Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classical leakage-resilient circuits from quantum fault-tolerant computation

Felipe G. Lacerda

Tese apresentada como requisito parcial
para conclusão do Doutorado em Informática

Orientador
Prof. Dr. Anderson C.A. Nascimento

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Doutorado em Informática

Coordenadora: Prof.ª Dr.ª Alba Cristina Magalhães de Melo

Banca examinadora composta por:

Prof. Dr. Anderson C.A. Nascimento (Orientador) — Universidade de Brasília
Prof. Dr. Maurício Ayala Rincón — Universidade de Brasília
Prof. Dr. Renato Renner — ETH Zürich
Prof.ª Dr.ª Anne Broadbent — University of Ottawa
Prof. Dr. Martin Rötteler — Microsoft Research

CIP — Catalogação Internacional na Publicação

Lacerda, Felipe G..

Classical leakage-resilient circuits from quantum fault-tolerant computation / Felipe G. Lacerda. Brasília : UnB, 2015.
107 p. : il. ; 29,5 cm.

Tese (Doutorado) — Universidade de Brasília, Brasília, 2015.

1. mecânica quântica, 2. criptografia, 3. computação, 4. resistência a vazamentos

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classical leakage-resilient circuits from quantum fault-tolerant computation

Felipe G. Lacerda

Tese apresentada como requisito parcial
para conclusão do Doutorado em Informática

Prof. Dr. Anderson C.A. Nascimento (Orientador)
Universidade de Brasília

Prof. Dr. Maurício Ayala Rincón
Universidade de Brasília

Prof. Dr. Renato Renner
ETH Zürich

Prof.ª Dr.ª Anne Broadbent
University of Ottawa

Prof. Dr. Martin Rötteler
Microsoft Research

Prof.ª Dr.ª Alba Cristina Magalhães de Melo
Coordenadora do Doutorado em Informática

Brasília, 30 de julho de 2015

Agradecimentos

Ao meu orientador, Prof. Dr. Anderson C.A. Nascimento, por ter me introduzido ao mundo da pesquisa em criptografia, por ter me ajudado a cada momento de dificuldade do doutorado, e por ter combatido com sucesso o meu pessimismo.

Ao Prof. Dr. Renato Renner, por ter me dado a oportunidade de trabalhar ao lado de algumas das pessoas mais inteligentes e esforçadas que já conheci, e por ter guiado alguns dos principais desenvolvimentos deste trabalho. Ao Dr. Joe Renes por ter me guiado ainda mais nestes desenvolvimentos.

Aos ex-colegas do Laboratório de Cálculo Científico da Universidade de Brasília, pela camaradagem e pelas cervejas. Em particular a Pedro Garcia e Bernardo David, por ter me convencido a continuar no meio acadêmico e pelo último ter me apresentado ao meu orientador.

Aos meus pais, pelo carinho e pelo apoio incondicional.

Finalmente, à CAPES, ao Swiss National Science Foundation e ao European Research Council pelo apoio financeiro.

Resumo

Implementações físicas de algoritmos criptográficos vazam informação, o que os torna vulneráveis aos chamados ataques de canal lateral. Atualmente, criptografia é utilizada em uma variedade crescente de cenários, e frequentemente a suposição de que a execução de criptossistemas é fisicamente isolada não é realista.

A área de *resistência a vazamentos* propõe mitigar ataques de canal lateral projetando protocolos que são seguros mesmo se a informação vaza durante a execução. Neste trabalho, estudamos computação resistente a vazamento, que estuda o problema de executar computação universal segura na presença de vazamento.

Computação quântica tolerante a falhas se preocupa com o problema de *ruído* em computadores quânticos. Uma vez que é extremamente difícil isolar sistemas quânticos de ruído, a área de tolerância a falhas propões esquemas para executar computações corretamente mesmo se há algum ruído.

Existe uma conexão entre resistência a vazamento e tolerância a falhas. Neste trabalho, mostramos que vazamento em um circuito *clássico* é uma forma de ruído, quando o circuito é interpretado como um circuito *quântico*. Posteriormente, provamos que para um modelo de vazamento arbitrário, existe um modelo de ruído correspondente para o qual um circuito que é tolerante a falhas de acordo com um modelo de ruído também é resistente a vazamento de acordo com o modelo de vazamento dado.

Também mostramos como utilizar construções para tolerância a falhas para implementar circuitos clássicos que são seguros em modelos de vazamento específicos. Isto é feito estabelecendo critérios para os quais circuitos quânticos podem ser convertidos em circuitos clássicos de certa forma que a propriedade de resistência a vazamentos é preservada. Usando estes critérios, convertemos uma implementação de computação quântica tolerante a falhas em um compilador resistente a vazamentos clássicos, isto é, um esquema que compila um circuito arbitrário em um circuito de mesma funcionalidade que é resistente a vazamentos.

