, se a resposta for sim, basta mostrar um tal d como certificado. Para testar este certificado, basta ver se o resto da divisão de n por d é 0, o que pode ser feito eficientemente. Por outro lado, se a resposta for não, um possível certificado é a fatoração completa de $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, onde cada p_i é primos. Pode-se verificar polinomialmente que cada um dos fatores p_i é realmente primo, usando por exemplo o AKS [AKS04], e basta efetuar um produto para verificar se a fatoração apresentada de fato fatora n. Basta então verificar se cada um dos fatores primos de n são maiores do que m, aceitando o certificado se todos forem.

A versão de decisão do problema da fatoração também tem a seguinte propriedade: se existe um algoritmo polinomial para o problema de decisão, então existe um algoritmo polinomial

para fatorar um inteiro. Para provar esta afirmação, considere que existe um algoritmo polinomial DECIDEFATOR que resolve o problema de decisão para dois inteiros n e m, como no parágrafo anterior. Então podemos usar o algoritmo DECIDEFATOR para fazer uma busca binária sobre o conjunto[6] $[n]$ e encontrar o menor fator d, que é o menor fator primo, de n. Como DECIDEFATOR é polinomial no comprimento da representação de n, então sua complexidade é $\mathcal{O}\left((\log n)^c\right)$ para alguma constante c. A busca binária faz $\mathcal{O}(\log n)$ chamadas ao algoritmo DECIDEFATOR. Então o custo de todas essas operações para encontrar o primeiro fator primo de n é $\mathcal{O}\left((\log n)(\log n)^c\right) = \mathcal{O}\left((\log n)^{c+1}\right)$. Podemos repetir tal procedimento para encontrar um menor fator primo de $n/(d_1 \dots d_i)$ a cada fator d_i encontrado. Como $d_i \geq 2$, este procedimento seria executado no máximo $\log_2 n$ vezes. Então a complexidade de fatorar completamente n é $\mathcal{O}\left((\log n)^{c+2}\right)$, que é polinomial no comprimento da representação de n. Não se sabe se existem tais algoritmos polinomiais, mas conjectura-se que não existam.

Os dois parágrafos anteriores ilustram dois fatos importantes sobre os problemas usados em criptografia. Pode-se ver que as versões de decisão de problemas não são necessariamente muito mais fracas do que as suas formulações originais, então muitas vezes podemos falar em termos de problemas de decisão. Também é mostrado que a versão de decisão do problema de fatoração está em $\mathcal{NP} \cap \text{co}\mathcal{NP}$, uma classe que acredita-se não conter problemas $\mathcal{NP}$-completos. Assim, parece não ser necessário usar problemas $\mathcal{NP}$-difíceis para obter um esquema criptográfico forte, como é o caso do RSA [RSA78] que continua sendo usado mesmo depois de 40 anos desde sua publicação. Para o uso em criptografia, não só é importante que um algoritmo seja difícil em suas piores instâncias, mas que seja difícil na grande maioria de suas instâncias.

Apesar de ser um problema muito estudado, não se conhece algoritmo polinomial para fatorar inteiros em modelos de computação convencionais. Uma análise similar à mostrada para este problema pode ser feita para o problema do logaritmo discreto, usado, por exemplo, em criptografia com de curvas elípticas [Mil86]. Em suas versões de decisão, estes problemas não foram demonstrados $\mathcal{NP}$-completos, e suspeita-se que sejam mais difíceis do que problemas em $\mathcal{P}$, porém que de fato não sejam $\mathcal{NP}$-completos. Apesar disso, a confiança em esquemas criptográficos baseados nos problemas da fatoração e do logaritmo discreto foi abalada quando Shor publicou seus algoritmos quânticos para resolvê-los [Sho97]. Isso fez com que a principal ameaça a esquemas criptográficos baseados nesses problemas seja a construção de um computador quântico com um número razoável de bits quânticos [CCJ+16]. Os esquemas criptográficos pós-quânticos, isto é, que não sofreram ataques críticos por meio de algoritmos quânticos, são em geral baseados em problemas $\mathcal{NP}$-difíceis, mas que também foram suficientemente estudados para que seja estabelecida certa confiança de que são problemas difíceis em geral, e não apenas no pior caso.

É importante notar que, mesmo que um esquema seja baseado num problema $\mathcal{NP}$-difícil, isso não quer dizer que atacar um esquema criptográfico é resolver um problema $\mathcal{NP}$-difícil. O que em geral ocorre, e em particular ocorre para o esquema de McEliece, é que o portador de uma chave privada do esquema conhece uma instância do problema cuja resolução é fácil (por construção), porém a sua chave pública é uma instância embaralhada de forma que seja indistinguível de uma instância difícil.

Noções de segurança

Para analisar formalmente a segurança de esquemas criptográficos usam-se modelos formais para os objetivos de segurança e para os ataques. Os principais objetivos de segurança em criptografia de chave pública são dados a seguir.

1. Encriptação de mão única (OWE): dado um texto encriptado, o adversário não consegue decriptar a mensagem.

2. Indistinguibilidade (IND) [GM84]: dadas duas mensagens de mesmo comprimento, $\mathbf{m}_0$ e $\mathbf{m}_1$, o adversário não consegue distinguir entre os textos encriptados de $\mathbf{m}_0$ e $\mathbf{m}_1$, respectiva-

[6] A busca poderia ser feita no conjunto $[\sqrt{n}]$, mas optamos por $[n]$ para simplificar a descrição.

mente. Uma definição alternativamente é que o adversário não consegue obter informação sobre uma mensagem, a menos de seu comprimento, a partir de sua encriptação.

3. Imaleabilidade (NM) [DDN03]: dado um texto encriptado $\mathbf{c}$, o adversário não consegue alterá-lo para $\mathbf{c}'$ de forma que os textos decriptados de $\mathbf{c}$ e $\mathbf{c}'$ sejam relacionados. Ou seja, conhecer $\mathbf{c}$ não deixa mais fácil gerar um tal texto encriptado $\mathbf{c}'$.

Os modelos de ataque representam formas diferentes de o atacante interagir com a chave privada. Podem ser mais fracos, em que o atacante conhece apenas a chave pública, ou mais fortes, em que o atacante pode pedir para um oráculo decriptações de textos arbitrários, a menos talvez da decriptação de um texto encriptado passado como desafio. Os principais modelos de atacantes são os seguintes.

1. Ataque passivo (CPA): o atacante conhece apenas a chave pública. Também é conhecido como ataque de texto encriptado escolhido, do inglês *chosen ciphertext attack*. Em esquemas de chave pública, todos os usuários são potencialmente atacantes CPA. Então qualquer esquema de chave pública considerável deve no mínimo ser seguro contra este tipo de ataque.

2. Ataque adaptativo de texto cifrado escolhido (CCA1) [NY90]: o atacante conhece a chave pública e pode pedir que sejam decriptados textos encriptados arbitrários por um período de tempo. Após este período de tempo, o atacante não pode mais interagir com o decriptador, e lhe é dado um desafio, de acordo com o objetivo de segurança desejado. O atacante então tenta resolver o desafio com uma probabilidade não negligenciável.

3. Ataque adaptativo de texto cifrado escolhido (CCA2) [RS91]: é similar ao ataque CCA1, com o atacante podendo pedir decriptações de textos arbitrários para o portador da chave pública por certo período de tempo, recebendo o desafio ao final do período. A diferença é que o atacante pode continuar pedindo decriptações de textos arbitrários, exceto os textos encriptados do desafio. Diz-se que o ataque é adaptativo pois agora o atacante pode adaptar os pedidos de decriptação ao desafio, o que não ocorre para atacantes CCA1.

Em geral ataques CCA1 são ataques de recuperação de chave, em que o atacante usa os textos decriptados conseguidos para inferir bits, ou correlações entre bits, da chave. Em particular, para o ataque de recuperação de chave ao QC-MDPC McEliece [GJS16], o atacante envia textos encriptados ao portador da chave privada, e verifica se o texto foi ou não decriptado. Como o atacante não precisa saber qual foi o texto decriptado, apenas a reação do decriptador, o ataque usa menos informação que ataques CCA1, e portanto é mais forte do que estes.

A segurança de um esquema é descrita na forma "objetivo do ataque-modelo de ataque". Por exemplo, se um esquema permite indistinguibilidade de textos encriptados contra um ataque adaptativo de texto encriptado escolhido, diz-se que ele tem segurança IND-CCA2. Similarmente, chama-se também de ataque IND-CCA2 um ataque que objetiva distinguir textos encriptados interagindo com o decriptador na forma CCA2.

Esquemas criptográficos que não são OWE certamente não podem ser usados de forma segura, pois os adversários podem decriptar as mensagens. Então as noções mais interessantes contra esquemas de chave pública são aquelas em que os objetivos são IND ou NM, e os atacantes são CPA, CCA1, ou CCA2. A Figura 2.3.2 mostra as relações entre tais noções de segurança descobertas por Bellare et al. [BDPR98]. Nesta figura, uma noção de segurança implica outra se e somente se há um caminho da primeira até a segunda seguindo os arcos direcionados.

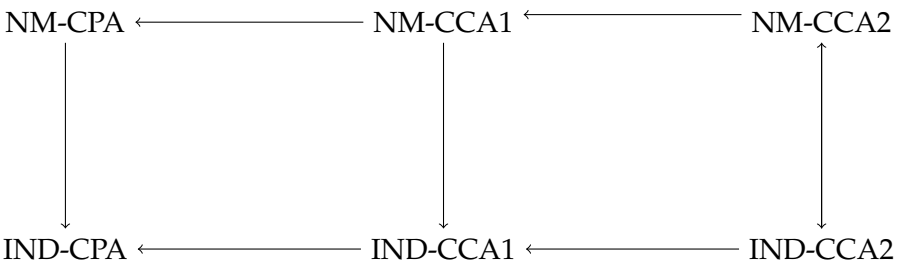

Figura 2.3.2: *Relações entre algumas noções de segurança [BDPR98, Figura 1 adaptada]. Uma noção de segurança A implica outra B se e somente se há um caminho de A a B*

2.4 Esquema de McEliece

O esquema de McEliece [McE78] é um algoritmo de encriptação de chave pública. O esquema é baseado em códigos corretores de erros. Sua segurança é fundamentada no problema $\mathcal{NP}$-difícil da decodificação por síndrome. Foi o primeiro algoritmo a usar um procedimento aleatório para a encriptação [MVOV96], porém não com a intenção de deixá-lo semanticamente seguro [GM84]. A encriptação e a decriptação são mais eficientes do que as respectivas operações do RSA [RSA78] e curvas elípticas [Mil86]. Porém sua usabilidade é prejudicada pelos grandes tamanhos de suas chaves públicas.

A chave pública de um usuário do esquema é a matriz geradora $\mathbf{G}$ de um código $\mathcal{C}$. A ideia de seu algoritmo de encriptação é codificar a mensagem usando $\mathbf{G}$ e adicionar um certo número de erros em posições aleatórias de um vetor de um código linear $\mathcal{C}$. Idealmente este vetor com erros só poderia ser decodificado por um portador de um decodificador eficiente para $\mathcal{C}$. Assim, a segurança do esquema depende da escolha de uma família de códigos tais que não se saiba construir um decodificador eficiente apenas a partir da matriz geradora do código. A família de códigos originalmente sugerida por McEliece é a dos códigos de Goppa [Gop70, Ber73].

A Tabela 2.4.1 mostra os parâmetros dos códigos de Goppa e os tamanhos das chaves públicas, para cada nível de segurança [BCS13].

λ	n	k	t	m	Tamanho da chave pública (Bytes)
81	2048	1751	27	11	65006
95	2048	1608	40	11	88440
105	2480	1940	45	12	130950
119	2690	2018	56	12	169512
146	3408	2604	67	12	261702
187	4624	3389	95	13	523177
263	6960	5413	119	13	1046739

Tabela 2.4.1: *Parâmetros de famílias de códigos de Goppa para cada nível de segurança λ [BCS13].*

Descrição do esquema

O Esquema de McEliece é uma tripla de algoritmos (GERACHAVES, ENCRIPTA, DECRIPTA), em que cada algoritmo é definido a seguir.

GERACHAVES é o algoritmo probabilístico de geração do par de chaves. Dado um nível de segurança λ, siga os seguintes passos.

1. Escolha uma tripla de parâmetros (n, t, k) de uma família $\mathcal{F}$ de códigos de Goppa que suporta um nível de segurança λ, usando por exemplo a Tabela 2.4.1.

2. Tome $\overline{\mathbf{G}}$ e g, que são respectivamente, uma matriz geradora e um polinômio de Goppa do código $\mathcal{G}$ escolhido aleatoriamente da família $\mathcal{F}$.

3. Escolha aleatoriamente $\mathbf{S}$, uma matriz binária $k \times k$ não singular, e $\mathbf{P}$, uma matriz de permutação $n \times n$. Note que ambas as matrizes são binárias.

4. Faça $\mathbf{G} = \mathbf{S}\overline{\mathbf{G}}\mathbf{P}$, e $\hat{\mathbf{G}} = \mathbf{S}\overline{\mathbf{G}}$.

5. Devolva o par de chaves secreta e pública, dadas respectivamente por

$$K_{\text{SEC}} = (\hat{\mathbf{G}}, \mathbf{P}, g), \quad \text{e} \quad K_{\text{PUB}} = (\mathbf{G}, t).$$

ENCRIPTA é o algoritmo probabilístico de encriptação. Dada uma mensagem $\mathbf{m}$, e a chave pública do destinatário K_{PUB}, siga os seguintes passos.

1. Separe os elementos da chave pública em $(\mathbf{G}, t) = K_{\text{PUB}}$.

2. Tome um vetor $\mathbf{e}$ aleatório de peso t de $\mathbb{F}_2^n$.

3. Devolva o vetor $\mathbf{c} = \mathbf{mG} + \mathbf{e}$.

DECRIPTA é o algoritmo determinístico de decriptação. Dado um texto cifrado $\mathbf{c}$ e a chave secreta do destinatário K_{SEC}, siga os seguintes passos.

1. Separe os elementos da chave secreta em $(\hat{\mathbf{G}}, \mathbf{P}, g) = K_{\text{SEC}}$.

2. Obtenha o decodificador Φ a partir de g.

3. Calcule $\mathbf{c}_1 = \mathbf{cP}^{-1}$. Assim, $\mathbf{c}_1$ representa

$$\mathbf{cP}^{-1} = (\mathbf{mG} + \mathbf{e})\mathbf{P}^{-1} = \mathbf{m}\hat{\mathbf{G}} + \mathbf{eP}^{-1}.$$

4. Calcule $\mathbf{c}_2 = \Phi(\mathbf{c}_1)$, que é igual a $\mathbf{m}\hat{\mathbf{G}}$.

5. Devolva $\mathbf{m}$ que é a solução do sistema sobredeterminado $\mathbf{m}\hat{\mathbf{G}} = \mathbf{c}_2$.

Segurança do esquema

A segurança do esquema de McEliece se baseia em duas hipóteses:

1. não existe algoritmo eficiente capaz de decodificar códigos lineares aleatórios;

2. as matrizes públicas de códigos de Goppa (possivelmente embaralhadas) são indistinguíveis de matrizes aleatórias.

A primeira hipótese é relativamente bem aceita, pois sabe-se que o problema de decodificar códigos lineares aleatórios é $\mathcal{NP}$-difícil [BMVT78], e, embora seja um problema importante e bem estudado, os melhores decodificadores de códigos aleatórios conhecidos são algoritmos exponenciais [OS09]. A segunda hipótese foi quebrada para códigos de $[n, k]$-lineares de Goppa com k próximo de n [FGUO+13]. Como este caso não é interessante para o esquema de McEliece, tais distinguidores não afetam significativamente o esquema.

Por ser baseado na impossibilidade de decodificar eficientemente códigos aleatórios, o esquema de McEliece é ameaçado por avanços em algoritmos genéricos de decodificação. Os melhores decodificadores conhecidos para códigos lineares aleatórios são decodificadores por conjunto de informação [Pra62].

Como exemplo, é dado a seguir um simples algoritmo genérico para decodificação. Sejam $\mathbf{G}$ uma matriz $k \times n$ geradora da chave pública de um usuário esquema de McEliece, $\mathbf{m}$ a mensagem secreta, e $\mathbf{c} = \mathbf{mG} + \mathbf{e}$ a mensagem encriptada, usando algum erro $\mathbf{e} = [e_1 e_2 \dots e_n]$ de peso t.

Seja I um conjunto de k índices quaisquer do vetor $\mathbf{c}$. Se $e_i = 0$ para todo índice i de I, e se a submatriz $\mathbf{G}_I$ de $\mathbf{G}$ formada pelas colunas k de índices em i for inversível, um atacante pode obter facilmente a mensagem encriptada $\mathbf{m}$. Basta fazer a multiplicação $\mathbf{m} = \mathbf{c}_I\mathbf{G}_I^{-1}$, onde $\mathbf{c}_I$ denota o vetor formado pelas colunas de $\mathbf{c}$ com índices em I. A probabilidade de um atacante escolher k índices i em que $e_i = 0$ é $\binom{n-t}{k}/\binom{n}{k}$. Então o número esperado de tentativas que ele deve fazer é $\binom{n}{k}/\binom{n-t}{k}$.

Há algoritmos similares para a decodificação de códigos lineares que podem ser mais eficientes [Pra62, LB88, Ste88, FS09]. Porém mesmo para eles, através da escolha de n, k, e t, para cada nível de segurança λ, pode-se fazer o ataque levar tempo esperado exponencial em função de λ.

Apesar de a encriptação ser probabilística, pode-se ver que o esquema de McEliece não provê indistinguibilidade de textos legíveis escolhidos (IND-CPA), que é uma noção básica de segurança para esquemas de chave pública. Tome duas mensagens $\mathbf{m}_1$ e $\mathbf{m}_2$, e um cifrado $\mathbf{c}$ de uma das mensagens,escolhida ao acaso. Para determinar qual mensagem foi encriptada para obter $\mathbf{c}$, basta calcular $\mathbf{d}_1 = \mathbf{m}_1\mathbf{G} + \mathbf{c}$, e $\mathbf{d}_2 = \mathbf{m}_2\mathbf{G} + \mathbf{c}$, e se $\mathrm{w}(\mathbf{d}_1) = t$, então $\mathbf{c} = \mathbf{m}_1\mathbf{G}$. Caso contrário, $\mathbf{c} = \mathbf{m}_2\mathbf{G}$. A segurança do esquema é ainda mais afetada quando parte do texto legível é conhecido, o que pode diminuir significativamente o número esperado de operações necessárias para recuperar a mensagem usando decodificadores por conjunto de informação [KI01].

Assim, para usar o esquema de McEliece de forma segura, é necessário fazer adaptações no esquema, chamadas conversões de segurança, objetivando atingir segurança contra ataques IND-CCA2. Há conversões genéricas, como a de Pointcheval [Poi00], e a de Fujisaki e Okamoto [FO99], que são aplicáveis ao esquema de McEliece. Porém, por serem genéricas, não usam o esquema de McEliece do modo mais eficiente possível, e portanto há redundância de dados[7] maior do que necessária.

Kobara e Imai [KI01] mostraram três conversões eficientes, chamadas α, β, e γ, que convertem o esquema de McEliece num esquema IND-CCA2, sob o modelo de oráculos aleatórios [BR93]. As suas conversões específicas chegam a ter redundância de dados até 4 vezes menor do que as genéricas, e ainda, para alguns parâmetros, a conversão γ pode ser até mais eficiente em uso de dados do que o McEliece original sem conversões, pois maximiza a quantidade de informação no vetor de erro. A Tabela 2.4.2 mostra a redundância de dados para cada conversão, com diferentes parâmetros (n, k, t) do esquema de McEliece.

Conversão	Redundância de dados para parâmetros (n, k, t)		
	(1024, 644, 38)	(2048, 1289, 69)	(4096, 2560, 128)
Pointcheval	1184	2208	4256
Fujisaki e Okamoto	1024	2048	4096
Kobara e Imai α e β	540	919	1696
Kobara e Imai γ	470	648	1040
McEliece original	380	759	1536

Tabela 2.4.2: *Comparação da redundância de dados para as conversões genéricas e específicas [KI01, Tabela 1 adaptada].*

Por ser a mais simples, mostramos aqui a conversão α, cuja encriptação é dada pelo Algoritmo 2, e a decriptação pelo Algoritmo 3. Nestas descrições, as funções HASH e PRNG representam, respectivamente, uma função de hash ideal, e um gerador ideal de números pseudo-aleatórios a partir de uma semente passada como argumento. Supõe-se que a saída de PRNG é um vetor de bits aleatórios. ESQUERDA$(\mathbf{c}, k)$ representa as k primeiras entradas de $\mathbf{c}$. A função auxiliar INTEIROPARAVETOR(z) codifica o número z num vetor de peso t e comprimento n. Algumas alternativas para implementar esta função foram discutidas por Overbeck e Sendrier [OS09].