Palavras-chave: mecânica quântica, criptografia, computação, resistência a vazamentos

Abstract

Physical implementations of cryptographic algorithms leak information, which makes them vulnerable to so-called side-channel attacks. Cryptography is now used in an ever-increasing variety of scenarios, and the assumption that the execution of cryptosystems is physically insulated is often not realistic.

The field of *leakage resilience* proposes to mitigate side-channel attacks by designing protocols that are secure even if information leaks during execution. In this work, we study leakage-resilient computation, which concerns the problem of performing secure universal computation in the presence of leakage.

Fault-tolerant quantum computation is concerned with the problem of *noise* in quantum computers. Since it is very hard to insulate quantum systems from noise, fault tolerance proposes schemes for performing computations correctly even if some noise is present.

It turns out that there exists a connection between leakage resilience and fault tolerance. In this work, we show that leakage in a *classical* circuit is a form of noise, when the circuit is interpreted as *quantum*. We then prove that for an arbitrary leakage model, there exists a corresponding noise model in which a circuit that is fault-tolerant against the noise model is also resilient against the given leakage model.

We also show how to use constructions for fault tolerance to implement classical circuits that are secure in specific leakage models. This is done by establishing criteria in which quantum circuits can be converted into classical circuits in such a way that the leakage resilience property is preserved. Using these criteria, we convert an implementation of universal fault-tolerant quantum computation into a classical leakage-resilient compiler, i.e., a scheme that compiles an arbitrary circuit into a circuit of the same functionality that is leakage-resilient.

Sumário

1 Introdução **1**

1.1 Trabalhos anteriores 3

1.2 Contribuição desta tese 4

2 Preliminares matemáticos **7**

2.1 Qubits e circuitos quânticos 7

2.1.1 O grupo de Pauli 8

2.1.2 Portas lógicas quânticas 8

2.1.3 Grupo de Clifford e universalidade quântica 10

2.1.4 Universalidade clássica e circuitos reversíveis 11

2.1.5 Operadores densidade e canais quânticos 15

2.1.6 Medições quânticas 16

2.2 Códigos corretores de erro quânticos 18

2.2.1 Propriedades de QECCs 20

2.2.2 Códigos estabilizadores 21

2.2.3 Códigos CSS 24

2.2.4 O código de Steane 26

3 Computação confiável e resistência a vazamentos **28**

3.1 Segurança demonstrável 28

3.2 Criptografia abstrata 29

3.2.1 Recursos 30

3.3 Definição de segurança para resistência a vazamentos 34

3.3.1 Motivação 34

3.3.2 Recurso ideal 35

3.3.3 Recurso real 36

3.4 Modelos de vazamento 38

3.5 Resistência a vazamentos na prática 41

3.5.1 Análise de consumo 41

3.5.2 Ataques acústicos . . . 42
3.5.3 *Probing* . . . 43
3.5.4 Contramedidas . . . 43
3.6 Computadores confiáveis . . . 43
3.7 Circuitos confiáveis são resistentes a vazamento . . . 44

4 Tolerância a falhas **49**
4.1 Circuitos tolerantes a falhas . . . 49
4.1.1 Portas lógicas . . . 50
4.1.2 Correção de erros . . . 52
4.1.3 Preparação de estados . . . 55
4.1.4 Medição . . . 59
4.2 Tolerância a falhas com códigos de distância 3 . . . 60
4.2.1 Portas lógicas . . . 60
4.2.2 Porta Toffoli . . . 62
4.2.3 Medições . . . 65
4.2.4 Preparação de estados . . . 66
4.2.5 O *Accuracy Threshold Theorem* . . . 73

5 Resistência a vazamentos concreta a partir de tolerância a falhas **79**
5.1 Tradução clássica de circuitos quânticos . . . 79
5.2 *Gadgets* resistentes a vazamentos . . . 82
5.2.1 Medições . . . 84
5.2.2 Correção de erros . . . 84
5.2.3 Preparação de estados . . . 85
5.2.4 Portas lógicas . . . 86
5.3 Discussão . . . 86

6 Conclusão **90**

Referências **92**

Lista de Figuras

2.1 Portas quânticas utilizadas neste trabalho. 11
2.2 Implementação da transposição (010 011) com portas Toffoli e X. 12
2.3 $T^{(4)}$ a partir de $T^{(3)}$. 13
2.4 $T^{(n)}$ a partir de $T^{(n-1)}$ e $T^{(3)}$. 14

3.1 Canal privado. 31
3.2 Recurso filtrado. 33
3.3 (a) Recurso ideal para resistência a vazamentos e (b) recurso real. Em ambos os casos, Alice tem acesso à interface mostrada à esquerda e Eva tem acesso à interface mostrada à direita. 36
3.4 A primeira condição na Definição 3 aplicada a resistência a vazamentos. Para provar que o conversor π_A é parte de um protocolo resistente a vazamentos π, é necessário mostrar a existência de um simulador σ_E tal que o recurso ideal $\mathcal{R}$ executado em conjunto com π_A é indistinguível da execução do recurso ideal $\mathcal{S}$ com σ_E. 36

4.1 Medição do operador $U = \bigotimes_{i=1}^{4}$ com um *cat state*. 53
4.2 Correção de erros de Steane. As portas CNOT e as medições são executadas de maneira transversal. 54
4.3 Verificação do *cat state*. 56
4.4 Verificação de estados auxiliares utilizados na correção de erros de Steane ($|\overline{0}\rangle$ em (a), e $|\overline{+}\rangle$ em (b)). 56
4.5 Decodificação de estados auxiliares no caso $t = 1$. 58
4.6 Porta Toffoli aplicada ao estado da base computacional $|xyz\rangle$. 64
4.7 Preparação de $|\overline{0}\rangle$. Todos os CNOTs e medições são feitas transversalmente. 67
4.8 Correção de erros de Steane sem verificação. 67

4.9 (a): Circuito de codificação para $|\overline{0}\rangle$ no código de Steane. O circuito que codifica $|\overline{+}\rangle$ é o mesmo, com a diferença de que alvos e controles são revertidos, e estados $|0\rangle$ são revertidos em $|+\rangle$ e vice-versa. (b): Circuito de codificação para $|\overline{0}\rangle$ no código de Steane. Por construção ,o resultado da medição Z no terceiro qubit dá o autovalor do operador lógico $\overline{Z} = Z_1Z_2Z_3$. O circuito de decodificação $|\overline{+}\rangle$ é o mesmo, com a diferença de que alvos e controles são revertidos e medições na base Z são revertidos em medições na base X e vice-versa. 68
4.10 Encoding (a) and decoding (b) circuits for $|\overline{+}\rangle$ in the $[[7,1,3]]$ code. . . . 70
4.11 (a) Preparação do estado de Shor $|\text{even}_7\rangle$. Os números indicam posições onde os erros introduzidos causam múltiplos erros na saída. (b) Medição da síndrome para o estado de Shor. 70
4.12 Preparação do estado de Toffoli. A caixa rotulada "Paridade" corresponde a medir a entrada na base Z de maneira transversal, seguida de computar a paridade da string de saída. 71
4.13 Dois 1-exRec's sobrepostos com 3 falhas. 76

5.1 (a) Circuito quântico e (b) sua tradução clássica. 80
5.2 Preparação de $|+\rangle$. As componentes dentro do retângulo tracejado são supostas não ter ruído. 80
5.3 Componentes quânticas utilizadas na implementação resistente a vazamentos (esquerda) e sua tradução clássica (direita). 88
5.4 Vazar um bit (codificado no fio acima) é equivalente a introduzir um erro de fase. 88
5.5 Medição na base X. 89
5.6 Porta Toffoli resistente a vazamentos, baseada na implementação da Figura 4.6. Todas as operações são executas em entradas codificadas. As portas de fase são substituídas pela identidade, e os *gadgets* de medição são realizados como explicado na Seção 5.2.1. A entrada $\overline{\Theta}$ é a saída da tradução clássica do circuito da Figura 4.12. 89

Mathematics is the study of analogies between analogies. All science is. Scientists want to show that things that don't look alike are really the same. That is one of their innermost Freudian motivations. In fact, that is what we mean by understanding.

Gian-Carlo Rota

Capítulo 1

Introdução

A história da criptografia abrange vários milênios, mas o paradigma de projetar criptossistemas cuja segurança é provada matematicamente é relativamente novo. Kerckhoffs [Ker83] introduziu a ideia de que um criptossistema seguro não necessita ter um modo de operação secreto—em vez disso, apenas uma pequena parte dele, a *chave*, precisa ser secreta. Antes disso, a maior parte dos criptossistemas eram projetados *ad hoc*: muitas vezes, o conhecimento do projeto era suficiente para quebrá-los. Posteriormente, auxiliados pelo desenvolvimento de computadores digitais, pesquisadores começaram a usar resultados em ciência da computação e teoria da informação para desenvolver novos criptossistemas, como documentado por Diffie e Hellman [DH76].