[7] A redundância de dados em uma conversão é a diferença entre os comprimentos do texto cifrado e do texto legível.

A função VETORPARAINTEIRO($\mathbf{e}$) denota a inversa da função INTEIROPARAVETOR(z). Como outras conversões de segurança sob o modelo de oráculos aleatórios, a ideia é fazer com que decriptar mensagens aleatórias encriptadas pelo esquema de McEliece é ao menos tão difícil para um atacante quanto decriptar mensagens não aleatórias, que o atacante até pode conhecer parcialmente. Além disso, a conversão impõe uma certa relação entre o erro $\mathbf{e}_z$ e a mensagem $\mathbf{m}$, em particular VETORPARAINTEIRO($\mathbf{e}_z$) = HASH($[\mathbf{r}|\mathbf{m}]$), de tal forma que não é fácil para um atacante alterar o texto encriptado sem invalidar esta relação.

Algoritmo 2: ENCRIPTA$_\alpha$: conversão IND-CCA2 de ENCRIPTA [KI01]

Entrada: $\mathbf{m}$ mensagem que Beto quer enviar a Alice
K_{PUB} a chave pública de Alice
Saída: $\mathbf{c}$ o texto cifrado da mensagem $\mathbf{m}$

início
  $(\mathbf{G}, t) \leftarrow K_{\text{PUB}}$
  $\mathbf{r} \leftarrow$ vetor de bits aleatórios ($\approx$ 160 bits)
  $z \leftarrow$ HASH($[\mathbf{r}|\mathbf{m}]$)
  $\mathbf{y} \leftarrow$ PRNG(z) + $[\mathbf{r}|\mathbf{m}]$
  $\mathbf{e}_z \leftarrow$ INTEIROPARAVETOR(z)
  $\mathbf{c} \leftarrow (\mathbf{yG} + \mathbf{e}_z)$
  devolva $\mathbf{c}$

Algoritmo 3: DECRIPTA$_\alpha$: conversão IND-CCA2 de DECRIPTA [KI01]

Entrada: c texto cifrado recebido por Alice
K_{SEC} a chave secreta de Alice
Saída: $\mathbf{m}$ a decifração de $\mathbf{c}$

início
  $\mathbf{y} \leftarrow$ DECRIPTA(ESQUERDA($\mathbf{c}, k$), K_{SEC})
  $\mathbf{e}_z \leftarrow$ ESQUERDA($\mathbf{c}, k$) + $\mathbf{yG}$
  $z \leftarrow$ VETORPARAINTEIRO($\mathbf{e}_z$)
  $[\mathbf{r}|\mathbf{m}] \leftarrow$ PRNG(z) + $\mathbf{y}$
  se $z =$ HASH($[\mathbf{r}|\mathbf{m}]$) **então**
    devolva $\mathbf{m}$
  senão
    devolva $\perp$

Capítulo 3

QC-MDPC McEliece

Em 2013, uma nova variante do Esquema de McEliece foi proposta por Misoczki et al. [MTSB13]. Esta variante usa códigos QC-MDPC, que são códigos lineares binários que admitem uma matriz de paridade quase cíclica e moderadamente esparsa, em contraste com as de baixa densidade dos códigos LDPC, do inglês *Low Density Parity Check*, de Gallager [Gal62].

Uma grande diferença entre códigos MDPC e a maioria dos outros códigos sugeridos para serem usados com o esquema de McEliece [Gab05, BCGO09, MB09], é que como os códigos LDPC, os códigos MDPC não têm estrutura algébrica. Isso dá certa confiança de que os esquemas usando esses códigos resistam a ataques de criptanálise algébrica [FOPT10, FOP+16].

Apesar serem similares a códigos aleatórios em sua construção, códigos LDPC não são adequados para o uso no esquema de McEliece [SMR00]. Suas matrizes de paridade têm peso muito baixo, o que permite que algoritmos que buscam palavras de baixo peso recuperem linhas das matrizes de paridade sem muita dificuldade [SMR00]. A ideia dos códigos MDPC é fortalecer o esquema de McEliece com códigos LDPC, aumentando a densidade da matriz de paridade de forma que a busca por linhas esparsas da matriz de paridade seja impraticável, obtendo códigos chamados MDPC. Como as chaves públicas do MDPC ainda são muito grandes, Misoczki et al. mostraram como introduzir certa ciclicidade nas matrizes para obter chaves compactas comparáveis às do RSA, mas de modo que preservasse a segurança do esquema.

Uma implementação eficiente do QC-MDPC McEliece pode ser vista no trabalho de Maurich et al. [MOG15]. Há duas variantes do esquema QC-MDPC McEliece: QC-MDPC McEliece suave [BSC16], e QC-MDPC McEliece p-ário [GJ16]. O esquema QC-MDPC McEliece suave é muito similar ao QC-MDPC McEliece, porém usa decodificadores baseados em decisões suaves, e os erros inseridos não são binários, mas números de ponto flutuante. Isso permite inserir informação sobre a mensagem nos erros, de modo a obter chaves ainda menores que o QC-MDPC McEliece. Infelizmente, o esquema pode sofrer ataques por decodificação de mensagens, e também é possível atacar o esquema por um ataque de reação similar ao de Guo et al. [GJS16] para o esquema de McEliece [FHS+17]. O esquema QC-MDPC McEliece p-ário usa códigos QC-MDPC em que as entradas não são binárias, mas pertencem ao corpo finito de p elementos. O esquema foi proposto muito recentemente, então não foi muito estudado. Mas parece resistir ao ataque de reação de Guo et al. contra o QC-MDPC McEliece[8].

3.1 Códigos QC-MDPC

Definição 3.1.1 (Códigos MDPC [MTSB13])**.** Um código (n, r, w)-MDPC é um código linear de comprimento n, e codimensão[9] r, que admite uma matriz de verificação de paridade $\mathbf{H}$ cujas linhas têm mesmos pesos iguais a $w = \mathcal{O}\left(\sqrt{n \log n}\right)$.

[8] O QC-MDPC p-ário foi proposto como alternativa ao QC-MDPC McEliece por dois dos autores responsáveis pelo ataque de reação contra este esquema.

[9] A codimensão de um código é a dimensão de seu espaço dual, ou equivalentemente o número de linhas de uma sua matriz de paridade.

Para conseguir chaves compactas usando códigos MDPC, Misoczki et al. sugerem que seja usada uma família de códigos MDPC cujas matrizes de paridade são quase-cíclicas. Uma matriz é quase cíclica se é uma matriz de blocos, com cada um de seus blocos sendo uma matriz cíclica. Uma matriz cíclica é uma matriz tal que cada uma de suas linhas é uma rotação circular de uma posição à direita da linha superior. Dessa forma, matrizes cíclicas podem ser completamente descritas apenas por sua primeira linha. Como exemplo, as seguinte matrizes $\mathbf{C}_0$ e $\mathbf{C}_1$ são cíclicas, enquanto a matriz $\mathbf{C}$ é quase cíclica

$$\mathbf{C}_0 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \mathbf{C}_1 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}, \mathbf{C} = [\mathbf{C}_0 | \mathbf{C}_1].$$

Definição 3.1.2 (Códigos QC-MDPC [MTSB13])**.** Um código $[n, r, w]$-QC-MDPC é um código (n, r, w)-MDPC que admite uma matriz de verificação de paridade quase cíclica $\mathbf{H}$. Para isso, n deve ser um múltiplo de r, e o número de blocos cíclicos de $\mathbf{H}$ é denotado por $n_0 = n/r$.

Por definição, a matriz de paridade de um código (n, r, w)-QC-MDPC é da forma

$$\mathbf{H} = [\, \mathbf{H}_0 \,|\, \mathbf{H}_1 \,|\, \ldots \,|\, \mathbf{H}_{n_0-1} \,],$$

onde cada matriz $\mathbf{H}_i$ é cíclica. Então, sabendo-se a codimensão r, a matriz $\mathbf{H}$ é completamente descrita pela sua primeira linha. Supondo-se que $\mathbf{H}_{n_0-1}$ é inversível, o que será garantido pelo algoritmo de construção de códigos MDPC, pode-se verificar, comprovando que $\mathbf{G}\mathbf{H}^T = \mathbf{0}$, que uma possível matriz geradora $\mathbf{G}$ do código de matriz de paridade $\mathbf{H}$ é

$$\mathbf{G} = \left[\begin{array}{c|c} & \left(\mathbf{H}_{n_0-1}^{-1} \cdot \mathbf{H}_0\right)^T \\ & \left(\mathbf{H}_{n_0-1}^{-1} \cdot \mathbf{H}_1\right)^T \\ \mathbf{I} & \vdots \\ & \left(\mathbf{H}_{n_0-1}^{-1} \cdot \mathbf{H}_{n_0-2}\right)^T \end{array}\right]. \tag{3.1.1}$$

Usar a matriz geradora pública em forma sistemática, como está a $\mathbf{G}$ apresentada, não permite transmissão segura de mensagens não aleatórias. Mas usando conversões de segurança como as de Kobara e Imai [KI01], não há nenhuma perda de segurança em usar a matriz nessa forma. Então pelo resto deste texto, supomos que a matriz $\mathbf{G}$ é dada na forma sistemática, e então para descrevê-la basta dar o O que permite a representação compacta da chave pública do QC-MDPC McEliece é o fato de que cada bloco $\mathbf{H}_{n_0-1}^{-1} \cdot \mathbf{H}_i$ da matriz $\mathbf{G}$ é cíclico [MTSB13].

Construção do código

Para construir um código $[n, r, w]$-QC-MDPC aleatório, onde n é um múltiplo de r, escolha aleatoriamente um vetor binário $\mathbf{h}$, de comprimento n e peso w. Divida $\mathbf{h}$ em $n_0 = n/r$ partes $\mathbf{h}_0, \ldots, \mathbf{h}_{n_0-1}$, todas de comprimento r, de forma que $\mathbf{h} = [\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n_0-1}]$. Construa a matriz de paridade $\mathbf{H}$ na forma

$$\mathbf{H} = [\, \mathbf{H}_0 \,|\, \mathbf{H}_1 \,|\, \ldots \,|\, \mathbf{H}_{n_0-1} \,],$$

onde cada matriz $\mathbf{H}_i$ é a matriz cíclica que tem como primeira linha o vetor $\mathbf{h}_i$. Se o bloco $\mathbf{H}_{n_0-1}$ não for inversível, o que ocorre com baixa probabilidade [MOG15], escolha um novo vetor $\mathbf{h}$ e recomece o procedimento. Agora a matriz geradora pode ser construída usando a equação 3.1.1.

Um modo simples de agilizar a construção de códigos QC-MDPC, é observar que a álgebra das matrizes cíclicas $r \times r$ é isomorfa à dos polinômios em $\mathbb{F}_2[x]$ módulo $x^r - 1$. Dessa forma, as multiplicações e inversões de matrizes cíclicas podem ser computadas mais eficientemente [MTSB13].

Tipicamente, as matrizes de paridade dos códigos usados no QC-MDPC McEliece que oferecem menores tamanhos de chave têm apenas dois blocos. Como exemplo, considere as matrizes de paridade e geradora de um código $(40, 20, 6)$-QC-MDPC mostradas, respectivamente, à esquerda e à direita na Figura 3.1.1. Nesta figura, retângulos mais escuros representam 1's, enquanto os mais claros representam 0's. Note que a matriz geradora não é esparsa.

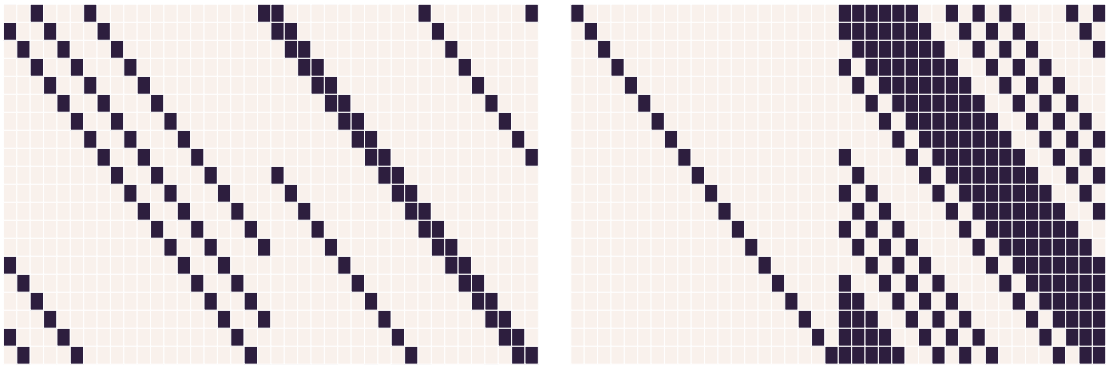

Figura 3.1.1: *Matrizes de paridade e geradora, respectivamente à esquerda e à direita, de um código* $(40, 20, 6)$*-QC-MDPC*

Codificação

A codificação de um vetor $\mathbf{m}$ é simplesmente o vetor $\mathbf{c}$, produto da multiplicação $\mathbf{c} = \mathbf{mG}$. Como $\mathbf{G}$ em geral está na forma sistemática $\mathbf{G} = [\,\mathbf{I} \mid \mathbf{P}\,]$, para alguma matriz $\mathbf{P}$, a codificação é ligeiramente simplificada para $\mathbf{c} = [\mathbf{m}|\mathbf{mP}]$.

Decodificação

Por serem baseados nos códigos LDPC, é comum decodificar códigos MDPC com algoritmos daqueles códigos. Mas como as matrizes de verificação de paridade de códigos MDPC são mais densas, é de se esperar que os algoritmos de decodificação para códigos LDPC não sejam tão eficientes. Isso motivou algumas melhorias tanto para diminuir o número de iterações para a decodificação, quanto para diminuir a probabilidade de falha.

Há duas classes de decodificadores para códigos LDPC: os por decisões abruptas, e os por decisões suaves. Com algoritmos de decisão abrupta, a decodificação de uma mensagem transmitida é feita através de sucessivas inversões de seus bits, até possivelmente chegar à mensagem original. Por isso, estes algoritmos são chamados de *bit-flipping*. Os decodificadores por decisão suave usam algoritmos de propagação de crença [Mac99] sobre a probabilidade de cada bit da mensagem transmitida ser 1 ou não.

Ambas as classes de algoritmos podem falhar com certa probabilidade, sendo que tipicamente algoritmos de *bit-flipping* têm taxas de falha mais altas do que algoritmos de propagação de crença [Mac99]. Apesar de serem capazes de corrigir mais erros, os decodificadores por decisões suaves são significativamente mais lentos do que os de *bit-flipping*. Por isso, e por terem implementação e análise mais fáceis, os decodificadores por decisão abrupta são mais usados nas implementações [MOG15, HVMG13]. Em particular, Misoczki et al. sugerem o uso do decodificador por *bit-flipping* descrito pelo Algoritmo 4. Para uma análise experimental detalhada de variantes do algoritmo de *bit-flipping* aplicados a códigos QC-MDPC, veja os estudo de Hayse et al. [HVMG13], e Maurich et al. [MOG15].

Note como o Algoritmo 4 é similar ao Algoritmo 1 analisado na Seção 2.2. A única diferença é que agora é usado um δ que aumenta o número de bits que são invertidos a cada iteração, podendo fazer o algoritmo terminar num menor número de passos. O valor ótimo de delta deve ser estimado através de simulações.

Algoritmo 4: Algoritmo de *bit-flipping* sugerido Misockzi et al. [MTSB13]

Entrada: **H** matriz de paridade esparsa $r \times n$ de um código LDPC
y uma palavra transmitida a ser decodificada
`max_it` o número máximo de iterações

Saída: A decodificação estimada de **y**, ou $\perp$ se o número máximo de iterações for excedido

início
  `it` $\leftarrow 0$
  enquanto $\mathbf{y}\mathbf{H}^T \neq \mathbf{0}$ *e* `it` $<$ `max_it` **faça**
    para *cada* $j = 1, 2, \ldots, n$ **faça**
      $f_j \leftarrow$ Número de vizinhos de v_j insatisfeitos
    `max_upc` $\leftarrow \max\{f_j\}$
    para *cada* $j = 1, 2, \ldots, n$ **faça**
      se $f_j \geq$ `max_upc` $- \delta$ **então**
        $y_j \leftarrow \overline{y_j}$
    `it` $\leftarrow$ `it` $+ 1$
  se `it` $\geq$ `max_it` **então**
    devolva $\perp$
  senão
    devolva **y**

3.2 QC-MDPC McEliece

Definição 3.2.1 (Esquema de McEliece)**.** O *Esquema QC-MDPC McEliece* é uma tripla de algoritmos (GERACHAVES, ENCRIPTA, DECRIPTA), em que cada algoritmo é definido como os seguintes.

GERACHAVES é o algoritmo para geração de um par de chaves. Dado um nível de segurança λ, siga os seguintes passos.

1. Escolha os parâmetros n, r, w, e t de uma família $\mathcal{F}$ de códigos de QC-MDPC que suporta um nível de segurança λ usando, por exemplo, a Tabela 3.3.1.
2. Tome **H** uma matriz de paridade de um código escolhido aleatoriamente da família $\mathcal{F}$.
3. Calcule **G** sistemática tal que $\mathbf{G}\mathbf{H}^T = \mathbf{0}$ usando a equação 3.1.1.
4. Construa $\mathbf{h}_i$, e $\mathbf{g}_j$, que são as primeiras linhas de cada bloco cíclico de **H** e **G**, respectivamente. Faça
$$\mathbf{h} = [\mathbf{h}_0|\mathbf{h}_1|\ldots|\mathbf{h}_{n_0-1}], \qquad \text{e} \qquad \mathbf{g} = [\mathbf{g}_0|\mathbf{g}_1|\ldots|\mathbf{g}_{n_0-2}].$$
5. Devolva o par de chaves secreta e pública, dados respectivamente por
$$K_{\text{SEC}} = \mathbf{h}, \quad \text{e} \quad K_{\text{PUB}} = \mathbf{g}.$$

ENCRIPTA é o algoritmo de encriptação. Dada uma mensagem **m**, e a chave pública do destinatário K_{PUB}, siga os seguintes passos.

1. Tome o vetor **g** de K_{PUB} que permite reconstruir a matriz geradora pública.
2. Construa a matriz geradora sistemática **G** usando as primeira linhas de cada bloco cíclico **g**.
3. Tome um **e**, um vetor de peso t aleatoriamente escolhido de $\mathbb{F}_2^n$.
4. Devolva o vetor $\mathbf{c} \leftarrow \mathbf{m}\mathbf{G} + \mathbf{e}$.

DECRIPTA é o algoritmo de decriptação. Dado um texto encriptado **c** e a chave secreta do destinatário K_{SEC}, siga os seguintes passos.

1. Construa a matriz de paridade **H** usando a primeira linha h e de K_{SEC} e o tamanho de cada bloco r.

2. Use o algoritmo de *bit-flipping* com a matriz esparsa **H** para decodificar **c** e obter **m**.

3. Se o algoritmo falhar, peça para o remetente reenviar a mensagem, que com alta probabilidade será encriptada com um vetor de erro que o decodificador pode corrigir.

3.3 Segurança do esquema

Para códigos QC-MDPC, não são necessárias as matrizes embaralhadoras **S** e **P**. Isso pois, com conversões IND-CCA2, não há vazamentos sobre informações da mensagem mesmo usando **G** sistemática, e é supostamente difícil de se obter **H** a partir de **G**. Os parâmetros sugeridos para alguns níveis de segurança são descritos na Tabela 3.3.1.

Segurança	n_0	n	r	w	t	Tamanho da Chave em bits
80	2	9602	4801	90	84	4801
128	2	19714	9857	142	134	9857
256	2	65542	32771	274	264	32771

Tabela 3.3.1: *Parâmetros sugeridos para cada nível de segurança*

A redução de segurança do QC-MDPC McEliece se baseia na hipótese de que é difícil distinguir matrizes geradoras de códigos QC-MDPC de matrizes geradoras de códigos quase cíclicos aleatórios. Este é um problema similar ao de encontrar palavras de pesos baixos em códigos lineares. Sob essa hipótese razoável, prova-se que quebrar o esquema QC-MDPC não é mais fácil do que resolver o problema da decodificação por síndrome, que é conhecidamente $\mathcal{NP}$-difícil, e conjecturado tipicamente difícil. Esta redução permite que seja possível um algoritmo Porém, esta boa redução de segurança não leva em conta o fato de o algoritmo de decriptação falhar com certa probabilidade. Isto não seria um problema se o algoritmo falhasse com probabilidade independente da similaridade entre o erro usado na encriptação e a chave secreta. Infelizmente, este não é o caso, e o QC-MDPC pode ser atacado usando as reações do decriptador, isto é, se foi ou não possível decriptar a mensagem, para um grande número de mensagens encriptadas [GJS16].

Capítulo 4

Ataque de reação ao QC-MDPC McEliece

Na AsiaCrypt 2016, Guo, Johansson, e Stankovski [GJS16] mostraram um ataque de recuperação de chave contra o QC-MDPC McEliece. O ataque é inteiramente derivado da observação de que a probabilidade de haver um erro de decodificação de um vetor $\mathbf{c} = \mathbf{m} + \mathbf{e}$ é menor quando a primeira linha da matriz esparsa secreta $\mathbf{h}$, e o vetor de erro $\mathbf{e}$, compartilham algumas propriedades. Dessa forma, uma atacante Eva pode enviar desafios de decodificação para a portadora da chave secreta, de modo a capturar informações estruturais sobre $\mathbf{h}$. Depois, com estas informações, Eva pode reconstruir a chave privada $\mathbf{h}$.

4.1 Estimação do espectro da chave privada

O ataque é composto por duas fases. Na primeira, chamada fase de estimação do espectro, uma atacante envia desafios de decodificação visando obter informação sobre a chave secreta usada pelo decodificador. A obtenção da informação é feita através da reação do decodificador, isto é, se o decodificador teve sucesso ou não. Na segunda fase, chamada de reconstrução a partir do espectro, a atacante usa as informações obtidas na primeira fase para reconstruir a chave.

A partir de um número suficiente de reações do decodificador, é possível, para cada bloco cíclico da chave secreta, recuperar os conjuntos das distâncias circulares entre as posições não nulas de cada bloco. Uma distância circular é simplesmente a menor distância entre duas posições, supondo que o vetor é circular.

Definição 4.1.1 (Distância circular e par d-distante)**.** Seja $\mathbf{e}$ um vetor binário de comprimento r. A distância circular entre duas posições i e j de $\mathbf{e}$, denotada por $\text{dist}_r(i, j)$, é o único inteiro positivo d tal que $d \leq \lfloor r/2 \rfloor$ e $d \equiv j - i \mod r$ ou $d \equiv i - j \mod r$. Dizemos que o par de índices s_1 e s_2 é d-distante se sua distância cíclica é d.

Exemplo 4.1.2. Considere o vetor $\mathbf{e} = [0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0]$ de 11 posições. Então as duas posições de suas entradas não nulas são 2 e 11. As possíveis distâncias circulares entre estas entradas são os valores $10 - 2 = 8 \mod 11$ e $2 - 10 = -8 \equiv 3 \mod 11$. Como, dentre os valores possíveis, somente 3 é menor que ou igual a $\lfloor 11/2 \rfloor$, então $\text{dist}_{11}(2, 10) = 3$.

O conjunto de todas as distâncias circulares entre posições de entradas não nulas de um vetor é chamado de seu espectro. Por esta definição, pode-se ver que o espectro de um vetor é invariante em relação a rotações de um vetor. Portanto os espectros de todas as linha de uma matriz cíclica são iguais. O espectro de um vetor $\mathbf{v}$ é denotado por $\sigma(\mathbf{v})$ e sua definição formal é dada a seguir.

Definição 4.1.3 (Espectro)**.** Seja $\mathbf{v}$ um vetor binário de comprimento r. O espectro de v, denotado por $\sigma(\mathbf{v})$, é o conjunto

$$\sigma(\mathbf{v}) = \{d \in \mathbb{N} : \text{existe um par } d\text{-distante de entradas não nulas em } \mathbf{v}\}.$$

O número máximo de distâncias no espectro de um vetor de r posições é igual ao número de distâncias circulares possíveis, isto é $|\sigma(\mathbf{h}_0)| \leq \lfloor r/2 \rfloor$.

Considere que $\mathbf{c}$ é um texto que foi encriptado usando o erro $\mathbf{e} = [\mathbf{e}_0|\ldots|\mathbf{e}_{n_0-1}]$, em que $|\mathbf{e}_i| = r$, para $i = 0, 1, \ldots, n_0 - 1$. Então $\mathbf{c} = \mathbf{mG} + [\mathbf{e}_0|\ldots|\mathbf{e}_{n_0-1}]$. Guo et al. observaram que a probabilidade de o decodificador falhar ao tentar corrigir o erro $\mathbf{e}$ é mais baixa quanto maiores forem os conjuntos $\sigma(\mathbf{e}_i) \cap \sigma(\mathbf{h}_i)$. Isso acontece pois, quanto maiores as interseções, maior a chance de a decodificação ter sucesso na primeira iteração. Uma explicação de por que isso ocorre pode ser vista na Seção 4.2.

Na visão do atacante, o ideal seria que ele pudesse escolher com quais erros encriptar a mensagem de modo a testar suas hipóteses sobre quais distâncias estão dentro do espectro de $\mathbf{h}_0$. Por exemplo, para testar se uma distância d está no espectro, o atacante poderia enviar textos cifrados com erros em que grande parte dos pares de suas entradas não nulas esteja à distância circular de d posições. O problema é que, quando são usadas conversões IND-CCA2, o atacante não tem controle sobre o vetor de erro. A solução apresentada por Guo et al. é usar, para cada distância d, um valor $\mathbf{p}_0[d]$ que estima a sua taxa de falha. A taxa de falha de uma distância é a razão entre o número de vezes em que ela aparece num vetor de erros que não foi decodificado corretamente, e o número total de vezes em que ela aparece nos vetores de erros testados. Ao final do experimento, o vetor $\mathbf{p}_0[d]$ pode ser usado para distinguir distâncias dentro e fora do espectro, usando algoritmos de agrupamento (*clustering*). O Algoritmo 5 formaliza o procedimento para obter o vetor $\mathbf{p}_0$. É difícil estimar a priori qual o valor do número de testes de decodificação necessários M. Guo et al. usam simulações para isso, e observaram que, para o nível de segurança de 80 bits, $M = 200$ milhões parece ser, em média, suficiente para que se possa distinguir as distâncias dentro e fora do espectro de $\mathbf{h}_0$. A Seção 5 mostra uma análise mais detalhada desse algoritmo.

Algoritmo 5: Estimação das taxas de falha de decodificação para as possíveis distâncias em $\sigma(\mathbf{h}_0)$

Entrada: n, r, w, t parâmetros do código QC-MDPC atacado
$\mathcal{D}$ oráculo de reação para o decodificador (devolve 1 para sucesso, ou 0 para falha)
M número de testes de decodificação
Saída: $\mathbf{p}_0$ estimativas das taxas de falha para possíveis distâncias em $\sigma(\mathbf{h}_0)$

início
  $\mathbf{a}_0, \mathbf{b}_0 \leftarrow$ vetores com $\lfloor r/2 \rfloor$ entradas inicialmente todas iguais a 0
  para *cada teste de decodificação* $i = 1, 2, \ldots, M$ **faça**
    $\mathbf{c} \leftarrow$ um texto aleatório encriptado com erro $\mathbf{e} = [\,\mathbf{e}_0 \,|\, \mathbf{e}_x\,]$, em que $|\mathbf{e}_0| = r$
    $v = \mathcal{D}(\mathbf{c})$
    para *cada distância* d *em* $\sigma(\mathbf{e}_0)$ **faça**
      $\mathbf{a}_0[d] \leftarrow \mathbf{a}_0[d] + v$
      $\mathbf{b}_0[d] \leftarrow \mathbf{b}_0[d] + 1$
  $\mathbf{p}_0 \leftarrow$ vetor com $\lfloor r/2 \rfloor$ entradas iguais a 0
  para *cada distância* d *em* $\{1, 2, \ldots, \lfloor r/2 \rfloor\}$ **faça**
    $\mathbf{p}_0[d] \leftarrow \mathbf{a}_0[d]/\mathbf{b}_0[d]$
  devolva $\mathbf{p}_0$

4.2 Como o espectro da chave afeta a decodificação de certos erros

Algoritmos de *bit-flipping* tipicamente terminam em poucas iterações [MOG15], de forma que seu desempenho na primeira iteração é crítico para que consigam corrigir os erros. Nestes algoritmos, cada variável v_j tem um contador associado f_j. Este contador é incrementado quando v_j faz parte de uma equação de paridade não satisfeita. Considerando a primeira iteração do algoritmo na decodificação de um vetor $\mathbf{c} = \mathbf{mG} + \mathbf{e}$, a síndrome de $\mathbf{c}$, em relação à matriz de paridade secreta $\mathbf{H} = (h)_{ij}$, é

$$\mathbf{s} = \mathbf{cH}^T = \mathbf{eH}^T = [s_1, s_2, \ldots, s_r],$$

em que cada s_i é da forma

$$s_i = \sum_{j=1}^{n} h_{ij} e_j.$$

Se $s_i = 0$, a equação i foi satisfeita, e nenhum contador é incrementado. Caso contrário, se $s_i = 1$, são incrementados os contadores de cada variável que aparece na equação i. Um contador pode ser incrementado corretamente, isto é, quando $e_j = 1$, ou incorretamente, quando $e_j = 0$.

Podemos analisar a primeira iteração de decodificação um pouco mais detalhadamente ao considerar quantos contadores são incrementados corretamente. Chame de κ_i o número de posições de $\mathbf{e}$ tais que $e_j = 1$ e $h_{ij} = 1$, ou seja $\kappa_i = \#\{j : h_{ij} e_j = 1\}$. Intuitivamente, κ_i é o número máximo de erros detectáveis pela linha i da matriz $\mathbf{H}$. Note que $\kappa_i \equiv s_j \mod 2$, mas em geral $\kappa_i \neq s_j$. Então s_j só dá informação sobre a paridade de κ_i.

Se $\kappa_i = 0$ temos que a equação i não poderia identificar nenhum bit incorreto, e os contadores das w variáveis na equação j não são modificados. Por outro lado, se $\kappa_i = 1$, temos que a equação i captura apenas 1 bit incorreto, mas incrementa todos os contadores das suas w variáveis, sendo que apenas 1 dos contadores deveria ser incrementado.

Dizemos que ℓ contadores são *tratados corretamente* quando uma equação insatisfeita incrementa ℓ contadores de variáveis que estavam com erro $(e_j = 1)$, ou quando uma equação satisfeita deixa de incrementar ℓ contadores de variáveis que estavam sem erro $(e_j = 0)$. Analogamente, dizemos que que ℓ contadores são *tratados incorretamente* quando uma equação insatisfeita deixa de incrementar ℓ contadores de variáveis que não estavam com erro, ou quando uma equação satisfeita deixa de incrementar ℓ contadores de variáveis que estavam com erro. Pela definição, a soma do número de contadores tratados correta e incorretamente numa equação é sempre w. A Tabela 4.2.1 mostra a relação entre κ_i e o tratamento dos contadores.

κ_i	contadores tratados corretamente	contadores tratados incorretamente
0	w	0
1	1	$w-1$
2	$w-2$	2
3	3	$w-3$
⋮	⋮	⋮

Tabela 4.2.1: *Relação entre κ_i e o número de contadores tratados correta ou incorretamente*

Para o decodificador cometer menos erros na primeira iteração, é interessante que o número de contadores tratados corretamente seja alto. Ou seja, o decodificador deve ter menor taxa de falha quando, para grande parte das equações i, o valor de κ_i for um número par pequeno ou um ímpar grande.

A distinção entre distâncias dentro e fora do espectro ocorre pois, quando uma distância d pertence simultaneamente a $\sigma(\mathbf{h}_0)$ e a $\sigma(\mathbf{e}_0)$, é forçado, em ao menos uma equação, que 2 posições de uma mesma linha, digamos j_1 e j_2, sejam tais que $h_{ij_1} e_{j_1} = 1$ e $h_{ij_2} e_{j_2} = 1$. Isto faz com que a distribuição dos κ_i tenda a ficar mais concentrada em valores pares pequenos, o que diminui a probabilidade de o decodificador falhar. A Figura 4.2.1 ilustra o que ocorre quando sobrepõe-se o erro

$$\mathbf{e}_0 = [0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0],$$

à matriz cíclica cuja primeira linha é dada por

$$\mathbf{h}_0 = [0\,0\,0\,0\,1\,1\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1\,0].$$

Note que $\sigma(\mathbf{e}) = \{3\}$, e que $\mathbf{h}_0$ tem três pares de 1's à distância circular de 3 posições. Isso impõe pares de posições em que $h_{ij} e_j = 1$ em 3 linhas, a saber, as linhas 11, 16, e 20. Esta pequena alte-

ração na distribuição dos κ_i é suficiente para, com um grande número de testes de decodificação, distinguir distâncias dentro e fora do espectro de $\mathbf{h}_0$.

Note que uma discussão análoga pode ser feita para os vetores $\mathbf{h}_1, \ldots, \mathbf{h}_{n_0-1}$. Então o algoritmo para distinguir distâncias dentro e fora do espectro também serve para cada um desses vetores.

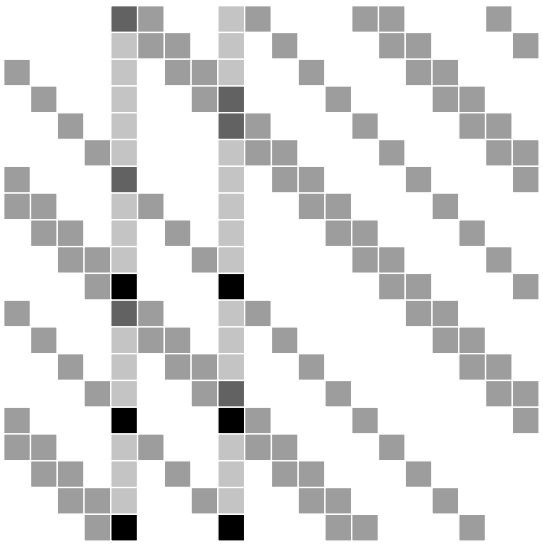

Figura 4.2.1: *Erro* **e** *sobreposto à matriz* **H**, *com destaque em preto para os pares de posições em que* $h_{ij}e_j = 1$, *impostos pela similaridade entre os espectros*

4.3 Reconstrução da chave a partir do espectro

Depois de identificado o espectro de $\mathbf{h}_0$ como $\sigma(\mathbf{h}_0) = \{s_1, s_2, \ldots, s_m\}$, pode-se usá-lo para reconstruir a chave. A proposta de Guo et al. é usar um algoritmo recursivo, que encontra a chave ao fazer uma busca profundidade numa árvore. Este procedimento é descrito pelo Algoritmo 6. O algoritmo encontra $\mathbf{h}_0$, a menos de uma rotação circular, o que não faz diferença, já que qualquer rotação de $\mathbf{h}_0$ gera uma matriz equivalente a **H**. Para construir **H** a partir de $\mathbf{h}_0$, primeiro note que, conhecendo a matriz pública **G** e $\mathbf{h}_0$, a atacante pode construir os vetores $\mathbf{h}_1, \ldots, \mathbf{h}_{n_0-1}$ através da equação 3.1.1, copiada a seguir

$$\mathbf{G} = \left[\begin{array}{c|c} & \left(\mathbf{H}_{n_0-1}^{-1} \cdot \mathbf{H}_0\right)^T \\ & \left(\mathbf{H}_{n_0-1}^{-1} \cdot \mathbf{H}_1\right)^T \\ \mathbf{I} & \vdots \\ & \left(\mathbf{H}_{n_0-1}^{-1} \cdot \mathbf{H}_{n_0-2}\right)^T \end{array}\right].$$

Uma vez que a atacante conhece cada um dos vetores $\mathbf{h}_i$, isto é, todas primeiras linhas dos blocos cíclicos de **H**, ela pode facilmente construir **H**. Também é fácil testar se a matriz **H** recuperada é de fato a matriz secreta. Basta verificar se **H** é uma matriz geradora do núcleo de **G**, que é a matriz geradora pública, ou seja, testar se $\mathbf{G}\mathbf{H}^T = \mathbf{0}$.

A ideia do algoritmo de reconstrução é tentar recursivamente colocar o máximo possível de 1's num vetor de tamanho r sem que ocorra uma distância fora do espectro. Quando é impossível colocar um 1 sem que apareça uma distância fora do espectro, o algoritmo retira o último 1 colocado, e retoma a partir da próxima posição possível para um 1.

Algoritmo 6: Algoritmo recursivo de reconstrução de chave [GJS16]

Entrada: n, r, w, t parâmetros do código QC-MDPC a ser atacado
$\sigma(\mathbf{h}_0)$ o espectro de $\mathbf{h}_0$
V o suporte parcial de uma rotação de $\mathbf{h}_0$ (inicialmente vale $\{1, 1+s_1\}$)
Saída: V uma rotação do vetor $\mathbf{h}_0$, ou $\perp$ se $\sigma(\mathbf{h}_0)$ for inválido

```
1  início
2      se |V| = ŵ então
3          se V é o suporte de uma rotação de h_0 então
4              devolva V
5          devolva ⊥
6      para cada posição possível j = 2, ..., n faça
7          se dist_r(v, j) ∈ σ(h_0) para todo v em V então
8              Adicione j a V
9              ret ← chamada recursiva usando o conjunto V atual
10             se ret ≠⊥ então
11                 v ← vetor com suporte V
12                 se v é uma rotação de h_0 então
13                     devolva v
14             Remova j de V
15     devolva ⊥
```

O principal argumento para justificar a eficiência do algoritmo é que ramos da árvore que não contêm a chave são podados pelo algoritmo num nível não muito baixo da árvore. Seja α a fração das $\lfloor r/2 \rfloor$ possíveis distâncias que pertencem a $\sigma(\mathbf{h}_0)$, isto é $\alpha = |\sigma(\mathbf{h}_0)|/\lfloor r/2 \rfloor$. Digamos que a busca começa no nível 0 da árvore, em que já foram colocadas as posições 1 e $1+s_1$ no suporte V. Para que uma posição p seja válida para entrar no nível 1, ambas as distâncias $\text{dist}_r(p, 1)$ e $\text{dist}_r(p, 1+s_1)$ devem pertencer a $\sigma(\mathbf{h}_0)$. Isso resulta em $r\alpha^2$ posições possíveis para o nível 1. Repetindo uma argumentação similar para níveis maiores, pode-se concluir que no nível ℓ são esperados em torno de $r\alpha^\ell$ nós. Assim, o número de nós em cada nível decresce exponencialmente.

Para estimar o número de caminhos possíveis, da raiz até uma folha[10] da árvore de busca, que o algoritmo pode explorar, Guo et al. supõem que as primeiras posições a serem acrescentadas, a menos a posição 1, são da forma $1+s$ para algum s no espectro. Isso não vale em geral, já que posições maiores do que $\lfloor r/2 \rfloor$ não estão no espectro. Mas com alta probabilidade isso vale para um pequeno número de posições. Nesse caso, o número de nós no nível ℓ da árvore é próximo de $r/2\alpha^{\ell+1}$. Chame de ϕ o menor valor de ℓ tal que $(r/2)\alpha^{\ell+2} < 1$, isto é, o nível $\phi+1$ é o primeiro nível a ter número esperado de nós menor do que 1. Então o número Γ de caminhos diferentes que devem podem ser explorados é em torno de

$$\Gamma \approx \prod_{l=1}^{\phi} |\sigma(\mathbf{h}_0)|\alpha^l = \lfloor r/2 \rfloor^\phi \alpha^{\phi(\phi+3)/2}. \tag{4.3.1}$$

O valor de ϕ, quando o espectro é totalmente conhecido, tipicamente é baixo, por exemplo $\phi = 6$ para o nível de segurança de 80 bits, de forma que é razoável esperar, como Guo et al. supõem, que as $2+\phi$ primeiras entradas estejam nas primeiras $\lfloor r/2 \rfloor$ entradas do vetor. Uma vez conhecidas estas $2+\phi$ posições de entradas não nulas, a posição das entradas não nulas restantes fica totalmente determinada com alta probabilidade. A Figura 4.3.1 ajuda a ilustrar, para o nível de segurança de 80 bits, como o fato de conhecer $\phi+2 = 8$ posições de entradas não nulas determina totalmente o restante das entradas não nulas, quando $\sigma(\mathbf{h}_0)$ é totalmente conhecido pelo atacante. Para gerar esta figura, foram consideradas as médias, entre 1000 vetores $\mathbf{h}_0$ testados, do

[10] Fixada a raiz de uma árvore com arestas direcionais, uma folha é um nó terminal, ou seja, um nó de que não sai nenhum arco.

número de posições nulas que não podem ser determinadas quando se conhecem as posições de i entradas não nulas, para $i = 1, 2, \ldots 45$. Note que, quando todas as entradas não nulas são determinadas, o que é representado pela ordenada 0 no gráfico, a posição de todas as entradas não nulas é automaticamente fixada, pois cada entrada é ou nula, ou não nula.