Assim, a criptografia teórica moderna se preocupa em grande parte em desenvolver esquemas que são comprovadamente seguros sob hipóteses razoáveis. No entanto, a área não está livre de falhas [Bel98]. Um problema que, recentemente, tem atraído bastante atenção é que os modelos normalmente considerados tratam a execução de algoritmos como uma caixa preta. Isto é, a possibilidade de que a implementação física do algoritmo vaze informação (através de um *canal lateral*[1]) é ignorada. Essa informação, se capturada, pode ser usada para ajudar em uma tentativa de quebrar o criptossistema.

O estudo de ataques de canais laterais é, em realidade, mais antigo que a criptografia moderna. Já em 1943, descobriu-se que um telétipo usado pelo exército americano para cifrar mensagens formava um padrão de sinais elétricos que, quando visualizado em um osciloscópio, podia ser utilizado para recuperar a mensagem [Nat07]. Mais recentemente, foram revelados ataques semelhantes a implementações de protocolos criptográficos amplamente usados. Um exemplo é o ataque "Lucky Thirteen" ao protocolo TLS em modo CBC [AP13], que consiste em medir o tempo que se leva para o servidor responder a uma requisição do cliente, e usar essa informação para quebrar

[1] *Side channel*, em inglês.

o algoritmo. Outro ataque recente [GST13] usa criptanálise acústica para atacar a implementação do algoritmo RSA no *software* GnuPG. Os autores conseguiram extrair a chave privada do RSA medindo o ruído acústico produzido pelo computador durante o processo de decifração de um conjunto específico de cifras. Embora o *software* tenha sido atualizado desde a publicação desta descoberta, de forma que o ataque não é mais possível, o exemplo ilustra a importância de se projetar implementações tendo em mente ataques de canais laterais.

Ataques de canal lateral não implicam uma limitação fundamental em criptografia teórica. A possibilidade desses ataques ilustram o fato de que os modelos normalmente considerados na literatura não são sempre suficientemente gerais para os cenários onde criptografia é utilizada na prática. A abordagem teórica do problema é estudar cenários onde ataques de canal lateral são possíveis e projetar *protocolos* que são resistentes a eles. Esse é o foco da área conhecida como "resistência a vazamentos", ou *leakage resilience*, em inglês.

Um número significativo de trabalhos tem se dedicado à resistência a vazamentos (por exemplo, [ISW03, IPSW06, NS12, GR12, SPY13]), garantindo o sucesso da área de um ponto de vista teórico. No entanto, o sucesso na prática tem sido modesto. A raíz do problema é que como é impossível alcançar segurança se o vazamento inclui toda a execução do algoritmo [GR12], é necessário restringir o escopo da informação que pode vazar para o adversário—isto é, impõe-se um *modelo de vazamento*. Em si, essa restrição é natural, uma vez que na prática, adversários recebem uma informação limitada; no entanto, frequentemente é difícil definir precisamente a restrição, isto é, projetar modelos que correspondam a cenários reais.

Recentemente, pesquisadores têm estudado o problema de diminuir a separação entre teoria e prática na defesa contra ataques de canal lateral [SPY$^+$10, SPY13, BFGV12, SMY06]. Comum a esses trabalhos é a caracterização dos modelos de vazamento: todos levam em conta a natureza *física* do vazamento. Esse trabalho fornece uma conexão entre vazamento e a área de informação quântica. Especificamente, o processo de vazamento de informação clássica pode ser visto como um canal quântico ruidoso. O evento de vazamento pode ser visto como um erro no estado da computação. Dessa forma, usando um código corretor de erros quântico, obtém-se proteção contra vazamento "de graça".

Esse resultado sugere uma abordagem nova ao vazamento: usar computação quântica tolerante a falhas. A ideia básica é que todas as ações realizadas na execução de um circuito clássico—assim como ataques ao circuito que usam o vazamento de informação—podem ser descritas usando o formalismo da mecânica quântica. Como veremos, o vazamento do estado físico da computação é equivalente a um *erro de fase*

em um circuito quântico. Como uma computação quântica tolerante a falhas protege contra erros de fase (assim como os convencionais erros de bit), o circuito é automaticamente resistente a vazamentos.

No entanto, esse procedimento requer um circuito tolerante a falhas quântico. Neste trabalho, obtemos um método para construir um circuito resistente a vazamentos *clássico*, modificando um circuito tolerante a falhas *quântico* apropriado. O circuito clássico é construído de maneira a imitar o circuito quântico, de forma que suas execuções não podem ser distinguidas do ponto de vista de um observador clássico. Como consequência, a resistência a vazamentos do circuito clássico é herdada do circuito quântico.