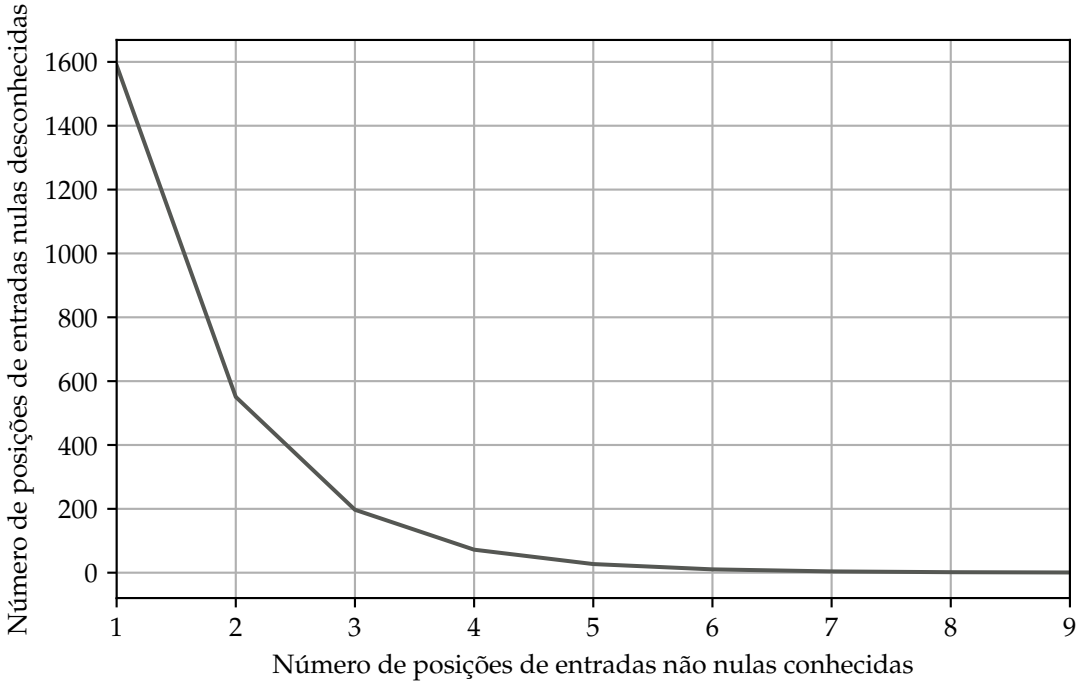

Figura 4.3.1: *Média do número de posições de entradas nulas desconhecidas em função do número de entradas não nulas conhecidas, tomando a média de 1000 vetores* $\mathbf{h}_0$ *gerados aleatoriamente com parâmetros para segurança de 80 bits*

Para comparar o desempenho do algoritmo original de reconstrução com os algoritmos de reconstrução propostos neste trabalho, é útil considerar uma cota inferior para Γ. Para isso, considere a seguinte manipulação da Equação 4.3.1

$$\Gamma = \lfloor r/2 \rfloor^{\phi} \alpha^{\phi^2/2} \alpha^{3\phi/2} = \left(\lfloor r/2 \rfloor \alpha^{\phi+1} \right)^{\phi/2} \left(\lfloor r/2 \rfloor \alpha^2 \right)^{\phi/2}.$$

Pela definição de ϕ, vale que $\lfloor r/2 \rfloor \alpha^{\phi+1} \geq 1$. Então

$$\Gamma \geq \left(\lfloor r/2 \rfloor \alpha^2 \right)^{\phi/2}.$$

Pode-se obter uma cota superior para Γ de maneira similar, considerando-se que

$$\Gamma = \left(\lfloor r/2 \rfloor \alpha^{\phi+2} \right)^{\phi/2} \left(\lfloor r/2 \rfloor \alpha \right)^{\phi/2}.$$

Mas agora lembre-se que, para $\phi + 2$, vale a inequação $\lfloor r/2 \rfloor \alpha^{\phi+2} < 1$. Então

$$\Gamma < \left(\lfloor r/2 \rfloor \alpha \right)^{\phi/2}.$$

Guo et al. não apresentaram uma análise detalhada do desempenho de seu algoritmo, dizendo apenas que sua implementação encontra a chave em alguns minutos. Entretanto, em sua apresentação na Asiacrypt [Guo16], Guo mostra que sua implementação demora em média 144 segudos para encontrar a chave, terminando em 49 minutes no pior caso, para o nível de segurança de 80 bits.

Para estimar a performance do algoritmo de Guo et al. para níveis de segurança mais altos,

consideramos uma extrapolação do tempo de execução de sua implementação da seguinte forma. Seja Γ_λ o número de caminhos possíveis que o algoritmo pode testar quando é recuperado o espectro completo de $\mathbf{h}_0$ de um código QC-MDPC que suporta a segurança de λ bits. Supomos que o número médio de caminhos que se deve testar até encontrar a chave é $\Gamma_\lambda/2$. Supomos também que o número médio de operações necessárias para testar cada caminho não depende do nível de segurança. Seja T_λ a quantidade média de trabalho, em alguma medida razoável, para se recuperar a chave através do algoritmo de Guo et al.. Podemos então escrever

$$T_\lambda = T_{80}\frac{\Gamma_\lambda}{\Gamma_{80}}.$$

.

A Tabela 4.3.1 mostra os resultados da extrapolação supondo que $T_{80} = 144$ segundos, que é o tempo médio obtido por Guo et al.. Pode-se ver que o desempenho do algoritmo não escala bem para níveis de segurança mais altos que 80. Os parâmetros α foram obtidos através de simulações. Pode-se ver que o número esperado de caminhos que o algoritmo original de reconstrução de chave deve visitar até encontrar a chave cresce rapidamente em função do nível de segurança.

O tempo esperado para se encontrar a chave pode ser diminuído através da paralelização da busca. Porém há limites para a paralelização da busca em profundidade [Rei85, GHR95], e não é claro quais estratégias de balanceamento de carga entre os *threads* oferecem melhores *speedups* [RK87, RS94]. Além disso, os expoentes crescentes dos valores de Γ para níveis de segurança mais altos fazem com que, mesmo com *speedups* lineares, a busca tome muito tempo.

Nível de segurança λ	α	ϕ	Γ	Extrapolação para o tempo médio de execução
80	0.3409	6	$2^{25.45}$	144s [Guo16]
128	0.3985	8	$2^{39.72}$	$\sim$ 32 dias
256	0.4351	10	$2^{61.95}$	$\sim$ 422930 anos

Tabela 4.3.1: *Desempenho estimado do algoritmo de reconstrução de chave*

Um outro problema do algoritmo, talvez tão crítico quanto o crescimento rápido de Γ em função de λ, é a impraticabilidade de seu uso quando o espectro não é totalmente conhecido. Quando o espectro é totalmente conhecido, os valores de α e ϕ são dados na Tabela 4.3.1. Quando há distâncias dentro do espectro incorretamente classificadas como fora do espectro, a linha 12 do Algoritmo 6 faz com que o algoritmo seja incapaz de encontrar $\mathbf{h}_0$. Mas, se algumas distâncias fora do espectro forem classificadas como dentro do espectro, o algoritmo ainda é capaz de encontrar $\mathbf{h}_0$, embora seu desempenho seja significativamente afetado. Um número maior de distâncias classificadas como dentro do espectro implica num maior valor de α. A Figura 4.3.2 mostra como o aumento no valor de α afeta a o número esperado de caminhos que o algoritmo deve explorar até que encontre $\mathbf{h}_0$.

Nas próximas seções, são mostrados algoritmos alternativos para a reconstrução de chave que, além de mais eficientes do que o original, conseguem trabalhar eficientemente com maiores valores de α.

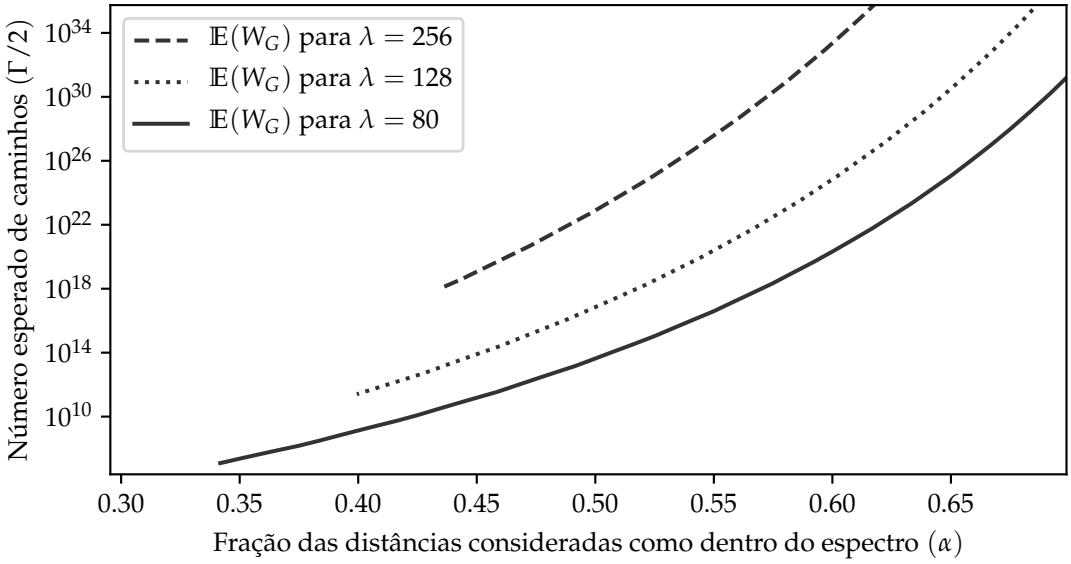

Figura 4.3.2: *Número esperado de caminhos até que se encontre a chave privada em relação ao número de distâncias consideradas como dentro do espectro para os níveis de segurança 80, 128, e 256*

Capítulo 5

Obtendo informação sobre os espectros dos blocos da chave privada

Nesta seção, o algoritmo de estimação do espectro de Guo et al. [GJS16] é analisado mais detalhadamente, e é introduzida uma métrica para a qualidade de sua saída, chamada d-exclusão. A d-exclusão representa o número de distâncias que podem consideradas como fora do espectro, dada a saída do algoritmo de estimação do espectro. Essa métrica é usada nas próximas seções para medir a eficiência dos algoritmos de reconstrução de chave. Em particular, queremos algoritmos que sejam eficientes mesmo com uma baixa d-exclusão obtida pelo algoritmo de estimação do espectro. Terminamos mostrando como o número de testes de decodificação afeta a d-exclusão através de simulações.

5.1 O algoritmo de recuperação do espectro

Começamos mostrando que o algoritmo de recuperação do espectro pode ser adaptado para que seja extraída mais informação de cada erro de decodificação. A partir de agora, consideram-se apenas códigos QC-MDPC cuja matriz de paridade esparsa tem dois blocos cíclicos, isto é, $n_0 = 2$. O algoritmo apresentado por Guo et al. recupera apenas o espectro de $\mathbf{h}_0$ simplesmente porque seu algoritmo de reconstrução usa informação somente sobre $\mathbf{h}_0$. Porém com uma pequena adaptação, o algoritmo é capaz de recuperar simultaneamente informação sobre $\mathbf{h}_0$ e $\mathbf{h}_1$. A descrição do algoritmo com essa adaptação é dada no Algoritmo 7.

A Figura 5.1.1 mostra o resultado do algoritmo contra um código (9602,4801,90)-QC-MDPC, que eram, até o ataque de reação de Guo et al. [GJS16], parâmetros recomendados para um nível de segurança de 80 bits. Para o gráfico, usamos as taxas de falha para possíveis distâncias em $\mathbf{h}_0$, isto é, o vetor $\mathbf{p}_0$ devolvido pelo algoritmo de estimação do espectro. Na figura, fica evidente a forte correlação entre as probabilidades de falha. Mas, mesmo com $M = 100$ milhões de testes de decodificação, não há uma separação completa entre as distâncias dentro e fora do espectro para taxas de falha próximas de 0.00058.

Algoritmo 7: Estimação das taxas de falha de decodificação para as possíveis distâncias em $\sigma(\mathbf{h}_0)$ e $\sigma(\mathbf{h}_1)$

Entrada: n, r, w, t parâmetros do código QC-MDPC atacado
$\mathcal{D}$ oráculo de reação para o decodificador (devolve 1 para sucesso, ou 0 para falha)
M número de testes de decodificação
Saída: $\mathbf{p}_0, \mathbf{p}_1$ estimativas das taxas de falha para possíveis distâncias em $\sigma(\mathbf{h}_0)$ e $\sigma(\mathbf{h}_1)$

início
  $\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1 \leftarrow$ vetores com $\lfloor r/2 \rfloor$ entradas inicialmente todas iguais a 0
  para *cada teste de decodificação* $i = 1, 2, \ldots, M$ **faça**
    $\mathbf{c} \leftarrow$ um texto aleatório encriptado com erro $\mathbf{e} = [\, \mathbf{e}_0 \,|\, \mathbf{e}_1 \,]$
    $v = \mathcal{D}(\mathbf{c})$
    para *cada distância* d *em* $\sigma(\mathbf{e}_0)$ **faça**
      $\mathbf{a}_0[d] \leftarrow \mathbf{a}_0[d] + v$
      $\mathbf{b}_0[d] \leftarrow \mathbf{b}_0[d] + 1$
    para *cada distância* d *em* $\sigma(\mathbf{e}_1)$ **faça**
      $\mathbf{a}_1[d] \leftarrow \mathbf{a}_1[d] + v$
      $\mathbf{b}_1[d] \leftarrow \mathbf{b}_1[d] + 1$
  $\mathbf{p}_0, \mathbf{p}_1 \leftarrow$ vetor com $\lfloor r/2 \rfloor$ entradas iguais a 0
  para *cada distância* d *em* $\{1, 2, \ldots, \lfloor r/2 \rfloor\}$ **faça**
    $\mathbf{p}_0[d] \leftarrow \mathbf{a}_0[d] / \mathbf{b}_0[d]$
    $\mathbf{p}_1[d] \leftarrow \mathbf{a}_1[d] / \mathbf{b}_1[d]$
  devolva $\mathbf{p}_0$ *e* $\mathbf{p}_1$

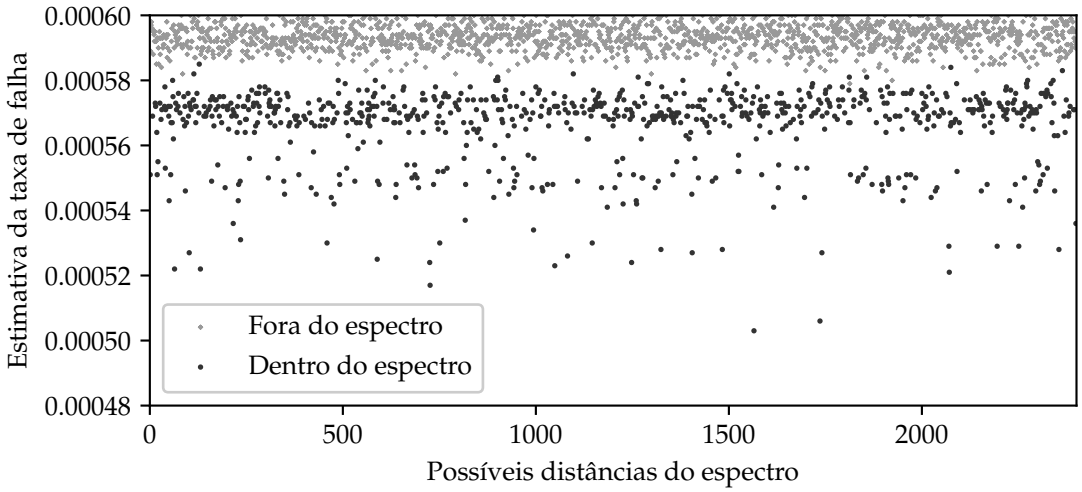

Figura 5.1.1: *Estimativa das probabilidades de falha* $\mathbf{p}_0$*, usando* $M = 100$ *milhões de testes de decodificação, para um código (9602,4801,90)-QC-MDPC gerado aleatoriamente*

5.2 Uma métrica para a qualidade da recuperação

O algoritmo de busca em profundidade para a reconstrução da chave apresentado por Guo et al. não é capaz de recuperar a chave se uma ou mais distâncias dentro do espectro forem classificadas incorretamente como fora do espectro. Porém ele consegue lidar com um pequeno número de distâncias fora do espectro classificadas como estando dentro do espectro. Esse fato nos motivou a definir a qualidade de um vetor de probabilidades recuperado como sendo o número máximo de distâncias cuja taxa de falha é maior do que a maior das taxas de falha das distâncias dentro do espectro. Chamamos essa métrica de d-exclusão pois ela conta o número máximo de distâncias que podem ser consideradas fora do espectro ao se considerar apenas um ponto de separação

entre distâncias dentro e fora do espectro.

Definição 5.2.1. Dizemos que a saída $\mathbf{p}_i$ do algoritmo de recuperação do espectro obtém uma d-exclusão para o espectro do bloco $\mathbf{h}_i$ se existe um número $p \in (0,1)$ tal que

1. Se $\mathbf{p}_i[d] \geq p$ então $d \notin \sigma(\mathbf{h}_i)$;
2. $\#\{\mathbf{p}_i[d] \geq p\} = d$.

Quando $d = \lfloor r/2 \rfloor - |\sigma(\mathbf{h}_i)|$, ou seja, d é exatamente o número de distâncias fora do espectro de $\mathbf{h}_i$, dizemos que $\mathbf{p}_i$ obtém d-exclusão completa.

A Figura 5.2.1 pode ajudar a exemplificar o conceito. Aqui foram feitos apenas $M = 30$ milhões de testes de decodificação para um código (9602,4801,90)-QC-MDPC aleatório e diferente do usado para a Figura 5.1.1. Podemos ver que há uma mistura bem maior entre distâncias dentro e fora do espectro. A d-exclusão obtida é de 983 distâncias, sendo que o total de distâncias fora do espectro de $\mathbf{h}_0$ é 1587.

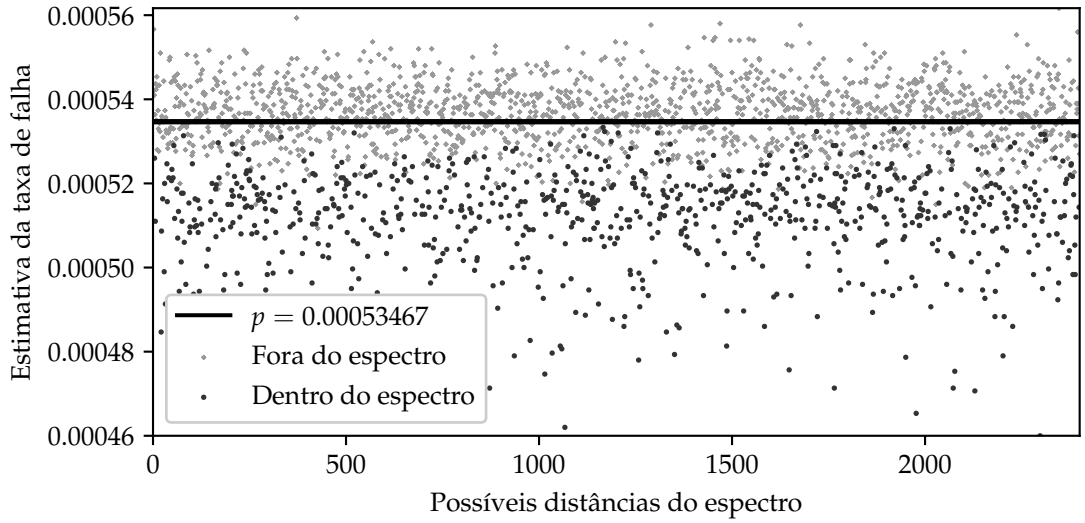

Figura 5.2.1: *Estimativa das probabilidades de falha* $\mathbf{p}_0$ *para um código (9602,4801,90)-QC-MDPC gerado aleatoriamente usando* $M = 100$ *milhões de testes de decodificação*

A eficiência de cada um dos algoritmos de reconstrução apresentados neste trabalho será analisada em função da d-exclusão obtida pelo algoritmo de recuperação do espectro. Com essa métrica, podemos especificar que o objetivo deste trabalho é encontrar algoritmos de reconstrução mais eficientes, mas que consigam tratar de menores d-exclusões.

Quanto menor a d-exclusão que um algoritmo de reconstrução precisa para funcionar eficientemente, também menor é o número de decodificações necessárias para quebrar uma chave. Estimar este número é importante para determinar qual deve ser a vida útil de chaves QC-MDPC para que estas sejam usadas de forma segura. Infelizmente não é fácil encontrar analiticamente o valor esperado dos tamanhos das d-exclusões em relação ao número M de decodificações. Isso pois essa relação é dependente da taxa de falha, que é específica de cada código, e não se conhecem meios analíticos de encontrá-la. Uma expressão para a probabilidade de a decodificação falhar para códigos QC-MDPC seria um bom primeiro passo para desenvolver decodificadores que, quando falham, não dão informação sobre o espectro. Esta é uma área de pesquisa ativa em Teoria de Códigos, e este é um problema difícil [RL09]. Portanto fizemos simulações para entender a relação entre M e as d-exclusões.

5.3 Como o número de decodificações afeta a d-exclusão

Para analisar a relação entre o número de decodificações e a d-exclusão obtida, implementamos em C o esquema QC-MDPC McEliece e foi feita uma simulação com 200 códigos QC-MDPC para o nível de segurança de 80 bits. Para cada código, foram considerados 100 milhões de testes de decodificação usando o algoritmo de *bit-flipping* de Gallager [Gal62]. Sua descrição pode ser vista no Algoritmo 8, e também pode ser visto como o algoritmo $\mathcal{B}$ considerado por Maurich et al. [MOG15]. Note que este algoritmo é muito similar aos outros decodificadores apresentados neste trabalho, com a diferença de que um bit é invertido se o número de equações de paridade insatisfeitas a que ele pertence for maior do que certos valores previamente calculados, chamados *thresholds* de Gallager, que só dependem dos parâmetros n, k, e $\hat{w}$ do código, e variam de acordo com o número da iteração em questão. Gallager discute em seu trabalho como calcular os valores dos *thresholds* [Gal62]. Escolhemos esse algoritmo para facilitar a comparação com o trabalho de Guo et al. [GJS16], já que eles usaram esse mesmo decodificador para seus testes. Todo o código-fonte e os dados gerados podem ser encontrados em www.ime.usp.br/~tpaiva/msc.

Algoritmo 8: Algoritmo de *bit-flipping* de Gallager [Gal62] usado para os testes de decodificação

Entrada: **H** matriz de paridade esparsa $r \times n$ de um código LDPC
y uma palavra transmitida a ser decodificada
`max_it` o número máximo de iterações
Saída: A decodificação estimada de **y**, ou $\perp$ se o número máximo de iterações for excedido

```
1  início
2      it ← 0
3      enquanto yH^T ≠ 0 e it < max_it faça
4          para cada j = 1, 2, ..., n faça
5              f_j ← Número de vizinhos de v_j insatisfeitos
6          para cada j = 1, 2, ..., n faça
7              se f_j ≥ GallagerThresholds[it] então
8                  y_j ← ȳ_j
9          it ← it + 1
10     se it ≥ max_it então
11         devolva ⊥
12     senão
13         devolva y
```

Cada teste de decodificação computa, para cada $M = 1, 2, \ldots, 100$ milhões, as d-exclusões obtidas para as taxas de falha para possíveis distâncias em $\sigma(\mathbf{h}_0)$ e em $\sigma(\mathbf{h}_1)$. Um exemplo do resultado de um teste de decodificação pode ser visto na Figura 5.3.1.

Tanto pela natureza aleatória dos testes de decodificação e até da geração dos códigos QC-MDPC, pode haver muita variabilidade nas d-exclusões obtidas. Para podermos analisar os dados para os 200 códigos testados, a Figura 5.3.2 mostra os *boxplots* separados por intervalos de 10 milhões de decodificações. Uma das observações que motivaram este trabalho é que a d-exclusão cresce rápido no intervalo entre 0 e 40 milhões de decodificações, e passa a crescer cada vez mais lentamente. Assim, desenhando algoritmos que tratem eficientemente de d-exclusões entre 750 e 1000, para 80 bits de segurança, o número de testes de decodificação de pouco mais de 200 milhões, como recomendado por Guo et al., pode ser diminuído para aproximadamente 30 milhões.

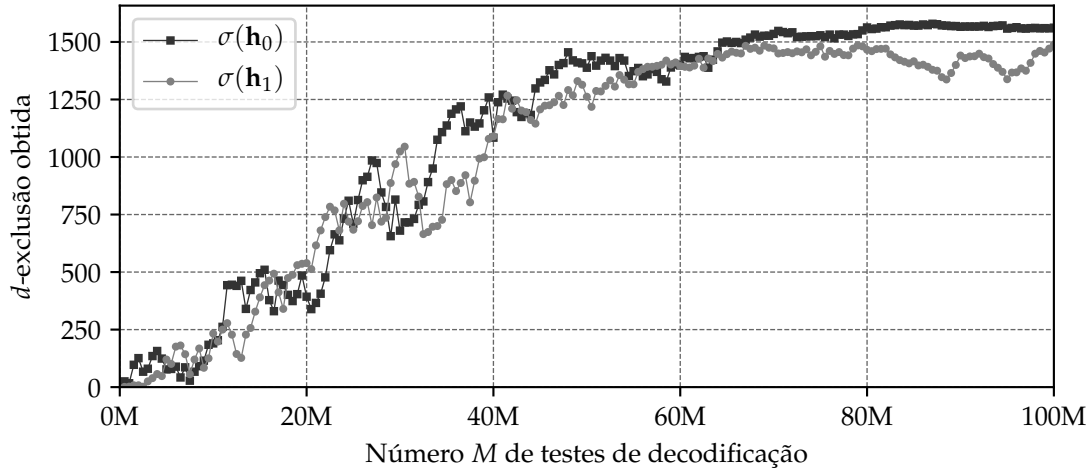

Figura 5.3.1: *Relação entre as d-exclusões obtidas e número de testes de decodificação, para um dos 200 códigos testados de testes de decodificação*

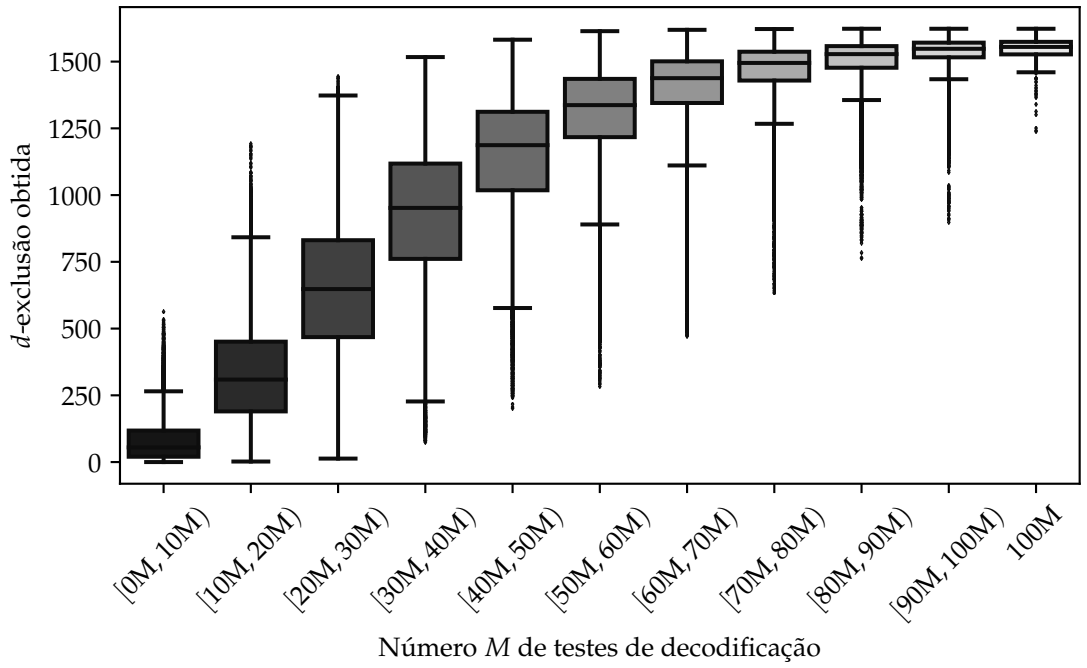

Figura 5.3.2: *Relação entre as d-exclusões obtidas e o número de testes de decodificação, considerando os resultados para 200 códigos diferentes com parâmetros de segurança de 80 bits*

5.4 Obtendo distâncias dentro e fora do espectro

Para que os algoritmos de reconstrução sejam eficazes, é importante que lhes sejam dados como entrada um conjunto, em geral grande (e.g. $\sim r/6$ entradas), de distâncias fora do espectro, e um outro conjunto, em geral pequeno (e.g. 1 entrada), de distâncias dentro do espectro.

Obter uma distância dentro do espectro é relativamente fácil. Pegue a saída $\mathbf{p}_0$ do algoritmo de estimação de espectro, e tome a distância d que tenha a menor taxa de falha em $\mathbf{p}_0$, isto é, $\mathbf{p}_0[d]$ é mínimo. Tipicamente, após um pequeno número de testes de decodificação a entrada com menor taxa de falha de fato está dentro do espectro. Pelo resto do texto, supomos que é fácil obter ao menos uma distância dentro do espectro de $\mathbf{h}_0$, e outra dentro do espectro de $\mathbf{h}_1$, denotadas por s_0 e s_1, respectivamente.

Obter conjuntos grandes de distâncias fora do espectro é mais difícil. Idealmente, se fosse conhecido que q distâncias podem ser d-excluídas a partir de $\mathbf{p}_0$, bastaria tomar as q distâncias com maiores taxas de falha em $\mathbf{p}_0$. O problema é que em geral não se sabe qual foi a d-exclusão obtida pela saída do algoritmo de estimação de espectro. Um modo conservador de lidar com isso é descobrir qual a mínima d-exclusão necessária para que o algoritmo de reconstrução escolhido trabalhe eficientemente. Uma vez determinado que esta mínima d-exclusão é q, faça o seguinte.

1. Faça $i \leftarrow 1$.
2. Faça M_i testes de decodificação e atualize os vetores $\mathbf{p}_0$ e $\mathbf{p}_1$.
3. Construa os conjuntos D_0 e D_1 com as q entradas cujas taxas de falha são máximas em $\mathbf{p}_0$ e $\mathbf{p}_1$, respectivamente.
4. Execute o algoritmo de reconstrução de chave usando os conjunto D_0 e D_1 como o conjunto das distâncias d-excluídas. Se o algoritmo termina sem encontrar a chave, D_0 ou D_1 devem conter alguma distância dentro do espectro, então faça $i \leftarrow i + 1$ e volte para o passo 2.

O passo 4 pode variar de acordo com o algoritmo de reconstrução usado. Por exemplo, o algoritmo de reconstrução de Guo et al. pode ser executado tanto com D_0 quanto com D_1. Se ambas as execuções falharem em encontrar a chave, então isso implica que ambos os conjuntos D_0 e D_1 contêm entradas dentro dos espectros correspondentes de $\mathbf{h}_0$ e $\mathbf{h}_1$. Em contraste com esse algoritmo, o algoritmo proposto na Seção 7 deve ser executado com ambos os conjuntos D_0 e D_1. Se o algoritmo falhar, então ao menos um dos conjuntos contém entradas dentro de seu espectro correspondente.

O número de decodificações a cada fase M_i deve ser modulado de acordo com os tempos de execução dos algoritmos. Em geral, é interessante que no começo sejam feitos muitos testes de decodificação, pois há baixa probabilidade de se obter uma boa d-exclusão com poucos testes, e depois sejam feitos cada vez menos testes de decodificação a cada execução do algoritmo de reconstrução. Por exemplo, para o algoritmo de reconstrução de Guo et al., considerando-se o nível de segurança de 80 bits, uma parametrização razoável seria $M_0 = 200$ milhões, e $M_i = 20$ milhões para todo $i \geq 1$.

Capítulo 6

Variante probabilística do algoritmo de reconstrução

O algoritmo de reconstrução de chave apresentado por Guo et al. [GJS16] a partir do espectro pode ser muito ineficiente para níveis de segurança mais altos que 80. Isso ocorre pois o número de estados possíveis é bem maior, podendo o algoritmo ficar preso por muito tempo num ramo próximo à raiz que não é parte da chave. A proposta apresentada neste capítulo é uma variante do algoritmo original de reconstrução em que, para cada nível da árvore de busca, um nó filho é escolhido aleatoriamente. Usando o fato de que bastam poucos níveis da árvore de busca corretamente percorridos para que a chave seja determinada, é possível mostrar que o número esperado de operações usando este algoritmo probabilístico é significativamente menos do que o original. Como exemplo, para os parâmetros de segurança de 80 bits, e considerando que o algoritmo tem o espectro completo de $\mathbf{h}_0$, o algoritmo termina em média após apenas 100 caminhos testados. Nos testes usando um chip Intel i7 870, usando apenas 1 *thread*, a chave é tipicamente encontrada em menos de 1 segundo, em contraste com a média de 144 segundos do trabalho de Guo et al.. Para níveis de segurança mais altos, as diferenças entre os tempos são ainda maiores. Além disso, é trivialmente paralelizável, e é capaz de encontrar eficientemente a chave com uma d-exclusão para $\mathbf{p}_0$ menor do que o algoritmo original.

6.1 Descrição do algoritmo

O algoritmo é uma extensão probabilística muito simples do algoritmo de reconstrução de Guo et al. [GJS16]. Ao invés de escolher deterministicamente o passo seguinte numa busca em largura, a proposta é percorrer a árvore de busca escolhendo aleatoriamente o próximo nó filho a ser percorrido. A descrição formal é dada no Algoritmo 9. Note que a descrição é dada em função de $\mathbf{h}_0$ por simplicidade, mas ele pode ser usado diretamente usando os parâmetros correspondentes para $\mathbf{h}_1$, isto é, s_1 e D_1.

A seguir é dada uma breve descrição do algoritmo. Os parâmetros s_0, que é uma distância dentro do espectro, e D_0, que é um conjunto de distâncias fora do espectro, são calculados como descrito na Seção 5.4. A cada iteração, o algoritmo começa com o conjunto $V = \{1, s_0 + 1\}$ e tenta completá-lo com mais $\hat{w} - 2$ posições, sem que ocorra nenhuma distância fora do espectro, isto é, distâncias em D_0 são proibidas. Para completar o vetor V, o algoritmo escolhe aleatoriamente uma distância do conjunto auxiliar F_l, que contém, para cada nível l, as posições possíveis para completar suporte. Ou seja, F_l contém os elementos de $[r]$ que não pertencem a V, e cuja distância até qualquer posição em V não está em D_0.

Nas próximas seções, é explicado por que essa variante probabilística tem um desempenho melhor do que a busca em profundidade sugerida por Guo et al.. Para isso é feita uma análise probabilística do algoritmo, e depois é mostrado o seu desempenho na prática.

Algoritmo 9: RECONSTRUÇÃOPROBABILÍSTICA : variante probabilística do algoritmo de reconstrução de chave

Entrada: n, r, w, t parâmetros do código QC-MDPC a ser atacado
$\hat{w}$ o peso do vetor $\mathbf{h}_0$
s_0 uma distância pertencente a $\sigma(\mathbf{h}_0)$
D_0 um conjunto de distâncias que não pertencem a $\sigma(\mathbf{h}_0)$
Saída: V o suporte de uma rotação do vetor $\mathbf{h}_0$

início
  faça
    $V \leftarrow \{1, 1 + s_0\}$
    $F_2 \leftarrow \{i \in [\lfloor r/2 \rfloor] - V : \text{dist}_r(i, v) \notin D_0 \, \forall v \in V\}$
    $l \leftarrow 2$
    enquanto $|V| < \hat{w}$ *e* $|F_l| > 0$ **faça**
      $p \leftarrow$ uma entrada escolhida aleatoriamente de F_l
      $V \leftarrow V \cup \{p\}$
      $F_l \leftarrow F_l - \{p\}$
      $F_{l+1} \leftarrow \{i \in F_l : \text{dist}_r(i, v) \notin D_0 \, \forall v \in V\}$
      $l \leftarrow l + 1$
  enquanto V *não for o suporte de uma rotação de* $\mathbf{h}_0$;
  devolva V

6.2 Análise probabilística

Em cada iteração, o algoritmo percorre a árvore de busca, da raiz até alguma de suas folhas, escolhendo aleatoriamente por qual filho irá seguir no próximo nível. Então a probabilidade de o algoritmo de construção encontrar $\mathbf{h}_0$ é exatamente a probabilidade de que cada filho escolhido seja um elemento do suporte de $\mathbf{h}_0$.

Seja s_0 uma distância pertencente ao espectro de $\mathbf{h}_0$. Suponha que estamos no nível l da árvore de busca, tendo até agora seguido o caminho $V = \{v_1 = 1, v_2 = 1 + s_0, \ldots v_l\}$, formado apenas por elementos do suporte de uma mesma rotação de $\mathbf{h}_0$. Seja F_l o conjunto dos filhos válidos no nível l, então

$$F_l = \{p \in [r] - V : \text{dist}_r(p, v) \notin D_0 \text{ para todo } v \in V\}.$$

Como o peso de $\mathbf{h}_0$ é $\hat{w}$, temos ao menos $\hat{w} - |V|$ escolhas de interesse para o próximo filho. Assim, a probabilidade de escolher um filho v_{l+1} dentro do suporte de $\mathbf{h}_0$ quando estamos no nível l, dado que todos os elementos que percorremos pertencem ao suporte de $\mathbf{h}_0$, é

$$\Pr\left(v_{l+1} \in \text{sup}(\mathbf{h}_0) \mid V \subset \text{sup}(\mathbf{h}_0)\right) = \frac{\hat{w} - |V|}{|F_l|} = \frac{\hat{w} - l}{|F_l|}.$$

Podemos então escrever a probabilidade de o algoritmo ter sucesso como

$$\begin{aligned}\Pr(V = \text{sup}(\mathbf{h}_0)) &= \Pr(v_{\hat{w}} \in \text{sup}(\mathbf{h}_0) \mid V - \{v_{\hat{w}}\} \subset \text{sup}(\mathbf{h}_0)) \\ &= \prod_{i=2}^{\hat{w}-1} \frac{\hat{w} - l}{|F_l|}.\end{aligned}$$

Lembre que o limite inferior do produto é $i = 2$ pois, como o valor inicial de V é $\{1, 1 + s_0\}$, a busca começa no segundo nível da árvore. O limite superior do produto é $\hat{w} - 1$ pois é neste nível em que ocorre a última decisão de por qual filho seguir. Podemos simplificar um pouco mais esse produto, considerando $\tau + 1$ como o primeiro valor de l para que $|F_l| = \hat{w} - l$. A partir deste nível $\tau + 1$, os elementos em F_τ são justamente os elementos que faltam para se chegar ao suporte de

$\mathbf{h}_0$. Com isso obtemos

$$\Pr(V = \sup(\mathbf{h}_0)) = \prod_{i=2}^{\tau} \frac{\hat{w} - l}{|F_l|}.$$

A distribuição da variável aleatória $|F_l|$ não é fácil de ser computada pois as distâncias em D_0 são dependentes dos elementos em V. Mas podemos aproximar o valor de sua esperança com um argumento similar ao usado por Guo et al.. Seja α a probabilidade de que uma distância não seja uma distância em D_0[11], então $\alpha = 1 - |D_0|/\lfloor r/2 \rfloor$. Suponha novamente que V só contenha elementos do suporte de $\mathbf{h}_0$. No nível l, há $\hat{w} - l$ filhos que são posições que pertencem ao suporte de $\mathbf{h}_0$, que são justamente as posições que faltam para completar V. Para as $(r - \hat{w})$ posições que não pertencem ao suporte, esperamos que α^l delas tenham sobrevivido aos crivos das posições possíveis feitos em cada nível. Assim, temos

$$\mathbb{E}\,|F_l| \approx (r - \hat{w})\alpha^l + \hat{w} - l.$$

Com isso, podemos aproximar a probabilidade de uma certa iteração do algoritmo encontrar a chave como

$$\Pr(V = \sup(\mathbf{h}_0)) \approx \prod_{l=2}^{\hat{w}-1} \frac{\hat{w} - l}{(r - \hat{w})\alpha^l + \hat{w} - l}.$$

Seja W a variável aleatória que conta o número de caminhos tetados até que a chave seja encontrada. Então a esperança de W é

$$\mathbb{E}(W) = \frac{1}{\Pr(V = \sup(\mathbf{h}_0))} = \prod_{l=2}^{\hat{w}-1} \frac{(r - \hat{w})\alpha^l + \hat{w} - l}{\hat{w} - l} = \prod_{l=2}^{\hat{w}-1} \left(\frac{(r - \hat{w})\alpha^l}{\hat{w} - l} + 1 \right). \tag{6.2.1}$$

Não é fácil limitar superiormente $\mathbb{E}(W)$ em função de de α, r, e $\hat{w}$. O problema parece similar ao de encontrar uma cota superior para um símbolo q de Pochammer, para o qual não parece haver resultado que podemos usar [Wei08]. Para limitar W por meio de uma expressão ligeiramente mais simples, considere τ como sendo o menor inteiro maior que 1 tal que

$$\prod_{l=\tau}^{\hat{w}-1} \left(\frac{(r - \hat{w})\alpha^l}{\hat{w} - l} + 1 \right) < r.$$

Então podemos escrever

$$\mathbb{E}(W) < r \prod_{l=2}^{\tau-1} \left(\frac{(r - \hat{w})\alpha^l}{\hat{w} - l} + 1 \right) < r \prod_{l=2}^{\tau-1} \left(\frac{(r - \hat{w})\alpha^l}{\hat{w} - \tau} + 1 \right).$$

Mas, como $0 \leq \alpha \leq 1$, note que

$$\frac{(r - \hat{w})\alpha^l}{\hat{w} - \tau} + 1 = \frac{(r - \hat{w})\alpha^l + \hat{w} - \tau}{\hat{w} - \tau} < \frac{r - \hat{w} + \hat{w} - \tau}{\hat{w} - \tau} = \frac{r - \tau}{\hat{w} - \tau} < \frac{r}{\hat{w} - \tau}.$$

Portanto o valor esperado de W pode ser limitado superiormente por

$$\mathbb{E}(W) < r \prod_{l=2}^{\tau-1} \frac{r}{\hat{w} - \tau} = r \left(\frac{r}{\hat{w} - \tau} \right)^{\tau-2} < \hat{w} \left(\frac{r}{\hat{w} - \tau} \right)^{\tau-1}.$$

Compare este valor com o valor esperado de caminhos que o algoritmo de reconstrução de Guo

[11] A razão α nesta seção é exatamente a mesma da Seção 4.3, com a diferença de que agora é definida a partir da d-exclusão.

et al. [GJS16] faz até recuperar a chave, denotado por W_G. Da Seção 4.3, vale que

$$W_G = \Gamma/2 \geq \frac{1}{2}\left(\lfloor r/2 \rfloor \alpha^2\right)^{\phi/2},$$

onde ϕ é o menor inteiro positivo tal que $(r/2)\alpha^{\phi+2} < 1$. Pode-se perceber que os expoentes $\tau - 1$ e $\phi/2$ são os parâmetros mais críticos para se determinar $\mathbb{E}(W)$ e $\mathbb{E}(W_G)$, mas infelizmente não é óbvia a relação entre ϕ e τ.