Essa abordagem é similar a provas de segurança de esquemas distribuição de chaves quânticos, como a prova apresentada por Shor e Preskill [SP00]. No trabalho citado, mostra-se que a operação do protocolo real—onde os resultados de medidas de estados quânticos emaranhados são pós-processadas classicamente—pode ser interpretada como uma imitação de um protocolo quântico para distilação de emaranhamento, do qual o protocolo deriva sua segurança. Nesse sentido, o protocolo real é puramente clássico, mas o argumento pelo qual sua segurança é provada vem do protocolo quântico imitado.

1.1 Trabalhos anteriores

A questão de esconder a computação interna de um observador externo está relacionada ao problema de *ofuscação de programas*. Ofuscação pode ser vista como uma "resistência a vazamentos em pior caso", em que o estado interno deve ser protegido mesmo se a execução do programa vaza completamente para o adversário.[2] No entanto, sabe-se que ofuscar programas é em geral impossível [BGI+01]. Logo, para obter qualquer forma de resistência a vazamentos, é necessário restringir de alguma forma o tipo de vazamento.

Isso não é necessariamente um problema, porque na prática o adversário tem suas próprias limitações no acesso a informação proveniente do sistema. Aqui, listamos alguns resultados na área, frisando que a lista de resultados existentes é bem maior (e crescente). Ishai *et al.* [ISW03, IPSW06] considerou adversários que podem aprender os valores de um número limitado de fios do circuito. Micali e Reyzin [MR04] introduziram modelos com a hipótese de que "apenas computação vaza informação", nos quais o vazamento a cada passo da computação depende apenas dos dados utilizados.

[2]Uma conexão explícita entre ofuscação e um modelo específico de vazamento (o modelo "somente computação vaza informação" de Micali e Reyzin[MR04]) foi feita em [BCG+11]; veja também [GR12].

Faust *et al.* [FRR+10] consideram um modelo onde o adversário recebe uma função do estado do circuit (isto é, a saída e todas as computações intermediárias), com a restrição de que a função deve ser computável em AC^0, a família de circuitos contendo apenas portas AND, NOT e OR com *fan-in* ilimitado, grau constante e tamanho polinomial na entrada. Eles também consideram *vazamento ruidoso*, onde todo o estado do circuito vaza, mas o adversário recebe apenas uma versão ruidosa deste estado: cada bit vazado é virado com uma probabilidade fixa. É também comum supor que uma componente—idealmente pequena—do circuito é confiável, isto é, não está sujeita a vazamento [GR10, FRR+10, DF12].

Apesar destes avanços, relacionar modelos de vazamento com vazamento visto em prática ainda é um desafio. Standaert *et al.* [SPY13] apresentam alguns problemas com a hipótese de "vazamento limitado"[3], em que se supõe que o vazamento a cada passo da computação é limitado; veja também [SPY+10].

Em termos gerais, este trabalho é relacionado ao de Smith *et al.* [CGS02], em que técnicas de computação quântica tolerante a falhas foram usadas para desenvolver uma construção para computação quântica de múltiplas partes segura. Dado que técnicas de computação de múltiplas partes são frequentemente utilizadas para resistência a vazamentos (em particular, compartilhamento de segredos [ISW03, DF12, GR10]), a conexão entre tolerância a falhas e resistência a vazamentos talvez não seja tão surpreendente. No entanto, a construção de Smith *et al.* é inerentemente quântica, enquanto nossa construção roda em uma máquina clássica.

1.2 Contribuição desta tese

Esta tese é baseada nos resultados de [LRR14], que foi submetido para o Journal of Cryptology.

Neste trabalho, é estabelecida uma conexão entre computação clássica resistente a vazamentos e computação quântica tolerante a falhas. Mais precisamente, mostramos como métodos de tolerância a falhas podem ser utilizados para construir compiladores de resistência a vazamentos, que transformam um circuito dado em outro circuito (clássico) com a mesma funcionalidade computacional, assim como resistência a vazamentos.

Isso é feito em dois passos. Primeiro, mostramos que circuitos quânticos confiáveis—isto é, circuitos que funcionam corretamente mesmo com presença de ruído—também são resistentes a vazamentos. Depois, utilizamos técnicas em computação quântica

[3] *Bounded leakage*, em inglês.

tolerante a falhas—usadas para implementar circuitos universais confiáveis—para derivar compiladores resistentes a vazamentos.