Para comparar os expoentes $\tau - 1$ e $\phi/2$, foram considerados os seus valores para os níveis de segurança 80, 128, e 256, e diferentes valores válidos de α. Os resultados podem ser vistos na Figura 6.2.1. Note que, para valores de α não muito altos, $\tau - 1 < \phi/2$, o que sugere que o algoritmo probabilístico tem melhor desempenho do que o algoritmo original. Na seção seguinte, o desempenho real do algoritmo probabilístico é avaliado, e comparado com o desempenho previsto pela Equação 6.2.1.

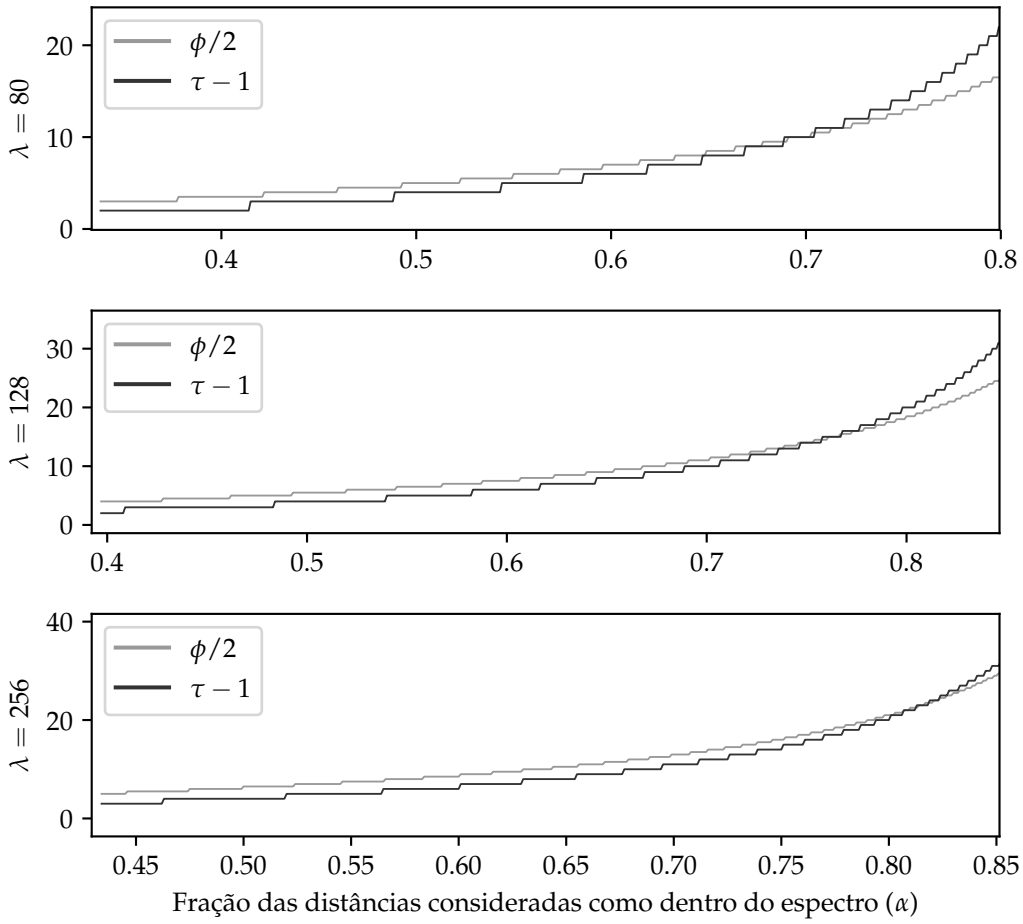

Figura 6.2.1: *Comparação entre os valores de $\tau - 1$ e $\phi/2$ para os níveis de segurança 80, 128, e 256, e valores válidos de α*

6.3 Resultados experimentais

Para testar a eficiência do algoritmo probabilístico, o algoritmo foi implementado em C, e seu desempenho foi avaliado para níveis de segurança 80, 128, e 256. A Figura 6.3.1 mostra os resultados dos números de caminhos testados até a chave ser encontrada e também as estimativas para estes números segundo a Equação 6.2.1. Foram considerados diferentes valores de $|D_0|$, isto é, do

número de distâncias d-excluídas. Para poder comparar as curvas, foi considerada a d-exclusão normalizada, isto é, o tamanho de $|D_0|$ dividido pelo número médio de distâncias fora do espectro. O eixo das ordenadas está em escala logarítmica, o que evidencia o crescimento superexponencial do número de caminhos explorados em relação a menores d-exclusões. Conforme o nível de segurança aumenta, a eficiência do algoritmo é significativamente reduzida para d-exclusões menores. Parece que as estimativas para $\mathbb{E}(W)$ se adequam bem aos valores observados.

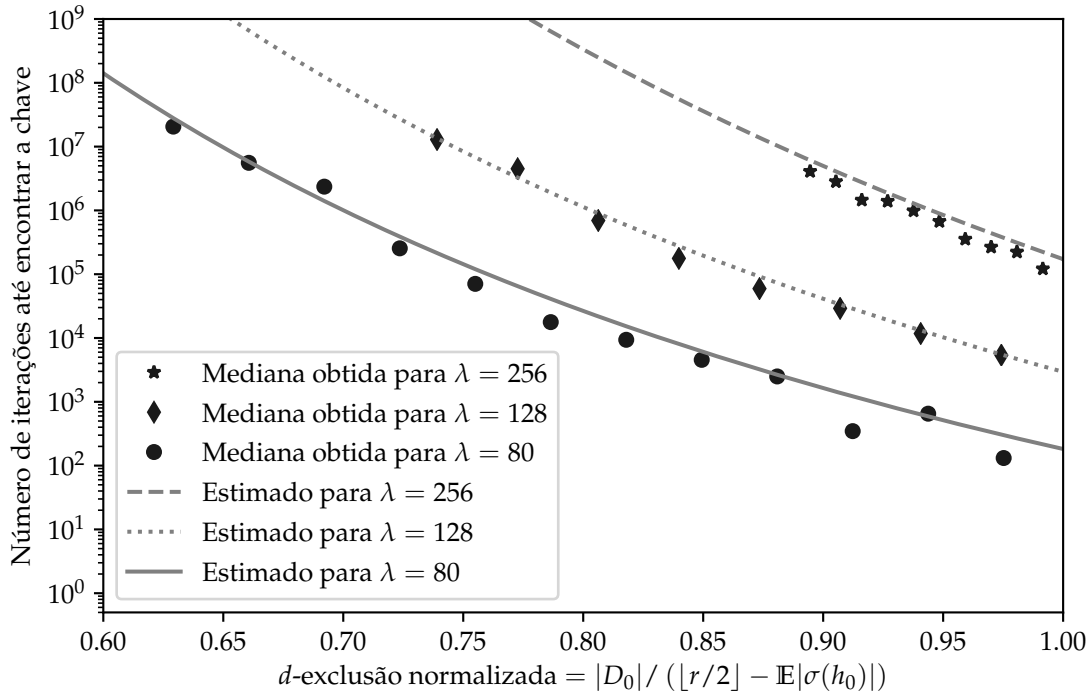

Figura 6.3.1: *Número de caminhos até encontrar a chave em função da proporção das entradas fora do espectro que são conhecidas*

Para dar um exemplo concreto do tempo de execução, o algoritmo foi executado numa máquina com um processador Intel Xeon E7-2870 a uma frequência de 2.40GHz, usando os seus 16 cores. Para cada nível de segurança, foram feitos 50 testes para cada valor de $|D_0|$, que é o conjunto das distâncias d-excluídas. Para cada teste, as distâncias d-excluídas foram escolhidas aleatoriamente. A Figura 6.3.3 mostra a mediana do tempo de execução do algoritmo. Note que o algoritmo termina muito rapidamente quando há informação completa sobre o espectro[12]. Para o nível de segurança de 80 bits, pode-se ver que o algoritmo é eficiente mesmo quando apenas 63% das distâncias fora do espectro são conhecidas.

Para comparar os desempenhos do algoritmo probabilístico com o do algoritmo original de Guo et al., considere as curvas de W_{PROB} e W_{G}, que representam os números esperados de caminhos que o algoritmo probabilístico e o algoritmo de Guo et al., respectivamente, devem testar até encontrar a chave. Os resultados são mostrados na Figura 6.3.3, e sugerem que o algoritmo probabilístico é significativamente melhor do que do que o algoritmo original para as d-exclusões consideradas. Não só o algoritmo probabilístico é mais eficiente do que o original quando são dadas as mesmas entradas, como o algoritmo probabilístico recuperando uma chave no nível de segurança de 256 bits parece ser mais eficiente do que o original recuperando uma chave do nível de 80 bits. É importante notar que, para d-exclusões muito baixas, o algoritmo original passa a ser mais eficiente do que o probabilístico, porém no ponto em que isso ocorre, o número esperado de caminhos até a chave ser encontrada já é grande o suficiente para ser impraticável o uso de

[12] Nesse caso o *speedup* usando 16 cores é bem ruim, deixando o algoritmo mais lento do que se fosse executado sem paralelização.

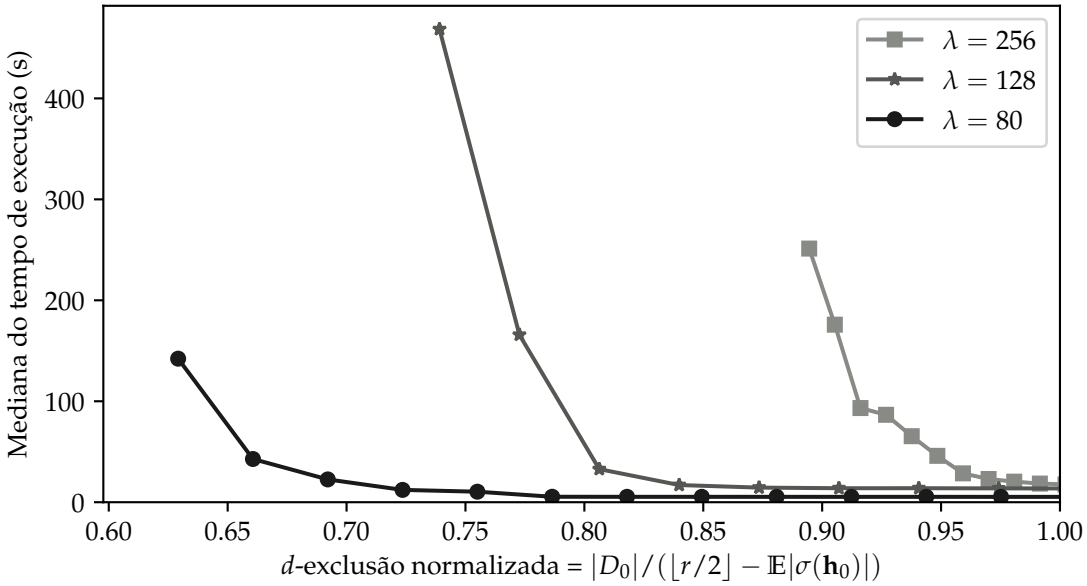

Figura 6.3.2: *Mediana do tempo de execução do algoritmo probabilístico*

qualquer um dos dois algoritmos.

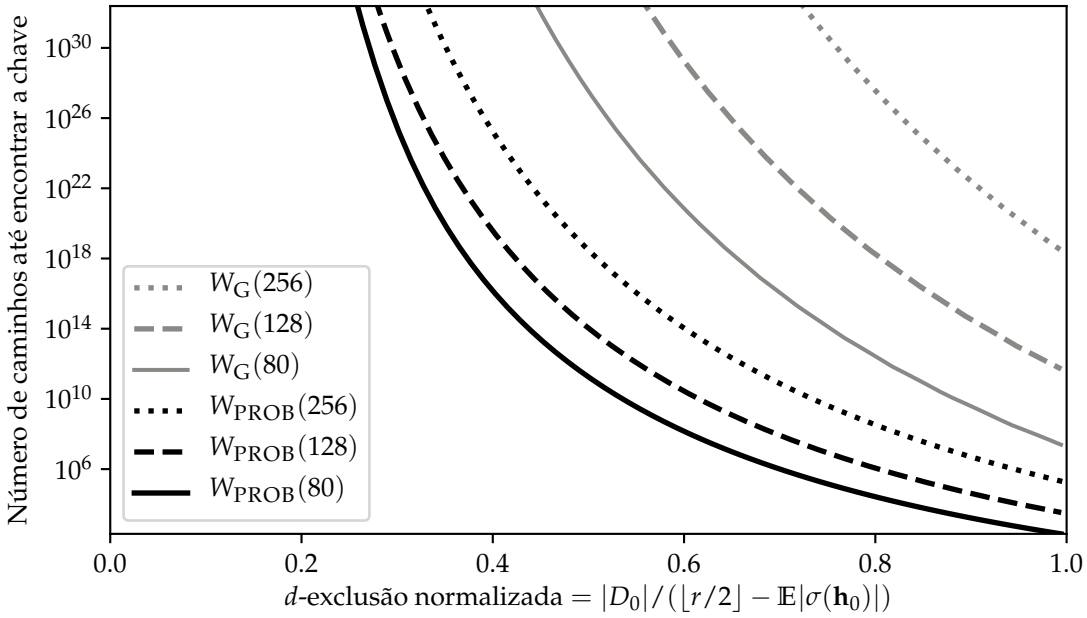

Figura 6.3.3: *Comparação entre os números esperados de caminhos percorridos até encontrar a chave para o algoritmo probabilístico (W_{PROB}) e para o algoritmo original de Guo et al. (W_{G}). Para cada curva, o valor entre parênteses é o nível de segurança correspondente*

Seja[13] q uma d-exclusão suficiente para que o algoritmo probabilístico termine rapidamente, onde rapidamente deve ser definido pelo atacante de maneira que leve em conta seu poder computacional. Por exemplo, para o nível de segurança de 80 bits, podemos considerar que o algoritmo trata eficientemente de uma d-exclusão de 1000 entradas, que normalizada equivale a apro-

[13]Este parágrafo e o próximo supõem que o leitor está familiarizado com as construções dos conjuntos D_0 e D_1 como descrito na Seção 5.4.

ximadamente 63% das distâncias fora do espectro. O algoritmo probabilístico, como o algoritmo de Guo et al., pode ser executado usando tanto D_0 como D_1, que são os conjuntos estimados das distâncias fora dos espectros correspondentes de $\mathbf{h}_0$ e $\mathbf{h}_1$. Então, para que o algoritmo possa encontrar a chave, basta que ao menos um dos conjuntos D_0 ou D_1 consista de q entradas fora de seu espectro correspondente. Em outras palavras, sejam d_0 e d_1 as d-exclusões obtidas para as saídas do algoritmo de estimação dos espectros, então, para o algoritmo recuperar a chave, é suficiente que

$$|D_0| = |D_1| = q \geq \max(d_0, d_1).$$

A Figura 6.3.4 mostra, para o nível de segurança de 80 bits, a relação entre $\max(d_0, d_1)$ e o número de decodificações. Os dados usados são os resultados dos testes de decodificação para os 200 códigos da Seção 5.4. Note que, para $q = 1000$, entre 30 e 50 milhões de testes de decodificação devem ser suficientes para que a chave seja recuperada. Isso é entre 4 e 6 vezes menos do que o número de testes de decodificação sugerido por Guo et al. [GJS16] para que a chave seja recuperada.

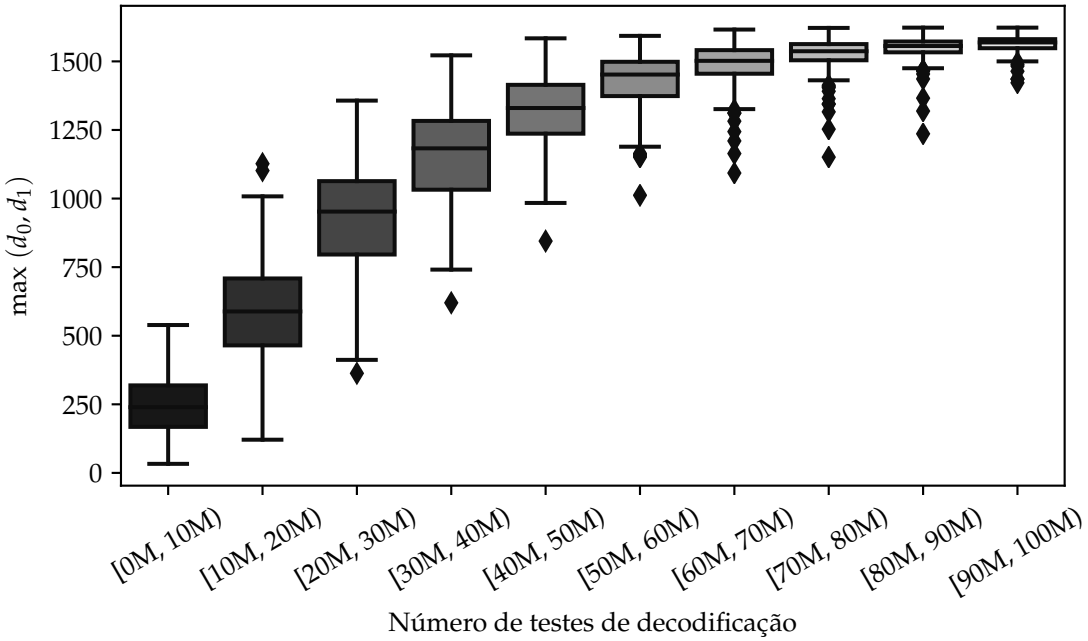

Figura 6.3.4: *A relação entre o número de testes de decodificação e o valor do máximo das d-exclusões obtidas para distâncias em* $\mathbf{h}_0$ *e* $\mathbf{h}_1$

Mesmo o algoritmo probabilístico obtendo bons resultados em relação ao algoritmo original, a d-exclusão de que ele consegue tratar eficientemente não é muito boa para níveis de segurança mais altos. A próxima seção mostra um algoritmo robusto em relação a esse problema.

Capítulo 7

Algoritmo iterativo de reconstrução baseado em álgebra linear

A seção anterior mostra que o simples fato de escolher aleatoriamente qual filho seguir na árvore de busca melhora significativamente o desempenho do algoritmo. Mesmo assim, a d-exclusão com que ele consegue trabalhar eficientemente não é muito baixa. Além disso, um problema mais crítico é que o desempenho do algoritmo probabilístico não escala muito bem para níveis de segurança mais altos, embora escale melhor do que o algoritmo original. Nesta seção, é proposto um novo algoritmo para a recuperação de chave que é totalmente baseado em álgebra linear em $\mathbb{F}_2$. Não sendo uma busca em árvore, o algoritmo consegue evitar os expoentes crescentes em sua complexidade para cada nível de segurança, presentes tanto no algoritmo original quanto no nosso algoritmo probabilístico de busca. Em particular, obedecidas certas condições sobre a d-exclusão, mostramos que sua complexidade é $\mathcal{O}(n^4)$. O desempenho do algoritmo na prática também é melhor do que os dos outros apresentados, e ainda tem a vantagem de poder ser paralelizado trivialmente. A d-exclusão de que ele precisa é significativamente menor do que a do algoritmo probabilístico de busca. Esta proposta foi submetida à revista *Transactions on Fundamentals of Electronics, Communications and Computer Sciences* do IEICE [PT17].

7.1 Descrição do algoritmo

Diferentemente dos outros algoritmos apresentados, este algoritmo usa necessariamente informação sobre os espectros de ambos os blocos da chave $\mathbf{h}_0$ e $\mathbf{h}_1$. Além destes dois conjuntos, o algoritmo também usa s_0 e s_1, que são, respectivamente, distâncias nos espectros de $\mathbf{h}_0$ e de $\mathbf{h}_1$. Todos esses parâmetros podem ser obtidos usando as descrições da Seção 5.4.

A ideia é explorar a relação $\mathbf{h}_1\mathbf{B} = \mathbf{h}_0$, onde a matriz $\mathbf{B}$ é pública, e o espectro parcial de $\mathbf{h}_1$ e de $\mathbf{h}_0$ pode ser obtido pelo algoritmo de estimação do espectro. Sejam Z_0 e Z_1 conjuntos de índices de entradas não nulas de $\mathbf{h}_0$ e $\mathbf{h}_1$, respectivamente. Denote por $\mathbf{B}^{Z_0}$ a matriz formada pelas colunas de $\mathbf{B}$ cujos índices estão em Z_0. Então, por definição

$$\mathbf{h}_1\mathbf{B}^{Z_0} = \mathbf{0}.$$

As entradas de $\mathbf{h}_1$ cujos índices estão em Z_1 podem ser descartadas, se for tomado o devido cuidado de também descartar as colunas correspondentes de $\mathbf{B}^{Z_0}$. Denote por Z_1' o complemento de Z_1 em relação aos possíveis índices das entradas de $\mathbf{h}_1$, ou seja $Z_1' = [r] - Z_1$. Então

$$\mathbf{h}_1^{Z_1'}\mathbf{B}_{Z_1'}^{Z_0} = \mathbf{0},$$

onde $\mathbf{h}_1^{Z_1'}$ é o vetor formado pelas colunas de $\mathbf{h}_1$ cujos índices estão em Z_1', e $\mathbf{B}_{Z_1'}^{Z_0}$ é a matriz formada pelas linhas de $\mathbf{B}^{Z_0}$ cujos índices estão em Z_1'. Em outras palavras, $\mathbf{h}_1^{Z_1'}$ é um vetor que pertence

ao espaço nulo da matriz $\mathbf{B}_{Z_1'}^{Z_0}$. Então podemos computar a matriz geradora do espaço nulo de $\mathbf{B}_{Z_1'}^{Z_0}$ e, com sorte, $\mathbf{h}_1^{Z_1'}$ será uma de suas colunas. Para transformar essa ideia num algoritmo, é preciso resolver alguns problemas, que são discutidos a seguir.

O primeiro problema é que encontrar um vetor de baixo peso no espaço nulo de $\mathbf{B}_{Z_1'}^{Z_0}$ pode ser impraticável [Ste88]. Uma maneira conservadora de lidar com esse problema é descobrir o quão grandes devem ser os conjuntos Z_0 e Z_1 para que o espaço nulo de $\mathbf{B}_{Z_1'}^{Z_0}$ consista somente no vetor $\mathbf{h}_1^{Z_1'}$, a partir do qual é direto obter $\mathbf{h}_1$. A resposta é dada formalmente na Seção 7.2, e é que o espaço nulo é formado apenas pelo vetor $\mathbf{h}_1^{Z_1'}$ com probabilidade aproximada de $1 - 1/2^{|Z_0|+|Z_1|-r}$. Assim, para o ataque ter boa probabilidade de sucesso, o ideal é que $|Z_0| + |Z_1|$ seja maior do que r, mas não precisa ser necessariamente muito maior, pois a probabilidade aumenta rapidamente em relação à soma. Os tamanhos dos conjuntos Z_0 e Z_1 estão intimamente relacionados às exclusões obtidas pelo algoritmo de estimação do espectro, o que leva ao próximo problema.

O segundo problema é determinar quanta informação sobre os espectros de $\mathbf{h}_0$ e $\mathbf{h}_1$ é necessária para que os conjuntos Z_0 e Z_1 sejam grandes o suficiente. Este problema está relacionado ao número necessário de decodificações para o ataque ter sucesso. Seja $\sigma(\mathbf{h}_0)$ o espectro $\mathbf{h}_0$, suponha que é conhecido que s_0 pertence a $\sigma(\mathbf{h}_0)$, e também que é sabido que as distâncias no conjunto $D_0 = \{f_1, \ldots, f_l\}$ não pertencem a $\sigma(\mathbf{h}_0)$. Então, denotando por $*$ entradas desconhecidas, é possível concluir que existe ao menos uma rotação do vetor $\mathbf{h}_0$ com o seguinte formato

$$[\underbrace{0*\ldots*\overbrace{\underbrace{1}_{f_1}*\ldots*\underbrace{0}_{f_1}}^{s_0}*\ldots*\underbrace{0}_{f_1}*\ldots*\underbrace{1}_{f_1}}*\ldots*\underbrace{0}_{f_1}*\ldots*].$$

Além disso, se forem considerados todos os outros f_i, é possível descobrir as posições de um grande número de entradas nulas dessa rotação de $\mathbf{h}_0$. Justamente estas posições de entradas nulas formam o conjunto Z_0. Uma construção análoga pode ser feita para o conjunto Z_1. Como exemplo, considere o vetor $\mathbf{h}_0 = [0\,0\,1\,1\,0\,0\,0\,1\,0]$, e então $\sigma(\mathbf{h}_0) = \{1, 4\}$. Suponha que sabemos que a distância $s_0 = 1$ pertence ao espectro $\sigma(\mathbf{h}_0)$, e que a distância no conjunto $D_0 = \{3\}$ não ocorrem em $\mathbf{h}_0$. Então podemos ter certeza de que existe ao menos uma rotação de $\mathbf{h}_0$ satisfazendo o seguinte formato

$$[1\,1\,*\,0\,0\,*\,0\,0\,*],$$

a partir do qual podemos construir o conjunto $Z_0 = \{4, 5, 7, 8\}$. Na Seção 7.2, é discutida a relação entre o número de distâncias conhecidas dentro e fora dos espectros $\sigma(\mathbf{h}_0)$ e $\sigma(\mathbf{h}_1)$, e os tamanhos dos conjuntos Z_0 e Z_1, respectivamente.

O terceiro e último problema é que os conjuntos de posições Z_0 e Z_1 são construídos com base apenas em distâncias circulares. Assim, não há nenhuma garantia de que Z_0 e Z_1 são posições de entradas nulas em rotações de $\mathbf{h}_0$ e $\mathbf{h}_1$ pelo mesmo número de posições. Em outras palavras, os conjuntos Z_0 e Z_1 são válidos, com alta probabilidade, para linhas diferentes de $\mathbf{H}_0$ e $\mathbf{H}_1$, respectivamente, e a relação $\mathbf{h}_1'\mathbf{B} = \mathbf{h}_0'$ vale somente se forem iguais os índices das linhas de $\mathbf{H}_0$ e $\mathbf{H}_1$ em que aparecem $\mathbf{h}_0'$ e $\mathbf{h}_1'$, respectivamente. Um modo de lidar com isso é fixar um dos conjuntos, a saber Z_1, e tentar iterativamente para cada $p = 0, 1, \ldots, r-1$ encontrar uma linha de baixo peso na matriz geradora do espaço nulo de $\mathbf{B}_{Z_1'}^{Z_0^{(p)}}$, onde $Z_0^{(p)}$ é o conjunto de posições em Z_0 circularmente deslocadas em p posições, ou seja

$$Z_0^{(p)} = \{i + p \mod r : i \in Z_0\},$$

onde $0 \mod r$ é identificado com a posição r, já que índices de vetores começam em 1.

O valor de p para o qual é possível encontrar a chave é aquele que alinha as devidas rotações dos vetores $\mathbf{h}_0$ e $\mathbf{h}_1$ correspondentes a Z_0 e Z_1. Na geração de chaves, os vetores $\mathbf{h}_0$ e $\mathbf{h}_1$ são construídos aleatoriamente, então, a priori, cada rotação de $\mathbf{h}_1$ é equiprovável de estar alinhada

com $\mathbf{h}_0$. Isso faz com que a probabilidade de que um deslocamento p alinhe os conjuntos Z_0 e Z_1 seja $1/r$, ou seja, a distribuição do p é uniforme no conjunto $[r]$. Na prática essa probabilidade pode ser ligeiramente mais alta, dependendo das multiplicidades com que as distâncias s_0 e s_1 aparecem nos respectivos vetores $\mathbf{h}_0$ e $\mathbf{h}_1$.

Considere o seguinte exemplo que ilustra o problema das rotações descrito acima.

$$\mathbf{H} = [\mathbf{H}_0|\mathbf{H}_1] = \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}\right], Z_0 = \{2,3\}, \text{ e } Z_1 = \{1\}.$$

Pode-se ver que os elementos de Z_0 correspondem às posições nulas da primeira linha do de $\mathbf{H}_0$. Porém o elemento de Z_1 corresponde à terceira linha de $\mathbf{H}_1$. Há várias formas de alinhar os dois conjuntos. Uma delas, que é a que escolhemos para o algoritmo, é rotacionar o conjunto Z_0 em 2 posições, obtendo o conjunto $Z_0^{(2)} = \{4 \mod 4, 5 \mod 4\} = \{0,1\} = \{4,1\}$. Note como $Z_0^{(2)}$ e Z_1 são posições de entradas nulas de uma mesma linha de $\mathbf{H}$, em relação a $\mathbf{H}_0$ e $\mathbf{H_1}$, respectivamente.

Agora é possível dar uma descrição completa do algoritmo, que pode ser vista no Algoritmo 10. Como o algoritmo não é recursivo e usa apenas operações comuns, não é difícil analisá-lo. As linhas 1, 3, e 5 são $\mathcal{O}(r)$. O custo de construir $\mathbf{B}_{Z_1'}$ na linha 2 é $\mathcal{O}(r|Z_1'|)$, e o custo de construir $\mathbf{B}_{Z_1'}^{Z_0}$ na linha 6 é $\mathcal{O}(|Z_0||Z_1'|)$. O laço da linha 4 faz r iterações no pior caso. O cálculo da matriz geradora do espaço nulo na linha 7 é a operação mais cara, custando $|Z_1'|^2|Z_1' + Z_0|$ operações. O cálculo do peso na linha 10 custa $\mathcal{O}(r)$. Finalmente, o laço da linha 12 itera $|\mathbf{v}| = |Z_1'|$ vezes. Isso dá o seguinte lema sobre o desempenho do algoritmo.

Algoritmo 10: Algoritmo de reconstrução de chave baseado em álgebra linear

Entrada: r, w parâmetros do código QC-MDPC
D_0 um conjunto de distâncias fora do espectro de $\mathbf{h}_0$
D_1 um conjunto de distâncias fora do espectro de $\mathbf{h}_1$
s_0 uma distância dentro do espectro de $\mathbf{h}_0$
s_1 uma distância dentro do espectro de $\mathbf{h}_1$
$\mathbf{B}$ o bloco cíclico à direita da matriz geradora

Saída: $\mathbf{h}_1$ ou $\perp$

1 $Z_1' \leftarrow \{i \in [r] : \text{dist}(1,i) \notin D_1 \text{ e } \text{dist}(1+s_1,i) \notin D_1\}$
2 $\mathbf{B}_{Z_1'} \leftarrow$ linhas de $\mathbf{B}$ cujos índices estão em Z_1'
3 $\overline{Z_0} \leftarrow \{i \in [r] : \text{dist}(1,i) \in D_0 \text{ ou } \text{dist}(1+s_0,i) \in D_0\}$
4 **para** $p = 0$ *to* $r-1$ **faça**
5 $\quad Z_0 \leftarrow \{i + p \mod r : i \in \overline{Z_0}\}$
6 $\quad \mathbf{B}_{Z_1'}^{Z_0} \leftarrow$ colunas de $\mathbf{B}^{Z_1'}$ cujos índices estão em Z_0
7 $\quad \mathbf{K} \leftarrow$ matriz geradora do núcleo de $\mathbf{B}_{Z_1'}^{Z_0}$
8 $\quad$ **se** $\dim \mathbf{K} = 1$ **então**
9 $\quad\quad \mathbf{v} \leftarrow$ única linha de de $\mathbf{K}$
10 $\quad\quad$ **se** $\mathrm{w}(\mathbf{v}) \leq w$ **então**
11 $\quad\quad\quad \mathbf{h}_1 \leftarrow \mathbf{0} \in \mathbb{F}_2^r$
12 $\quad\quad\quad$ **para** *cada* $i = 1$ *até* $|\mathbf{v}|$ **faça**
13 $\quad\quad\quad\quad \mathbf{h}_1[Z_1[i]] \leftarrow \mathbf{v}[i]$
14 $\quad\quad\quad$ **devolva** $\mathbf{h}_1$
15 **devolva** $\perp$

Lema 7.1.1. *A complexidade do algoritmo de reconstrução é*

$$\mathcal{O}\left(r + r|Z_1'| + r + r\left(r + |Z_0||Z_1'| + |Z_1'|^2|Z_0 + Z_1'| + |Z_1'|\right)\right).$$

Como $|Z_0|$ e $|Z_1'|$ são $\mathcal{O}(r)$, a complexidade do algoritmo pode ser simplificada para $\mathcal{O}(r^4)$.

Note que o algoritmo pode não encontrar a chave, e o lema apresentado não diz nada sobre a probabilidade de encontrá-la. Ele apenas mostra que, se o algoritmo encontrar a chave, ele a encontra em $\mathcal{O}(r^4)$ passos. As próximas seções são dedicadas a mostrar que o algoritmo tipicamente encontra a chave, desde que a d-exclusão obtida pelo algoritmo de estimação do espectro não seja muito baixa. Em particular, queremos encontrar a relação entre a probabilidade de a reconstrução ser bem sucedida e o tamanho da d-exclusão obtida.

7.2 Análise probabilística

A probabilidade de o ataque falhar

Sejam Z_0 e Z_1 conjuntos de índices de entradas não nulas de $\mathbf{h}_0$ e $\mathbf{h}_1$, possivelmente rotacionados pelo mesmo número de posições. Pela construção de Z_0 e Z_1, a matriz geradora do espaço nulo de $\mathbf{B}_{Z_1'}^{Z_0}$, denotada por $\mathbf{K}$, tem ao menos uma linha, porém em geral a sua dimensão pode ser muito grande. Para o algoritmo encontrar a chave, a matriz $\mathbf{K}$ deve ter exatamente uma linha. Em outras palavras, a equação linear em $\mathbf{x}$ dada por $\mathbf{x}\mathbf{B}_{Z_1'}^{Z_0} = \mathbf{0}$ deve ter exatamente uma solução.

O número de soluções para essa equação é exatamente o número de pares de vetores não nulos $\mathbf{u}$ e $\mathbf{v}$ tais que $\mathbf{u}\mathbf{B} = \mathbf{v}$, sujeitos a $\sup(\mathbf{u}) \cap Z_1 = \varnothing$ e $\sup(\mathbf{v}) \cap Z_0 = \varnothing$. Multiplicando os dois lados da equação por $\mathbf{H}_1$ à direita, que é uma matriz não singular, obtemos

$$\mathbf{u}\mathbf{B} = \mathbf{v} \Leftrightarrow \mathbf{u}\mathbf{B}\mathbf{H}_1 = \mathbf{v}\mathbf{H}_1 \Leftrightarrow \mathbf{u}\left(\mathbf{H}_1^{-1}\mathbf{H}_0\right)\mathbf{H}_1 = \mathbf{v}\mathbf{H}_1.$$

Mas como o produto de matrizes cíclicas é comutativo, então

$$\mathbf{u}\mathbf{H}_0 = \mathbf{v}\mathbf{H}_1 \Leftrightarrow \mathbf{u}\mathbf{H}_0 + \mathbf{v}\mathbf{H}_1 = \mathbf{0} \Leftrightarrow [\mathbf{u}|\mathbf{v}]\left[\mathbf{H}_0^T|\mathbf{H}_1^T\right]^T = \mathbf{0}.$$

Ou seja, $\mathbf{u}\mathbf{B} = \mathbf{v}$ se e somente se o vetor $[\mathbf{u}|\mathbf{v}]$ é uma palavra do código QC-MDPC com matriz de paridade dada por $\mathbf{H}' = \left[\mathbf{H}_0^T|\mathbf{H}_1^T\right]$.

O problema de contar o número de pares desses vetores $\mathbf{u}$ e $\mathbf{v}$ está relacionado à distribuição de pesos num código QC-MDPC, que é um problema difícil, e para o qual não parece haver resultados teóricos que possamos usar [Gal62, RL09]. Esse problema de contagem fica ainda pior com as restrições sobre os suportes de $\mathbf{u}$ e $\mathbf{v}$. Para lidar com isso, podemos considerar uma aproximação comumente usada [Lev68, Cor64], que considera que a probabilidade de o vetor $[\mathbf{u}|\mathbf{v}]$ estar num código $\mathcal{C}$ é a probabilidade de que um vetor escolhido aleatória e uniformemente pertença a $\mathcal{C}$, independentemente das restrições sobre $\mathbf{u}$ e $\mathbf{v}$. Então seja $\mathcal{C}$ um código $[n, r, w]$-QC-MDPC, temos a aproximação

$$\Pr\left([\mathbf{u}|\mathbf{v}] \in \mathcal{C} | \sup(\mathbf{u}) \cap Z_1 = \sup(\mathbf{v}) \cap Z_0 = \varnothing\right) = 2^{-(n-r)}.$$

Neste trabalho, considerado apenas o caso $n = 2r$ é considerado, então a probabilidade de o algoritmo encontrar a chave pode ser aproximada como

$$\begin{aligned}\Pr(\dim \mathbf{K} > 1) &\approx \sum_{\mathbf{u}:\sup(\mathbf{u})\cap Z_1=\varnothing} \sum_{\mathbf{v}:\sup(\mathbf{v})\cap Z_0=\varnothing} \Pr\left([\mathbf{u}|\mathbf{v}] \in \mathcal{C} | \sup(\mathbf{u}) \cap Z_1 = \sup(\mathbf{v}) \cap Z_0 = \varnothing\right) \\ &\approx 2^{r-|Z_0|}2^{r-|Z_1|}\frac{1}{2^r}.\end{aligned}$$

Isso dá o seguinte lema relacionando a probabilidade de o algoritmo encontrar a chave e os tamanhos dos conjuntos Z_0 e Z_1.

Lema 7.2.1. *A probabilidade de o algoritmo encontrar a chave pode ser aproximada por*

$$\Pr(\textit{Algoritmo encontrar a chave}) \approx 1 - 2^{r-|Z_0|-|Z_1|}.$$

O tamanho da d-exclusão para uma alta probabilidade de sucesso

Sabemos pelo Lema 7.2.1 que o valor de $|Z_0| + |Z_1|$ determina a probabilidade de o algoritmo encontrar a chave, para um r fixo. Sejam D_0 e D_1, os conjuntos das distâncias que sabemos estarem fora dos espectros de $\mathbf{h}_0$ e $\mathbf{h}_1$, respectivamente, isto é, são as distâncias d-excluídas pelo algoritmo de estimação dos espectros. Estamos interessados em entender como os tamanhos de D_0 e D_1 afetam os tamanhos dos conjuntos Z_0 e Z_1, respectivamente.