A relação entre resistência a vazamentos e computação confiável é estabelecida no Capítulo 3. Resistência a vazamentos é definida na Seção 3.3, e computação confiável é definida na Seção 3.6. A conexão entre as duas áreas é concretizada na Seção 3.7.

O ponto de partida para relacionar computação clássica resistente a vazamentos e computação quântica tolerante a falhas é a observação de que qualquer operação lógica *clássica* pode ser vista como uma operação *quântica* executando a mesma ação na chamada base computacional. (Uma introdução ao formalismo da mecânica quântica utilizado neste trabalho é apresentada no Capítulo 2.) Como mostramos na Seção 3.7, qualquer modelo de vazamento pode ser interpretado como um modelo específico de *ruído de fase* no circuito quântico correspondente.

Ruído de fase não é o tipo mais geral de ruído quântico. Não obstante, como mostrado na Seção 3.7 (Teorema 2), se confiabilidade é possível para um modelo de ruído dado—grosso modo, se a correção de erros é executado de forma tão frequente que a informação codificada essencialmente nunca tem erros—, a computação é resistente ao vazamento do modelo de vazamento correspondente.

No entanto, queremos construir circuitos clássicos resistentes a vazamento sem necessitar de um computador quântico para alcançar esse objetivo. Felizmente, a estrutura de certos códigos corretores de erros quânticos é tal que podemos imitar os passos de correção de erros com circuitos clássicos. Seguindo essa abordagem, construimos um compilador de resistência a vazamentos geral executando um procedimento que imita os componentes básicos de tolerância a falhas do esquema apresentado em [AGP05], usando apenas componentes clássicos. A teoria de tolerância a falhas é introduzida no Capítulo 4; as ideias gerais são fornecidas na Seção 4.1, enquanto na Seção 4.2, descrevemos a construção principal de [AGP05].

A Seção 5.1 descreve como transformar em circuitos clássicos certos tipos simples de circuitos quânticos que servem como blocos básicos para circuitos arbitrários. A Seção 5.2 apresenta uma implementação da porta *Toffoli*, uma porta que é universal para computação clássica. Combinando esse resultado com o Corolário 1 na Seção 5.1 dá um compilador resistente a vazamentos.

Como em outros trabalhos, nossa construção supõe a existência de uma parte do circuito que é livre de vazamentos. No entanto, a única tal componente que usamos é uma fonte de bits aleatórios (com distribuição uniforme). Usando a construção, é possível transformar um circuito clássico arbitrário em um circuito que é resistente a vazamento em qualquer modelo para o qual computação quântica confiável é possível—no modelo de ruído correspondente. Um modelo de vazamento que se traduz bem a um

modelo de ruído quântico bem estudado é o vazamento independente, no qual o valor de cada fio no circuito vaza com uma probabilidade fixa. Apesar deste modelo ser potencialmente muito restrito (em particular, a hipótese de independência implica que o adversário não pode escolher os fios cujos dados vazam), sua simplicidade permite uma interpretação clara do cenário quântico como o modelo de erros de fase independente, para o qual várias construções tolerantes a falhas são conhecidas (por exemplo [AGP05]). Esse modelo é usado em nossa construção na Seção 5.2. Além disso, mostrou-se [DDF14] que um compilador que é resistente a vazamentos nesse modelo também é resistente a vazamentos no modelo de *vazamento ruidoso*, em que todos os bits do circuito vazam, mas o adversário recebe uma versão ruidosa desses bits. Uma prova diferente desse resultado é apresentada na Seção 3.4 para o caso particular de "ruído tipo Bernoulli", cada bit recebido pelo adversário é virado com uma probabilidade fixa.

Frisamos que em vez de apresentar um esquema específico resistente a vazamentos, a contribuição deste trabalho procura fornecer uma abordagem original à resistência a vazamentos e conectar essa área da criptografia à área de computação quântica tolerante a falhas. Nosso trabalho mostra como resultados em uma área (por exemplo, teoremas de limiar em tolerância a falhas quântica) podem ser traduzidos para a outra área (por exemplo, cotas no desempenho de comopiladores resistentes a vazamento). No Capítulo 6, discutimos possíveis direções futuras para essa linha de trabalho.

Capítulo 2

Preliminares matemáticos

Esse capítulo introduz o formalismo quântico e seus aspectos que são utilizados neste trabalho. A Seção 2.1 fornece a definição básica de estados quânticos, portas lógicas e canais, e a Seção 2.2 discute em detalhe códigos corretores de erro quânticos (QECCs[1]).