Fixe a probabilidade de o ataque falhar como sendo $1/2^\gamma$, para um certo γ. Queremos encontrar o menor inteiro positivo d tal que, se ambos $|D_0|$ e $|D_1|$ forem maiores ou iguais a d, então o algoritmo encontra a chave com probabilidade maior do que $1 - 1/2^\gamma$, para uma grande fração α de todas as chaves secretas possíveis. Note que queremos que α e γ sejam grandes.

A ideia é fazer uma busca binária pelo menor tal d no conjunto $[\lfloor r/2 \rfloor] = \{1, \ldots, \lfloor r/2 \rfloor\}$. O problema que enfrentamos é que a interdependência entre as distâncias fora do espectro torna difícil computar analiticamente a distribuição do conjunto $|Z_0|$ como uma função de $|D_0|$, e similarmente para $|Z_1|$ e $|D_1|$. Note que, na Seção 6.2, foi resolvido um problema similar, porém mais simples pois o interesse era apenas no valor esperado de $|Z_0|$. Agora, como a probabilidade de o algoritmo encontrar a chave é exponencial em $|Z_0|$ e $|Z_1|$, apenas o valor esperado não basta. Para resolver o problema, usamos simulações para estimar, para cada d, a fração f_d das chaves secretas para as quais o algoritmo encontra a chave secreta com probabilidade maior ou igual a $1 - 1/2^\gamma$ quando $|D_0| = |D_1| = d$. Para isso, são construídos N pares válidos (D_0, Z_0) e (D_1, Z_1), e o estimador para f_d é dado por

$$\hat{f}_d = \frac{\#\{\text{Pares } Z_0 \text{ e } Z_1 \text{ gerados} : |Z_0| + |Z_1| - r \geq \gamma\}}{N}.$$

A busca termina com o menor valor de d para que $\hat{f}_d \geq \alpha$.

A Tabela 7.2.1 mostra os valores de d obtidos para diferente níveis de segurança considerando como parâmetros da simulação $N = 5000$, $\alpha = 99\%$, e $\gamma = 20$. Para uma comparação entre diferentes níveis de segurança, também são mostradas as frações d/σ', onde σ' é o número médio de distâncias fora do espectro.

Nível de segurança λ	Tamanho estimado para D_0 e D_1 pela simulação d	Número médio de distâncias fora do espectro σ'	d-exclusão normalizada d/σ'
80	$d_{80} = 722$	1582.12	45.63%
128	$d_{128} = 1470$	2964.72	49.58%
256	$d_{256} = 4845$	9256.64	52.34%

Tabela 7.2.1: *Resultados da simulação para os valores de d, usando como parâmetros $N = 5000$, $\alpha = 99\%$, e $\gamma = 20$*

7.3 Resultados experimentais

A análise do algoritmo apresentada mostra que esse algoritmo tem melhor complexidade assintótica do que os outros baseados em buscas em árvores. Agora são considerados os resultados

do seu desempenho na prática. O algoritmo foi implementado em linguagem C, usando a biblioteca M4RI [Mar12] para o cálculo da matriz geradora do espaço nulo.

A análise foi feita para os parâmetros que oferecem níveis de segurança 80, 128, e 256 bits usando $n_0 = 2$ blocos cíclicos. Os testes de desempenho do algoritmo foram feitos no limite da d-exclusão mínima que o algoritmo precisa para encontrar consistentemente a chave, ou seja, os valores de d dados pela Tabela 7.2.1 para os tamanhos dos conjuntos D_0 e D_1.

Note que o tempo total de execução, ou mesmo o número total de ciclos, não é uma métrica muito informativa para esse algoritmo. Isso pois o número de iterações necessário para recuperar a chave segue uma distribuição uniforme sobre o conjunto $[r]$. Assim, sabendo o tempo médio de execução para cada iteração, ou o número de ciclos por iteração, é possível modelar com maior precisão o desempenho do algoritmo.

A Tabela 7.3.1 mostra o desempenho do algoritmo num computador com um processador i7 870 Lynnfield, com frequência de 2.93GHz, e 8GB de RAM.

Nível de segurança λ	Tempo médio por iteração	Número médio de ciclos por iteração	Tempo médio de execução
80	0.0169s	521048150	**41s**
128	0.1710s	4454356686	**14m3s**
256	3.4179s	87756949017	**15h33m24s**

Tabela 7.3.1: *O desempenho do algoritmo de reconstrução para diferentes níveis de segurança*

Como na análise experimental do algoritmo probabilístico, também estamos interessado no número de testes de decodificação necessários para que as d-exclusões tanto para $\mathbf{h}_0$ quanto $\mathbf{h}_1$ sejam suficientes para que o algoritmo encontre a chave. Chame de $d_0(i)$ e $d_1(i)$ as d-exclusões obtidas para $\mathbf{h}_0$ e $\mathbf{h}_1$, respectivamente, com i testes de decodificação. Teoricamente, bastaria tomar o primeiro i tal que $d_0(i) + d_1(i) \geq 2d_\lambda$. Mas, como discutido na Seção 5.4, há o problema de que, para definir os conjuntos D_0 e D_1 é necessário saber quais são os valores de $d_0(i)$ e $d_1(i)$, o que não se sabe a priori. Uma ideia conservadora para lidar com isso é sempre construir os conjuntos D_0 e D_1 com as d_λ distâncias com maior taxa estimada de falha para distâncias em $\mathbf{h}_0$ e em $\mathbf{h}_1$, respectivamente. Então o algoritmo pode encontrar a chave após i testes de decodificação se $\min(d_0(i), d_1(i)) \geq d_\lambda$.

Na prática, porém, rodar o algoritmo de construção a cada iteração do algoritmo de estimação é muito ineficiente. Melhor seria usar uma política como: faça $M_0 = 10$ milhões de testes de decodificação, e depois, a cada $M_i = 100000$ testes de decodificação, rode o algoritmo de reconstrução de chave. Assim a chave é encontrada relativamente cedo, mas sem perder muito tempo rodando o algoritmo de reconstrução sem sucesso.

Para os experimentos, foi considerado apenas o nível de segurança de 80 bits, que é o mesmo nível de segurança considerado por Guo et al.. Não consideramos níveis de segurança mais altos pois as simulações tomam muito tempo, mas o código-fonte disponibilizado pode ser usado diretamente para rodar simulações para níveis de segurança mais altos. Os dados usados nesta seção são os mesmos resultados de decodificação mostrados na Seção 5, isto é, são testes de decodificação para 200 códigos [9602, 4801, 90]-QC-MDPC. A Figura 7.3.1 mostra o histograma das primeiras iterações i que são um múltiplo de 100000 e em que $\min(d_0(i), d_1(i)) \geq d_{80} = 722$, considerando os resultados dos 200 códigos. Esses resultados sugerem que esse algoritmo recupera a chave com um número de testes de decodificação M entre 5 e 10 vezes menor do que o necessário para que o algoritmo de Guo et al. tenha sucesso.

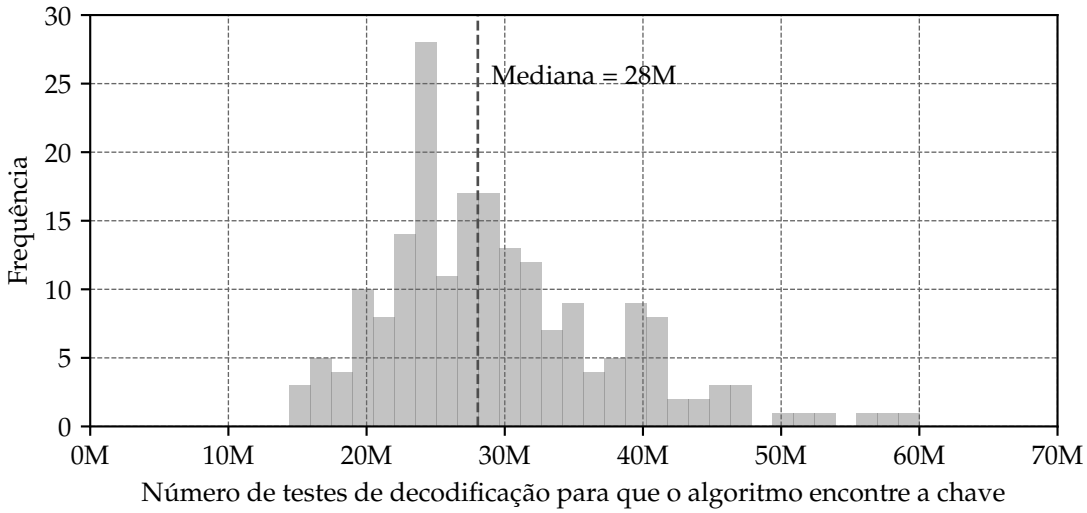

Figura 7.3.1: *Histograma do número de testes de decodificação para que o algoritmo encontre a chave*

Referências Bibliográficas

[AB09] Sanjeev Arora e Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. 1, 14

[ABB+15] Daniel Augot, Lejla Batina, Daniel J Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, O Dunkelmann, Tim Güneysu, Shay Gueron, Andreas Hülsing et al. Initial recommendations of long-term secure post-quantum systems. *URL: https://pqcrypto. eu. org/docs/initial-recommendations. pdf. Citations in this document*, 16, 2015. 2

[AKS04] Manindra Agrawal, Neeraj Kayal e Nitin Saxena. Primes is in p. *Annals of mathematics*, páginas 781–793, 2004. 14

[BBD09] Daniel J Bernstein, Johannes Buchmann e Erik Dahmen. *Post-quantum cryptography*. Springer Science & Business Media, 2009. 1, 8

[BCGO09] Thierry P Berger, Pierre-Louis Cayrel, Philippe Gaborit e Ayoub Otmani. Reducing key length of the McEliece cryptosystem. Em *Progress in Cryptology–AFRICACRYPT 2009*, páginas 77–97. Springer, 2009. 2, 21

[BCS13] Daniel J Bernstein, Tung Chou e Peter Schwabe. McBits: fast constant-time code-based cryptography. Em *International Workshop on Cryptographic Hardware and Embedded Systems*, páginas 250–272. Springer, 2013. 1, 8, 17

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval e Phillip Rogaway. Relations among notions of security for public-key encryption schemes. Em *Advances in Cryptology—CRYPTO'98*, páginas 26–45. Springer, 1998. 2, 16, 17

[Ber73] Elwyn R Berlekamp. Goppa codes. *Information Theory, IEEE Transactions on*, 19(5):590–592, 1973. 2, 7, 17

[BLP08] Daniel J Bernstein, Tanja Lange e Christiane Peters. Attacking and defending the McEliece cryptosystem. Em *Post-Quantum Cryptography*, páginas 31–46. Springer, 2008. 8

[BMVT78] Elwyn R Berlekamp, Robert J McEliece e Henk CA Van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. 1, 7, 18

[BR93] Mihir Bellare e Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. Em *Proceedings of the 1st ACM conference on Computer and communications security*, páginas 62–73. ACM, 1993. 19

[BSC16] Marco Baldi, Paolo Santini e Franco Chiaraluce. Soft McEliece: MDPC code-based McEliece cryptosystems with very compact keys through real-valued intentional errors. páginas 795–799, 2016. 21

[BT+06] Andrej Bogdanov, Luca Trevisan et al. Average-case complexity. *Foundations and Trends® in Theoretical Computer Science*, 2(1):1–106, 2006. 1

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner e Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016. 1, 15

[CFS01] Nicolas T Courtois, Matthieu Finiasz e Nicolas Sendrier. How to achieve a McEliece-based digital signature scheme. Em *International Conference on the Theory and Application of Cryptology and Information Security*, páginas 157–174. Springer, 2001. 1

[Cho16] Tung Chou. QcBits: constant-time small-key code-based cryptography. Em *International Conference on Cryptographic Hardware and Embedded Systems*, páginas 280–300. Springer, 2016. 3

[Cor64] F Corr. A statistical evaluation of error detection cyclic codes for data transmission. *IEEE Transactions on Communications Systems*, 12(2):211–216, 1964. 52

[DDN03] Danny Dolev, Cynthia Dwork e Moni Naor. Nonmalleable cryptography. *SIAM review*, 45(4):727–784, 2003. 16

[DES77] DES. Data encryption standard. Em *In FIPS PUB 46, Federal Information Processing Standards Publication*, páginas 46–2, 1977. 12

[DH76] Whitfield Diffie e Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976. 12

[DR13] Joan Daemen e Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013. 12

[FGUO+13] Jean-Charles Faugere, Valérie Gauthier-Umana, Ayoub Otmani, Ludovic Perret e Jean-Pierre Tillich. A distinguisher for high-rate McEliece cryptosystems. *IEEE Transactions on Information Theory*, 59(10):6830–6844, 2013. 2, 8, 18

[FHS+17] Tomáš Fabšič, Viliam Hromada, Paul Stankovski, Pavol Zajac, Qian Guo e Thomas Johansson. A reaction attack on the QC-LDPC McEliece cryptosystem. Em *International Workshop on Post-Quantum Cryptography*, páginas 51–68. Springer, 2017. 3, 21

[FO99] Eiichiro Fujisaki e Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. Em *Annual International Cryptology Conference*, páginas 537–554. Springer, 1999. 19

[FOP+16] Jean-Charles Faugere, Ayoub Otmani, Ludovic Perret, Frédéric De Portzamparc e Jean-Pierre Tillich. Structural cryptanalysis of McEliece schemes with compact keys. *Designs, Codes and Cryptography*, 79(1):87–112, 2016. 2, 21

[FOPT10] Jean-Charles Faugere, Ayoub Otmani, Ludovic Perret e Jean-Pierre Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. Em *Advances in Cryptology–Eurocrypt 2010*, páginas 279–298. Springer, 2010. 2, 21

[FS09] Matthieu Finiasz e Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. Em *Asiacrypt*, volume 5912, páginas 88–105. Springer, 2009. 19

[Gab05] Philippe Gaborit. Shorter keys for code based cryptography. Em *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, páginas 81–91, 2005. 2, 21

[Gal62] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962. 8, 9, 21, 38, 52

[Gal12] Steven D Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012. 12

[GHR95] Raymond Greenlaw, H James Hoover e Walter L Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press on Demand, 1995. 33

[GJ16] Qian Guo e Thomas Johansson. A p-ary MDPC scheme. Em *Information Theory (ISIT), 2016 IEEE International Symposium on*, páginas 1356–1360. IEEE, 2016. 21

[GJS16] Qian Guo, Thomas Johansson e Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. Em *22nd Annual International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*, 2016. 2, 3, 9, 16, 21, 25, 27, 31, 35, 38, 41, 44, 47

[GM84] Shafi Goldwasser e Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984. 15, 17

[Gop70] Valerii Denisovich Goppa. A new class of linear correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970. 2, 7, 17

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. Em *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, páginas 212–219. ACM, 1996. 1

[Guo16] Qian Guo. Guo's presentation at Asiacrypt. https://youtu.be/tKvDdGLJLZc?t=1006, 2016. 32, 33

[HGS99] Chris Hall, Ian Goldberg e Bruce Schneier. Reaction attacks against several public-key cryptosystem. Em *International Conference on Information and Communications Security*, páginas 2–12. Springer, 1999. 2

[HP10] W Cary Huffman e Vera Pless. *Fundamentals of error-correcting codes*. Cambridge university press, 2010. 9, 10

[HPS98] Jeffrey Hoffstein, Jill Pipher e Joseph H Silverman. Ntru: A ring-based public key cryptosystem. Em *International Algorithmic Number Theory Symposium*, páginas 267–288. Springer, 1998. 2

[HVMG13] Stefan Heyse, Ingo Von Maurich e Tim Güneysu. Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. Em *Cryptographic Hardware and Embedded Systems-CHES 2013*, páginas 273–292. Springer, 2013. 23

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. Em *Structure in Complexity Theory Conference, 1995., Proceedings of Tenth Annual IEEE*, páginas 134–147. IEEE, 1995. 1

[KI01] Kazukuni Kobara e Hideki Imai. Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC. Em *International Workshop on Public Key Cryptography*, páginas 19–35. Springer, 2001. 2, 19, 20, 22

[LB88] Pil Joong Lee e Ernest F Brickell. An observation on the security of mceliece's public-key cryptosystem. Em *Eurocrypt*, volume 88, páginas 275–280. Springer, 1988. 19

[Lev68] J Levy. A weight distribution bound for linear codes. *IEEE Transactions on Information Theory*, 14(3):487–490, 1968. 52

[Mac99] David JC MacKay. Good error-correcting codes based on very sparse matrices. *IEEE transactions on Information Theory*, 45(2):399–431, 1999. 23

[Mar12] Martin Albrecht and Gregory Bard. *The M4RI Library – Version 20121224*. The M4RI Team, 2012. 54

[MB09] Rafael Misoczki e Paulo SLM Barreto. Compact McEliece keys from goppa codes. Em *Selected Areas in Cryptography*, páginas 376–392. Springer, 2009. 2, 21

[McE78] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Deep Space Network Progress Report*, 44:114–116, 1978. 1, 7, 17

[Mil86] V.S. Miller. Use of elliptic curves in cryptography. *Advances in Cryptology (CRYPTO85)*, páginas 417–426, 1986. 1, 12, 15, 17

[MOG15] Ingo Von Maurich, Tobias Oder e Tim Güneysu. Implementing QC-MDPC McEliece encryption. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(3):44, 2015. 9, 21, 22, 23, 28, 38

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier e Paulo SLM Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. Em *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, páginas 2069–2073. IEEE, 2013. 2, 5, 21, 22, 24

[MVOV96] Alfred J Menezes, Paul C Van Oorschot e Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996. 1, 17

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory, problems of control and information theory 15, 159–166. *Citations in this document*, 1(6), 1986. 1

[NY90] Moni Naor e Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. Em *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, páginas 427–437. ACM, 1990. 16

[OS09] Raphael Overbeck e Nicolas Sendrier. Code-based cryptography. Em *Post-quantum cryptography*, páginas 95–145. Springer, 2009. 1, 18, 19

[OTD10] Ayoub Otmani, Jean-Pierre Tillich e Léonard Dallot. Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes. *Mathematics in Computer Science*, 3(2):129–140, 2010. 2

[Pat75] Nicholas J Patterson. The algebraic decoding of goppa codes. *Information Theory, IEEE Transactions on*, 21(2):203–207, 1975. 8

[Poi00] David Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. Em *International Workshop on Public Key Cryptography*, páginas 129–146. Springer, 2000. 19

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962. 18, 19

[PT17] Thales Bandiera Paiva e Routo Terada. Improving the efficiency of a reaction attack on the QC-MDPC McEliece (submetido em outubro de 2017). *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2017. 4, 49

[Rei85] John H Reif. Depth-first search is inherently sequential. *Information Processing Letters*, 20(5):229–234, 1985. 33

[RHHM17] Mélissa Rossi, Mike Hamburg, Michael Hutter e Mark E Marson. A side-channel assisted cryptanalytic attack against QcBits. Em *International Conference on Cryptographic Hardware and Embedded Systems*, páginas 3–23. Springer, 2017. 3

[RK87] V Nageshwara Rao e Vipin Kumar. Parallel depth first search. part i. implementation. *International Journal of Parallel Programming*, 16(6):479–499, 1987. 33

[RL09] William Ryan e Shu Lin. *Channel codes: classical and modern*. Cambridge University Press, 2009. 8, 9, 37, 52

[RS91] Charles Rackoff e Daniel R Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. Em *Annual International Cryptology Conference*, páginas 433–444. Springer, 1991. 16

[RS94] Alexander Reinefeld e Volker Schnecke. Work-load balancing in highly parallel depth-first search. Em *Scalable High-Performance Computing Conference, 1994., Proceedings of the*, páginas 773–780. IEEE, 1994. 33

[RSA78] R. L. Rivest, A. Shamir e L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. 1, 12, 14, 15, 17

[Sha48] Claude E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948. 6

[Sho97] P.W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1481–1509, 1997. 1, 15

[SMR00] Amin Shokrollahi, Chris Monico e Joachim Rosenthal. Using low density parity check codes in the McEliece cryptosystem. Em *IEEE International Symposium on Information Theory (ISIT 2000)*, página 215, 2000. 2, 21

[SS11] Damien Stehlé e Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. Em *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, páginas 27–47. Springer, 2011. 2

[Ste88] Jacques Stern. A method for finding codewords of small weight. Em *International Colloquium on Coding Theory and Applications*, páginas 106–113. Springer, 1988. 19, 50

[Wei08] Eric W Weisstein. q-Pochhammer symbol. 2008. 43