2.1 Qubits e circuitos quânticos

Um sistema quântico pode ser descrito por um espaço de Hilbert complexo. Neste trabalho, lidamos apenas com sistemas de dois níveis. Seja $\mathcal{H}_2$ um espaço de Hilbert complexo bidimensional e seja $\{|0\rangle, |1\rangle\}$ uma base ortonormal, que denotaremos a base padrão ou base computacional. Um elemento arbitrário $|\psi\rangle$ de $\mathcal{H}_2$ pode ser escrito como $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, onde $\alpha, \beta \in \mathbb{C}$. Estados quânticos são representados por vetores unitários, isto é, $|\alpha|^2 + |\beta|^2 = 1$.

Sistemas compostos são dados pelo produto tensorial, isto é, se A, B são espaços de Hilbert complexos, o estado do sistema composto por A e B é um elemento de $A \otimes B$. A base deste espaço consiste em elementos da forma $|a\rangle \otimes |b\rangle$ onde $|a\rangle \in A$ e $|b\rangle \in B$; para manter expressões breves, escrevemos também $|a\rangle \otimes |b\rangle$ como $|a, b\rangle$ ou mesmo $|ab\rangle$, quando não houver ambiguidade.

Neste trabalho, lidaremos frequentemente com estados que são superposições uniformes, isto é, os coeficientes dos elementos da expansão do estado na base considerada são iguais. Quando este é o caso, o coeficiente será omitido. Então, por exemplo, escrevemos o estado $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ simplesmente como $|0\rangle - |1\rangle$.

[1] Quantum error-correcting codes, em inglês.

2.1.1 O grupo de Pauli

Considere a base computacional para $\mathcal{H}_2$ e seja

$$\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.1}$$

X, Y, Z são as chamadas *matrizes de Pauli*. Elas anticomutam uma com a outra. Além disso, uma matriz arbitrária A sobre $\mathcal{H}_2$ pode ser escrita $A = a_0\mathbb{1} + a_1X + a_2Y + a_3Z$ with $a_0, a_1, a_2, a_3 \in \mathbb{C}$. Em outras palavras, as matrizes de Pauli, junto com a identidade $\mathbb{1}$, geram o espaço de matrizes sobre $\mathcal{H}_2$.

De forma semelhante, operadores em $\mathcal{H}_2^{\otimes n}$ também podem ser escritos em termos de matrizes de Pauli. Seja $\mathcal{P}_n$ o conjunto de operadores da forma $i^k \bigotimes_{i=1}^{n} P_i$, onde i é a unidade imaginária, $k \in \mathbb{Z}$ e $P_i \in \{\mathbb{1}, X, Y, Z\}$. O conjunto $\mathcal{P}_n$ é um grupo não abeliano; ele é chamado o *grupo de Pauli* em n qubits. Como $Y = iXZ$, o grupo é gerado por X e Z, a menos de um fator de fase; isto é, $\mathcal{P}_n = \langle i\mathbb{1}, X_1, \ldots, X_n, Z_1, \ldots, Z_n \rangle$, onde X_i denota o operador que atua como X no i-ésimo qubit, e analogamente para Z_i. Uma matriz arbitrária sobre $\mathcal{H}_2^{\otimes n}$ pode ser escrita como uma combinação linear de elementos de $\mathcal{P}_n$.

Diz-se que operadores genéricos A e B comutam se $AB = BA$. Definimos o *comutador* de A e B como $[A, B] := AB - BA$ e o *anticomutador* de A e B comon $\{A, B\} := AB + BA$.

Notamos as seguintes propriedades das matrizes de Pauli. Elas possuem autovalores $+1, -1$, e $X^2 = Y^2 = Z^2 = \mathbb{1}$. Além disso, elas anticomutam uma com a outra: $\{X, Y\} = \{X, Z\} = \{Y, Z\} = 0$.

Note que os autoestados (autovetores) de Z são os elementos da base canônica (representados aqui como vetores de coluna) $(1, 0)^T$ e $(0, 1)^T$. Chamaremos essa base de *base computacional*, com $|0\rangle = (1, 0)^T$ e $|1\rangle = (0, 1)$. Frequentemente também chamaremos essa base de base Z. Outra base importante será a *base de fase*, formada pelos autovetores de X; por esse motivo, também a chamaremos de base X. Usaremos os símbolos $|+\rangle$ e $|-\rangle$ para rotular os elementos desta base. Eles são relacionados à base computacional pelas relações $|+\rangle = |0\rangle + |1\rangle$, $|-\rangle = |0\rangle - |1\rangle$.

2.1.2 Portas lógicas quânticas

X, Y, Z estão entre as portas quânticas mais utilizadas. Outras portas utilizadas nesse trabalho são descritas a seguir.

Hadamard. A operação Hadamard, representada por H, tem a seguinte ação na base computacional:

$$H|0\rangle = |0\rangle + |1\rangle \tag{2.2}$$

$$H|1\rangle = |0\rangle - |1\rangle \tag{2.3}$$

Fixada a base computacional, a matriz H é representada por:

$$H = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.4}$$

Note que X e Z se relacionam pela transformação H: temos $HZH = X$ e $HXH = Z$.

CNOT. A porta CNOT (de *controlled NOT*) é uma operação em dois qubits. O CNOT executa um *bit flip* no segundo qubit (o qubit *alvo*) condicionado no valor do primeiro qubit (o qubit de *controle*). Isto é, temos $\text{CNOT}|a, b\rangle = |a, a \oplus b\rangle$. A matriz para o CNOT é dada por

$$\text{CNOT} = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & X \end{pmatrix} \tag{2.5}$$

(Onde 0 denota a matriz nula 2×2).[2] O CNOT é uma porta clássica, já que ela leva entradas da base computacional a saídas da base computacional. Por uma questão de clareza, frequentemente escreveremos $\text{CNOT}_{i \to j}$ para denotar um CNOT com o i-ésimo como controle e o j-ésimo qubit como alvo. Esta notação simplifica o tratamento nos casos em que a entrada tem mais de 2 qubits, ou quando a ordem de alvo e controle é revertida.

CZ. A porta CZ (também denotada a porta de *controlled phase*) efetua a mesma operação que o CNOT, mas com um *phase flip* em vez de um *bit flip*. Isto é, o CZ atua por $\text{CZ}|a, b\rangle = (-1)^{ab}|a, b\rangle$. A matriz é dada por

$$\text{CZ} = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & Z \end{pmatrix} \tag{2.6}$$

Assim como no caso de X e Z, as portas CZ e CNOT são relacionadas por uma transformação H. Temos $(\mathbb{1} \otimes H)\text{CZ}(\mathbb{1} \otimes H) = \text{CNOT}$ e $(\mathbb{1} \otimes H)\text{CNOT}(\mathbb{1} \otimes H) = \text{CZ}$. Essa relação pode ser facilmente verificada pela representação matricial.

[2]Notamos que a matriz CNOT atua em elementos do espaço formado pelo produto tensorial, de forma que se, por exemplo, $|a\rangle = (a_1, a_2)^T$ e $|b\rangle = (b_1, b_2)^T$, então executar CNOT em $|a, b\rangle$ corresponde a aplicar a matriz (2.5) ao vetor $(a_1(b_1, b_2), a_2(b_1, b_2))^T = (a_1b_1, a_1b_2, a_2b_1, a_2b_2)^T$.

Note que, ao contrário do CNOT, não existe diferença prática entre controle e alvo, embora às vezes seja útil fazer uma distinção entre os dois. Escreveremos CZ_{ij} para denotar um CZ envolvendo o i-ésimo e j-ésimo qubits.

Porta Toffoli. A porta Toffoli, que denotamos por T, tem três qubits como entradas e atua na base computacional por $T|a,b,c\rangle = |a,b,c \oplus ab\rangle$. Esta porta é também chamada a porta *controlled-controlled NOT*, já que ela atua como um CNOT, mas com duas entradas como controle. Sua representação matricial é

$$T = \begin{pmatrix} \mathbb{1} & 0 & 0 & 0 \\ 0 & \mathbb{1} & 0 & 0 \\ 0 & 0 & \mathbb{1} & 0 \\ 0 & 0 & 0 & X \end{pmatrix} \tag{2.7}$$

Note que, assim como o CNOT, a porta Toffoli é uma porta clássica: ela toma entradas na base computacional a saídas na base computacional. Escrevemos $T_{ij\to k}$ para denotar uma porta Toffoli com entradas i e j como controle e a entrada k como alvo.

A notação de circuito para as portas utilizadas nessa trabalho é mostrada na Figura 2.1. Note que como $Y = iXZ$, a porta Y não é representada na figura. Além disso, note que escrevemos CZ de maneira assimétrica, mas não existe diferença prática na escolha do fio na qual o fator de fase é adicionado.

2.1.3 Grupo de Clifford e universalidade quântica

O grupo de Clifford $\mathcal{C}_n$ é o conjunto de operações unitárias que mantém o grupo de Pauli em n qubits, $\mathcal{P}_n$, invariante sob conjugação:

$$\mathcal{C}_n = \{C \in U(2^n) : \forall g \in \mathcal{P}_n, CgC^\dagger \in \mathcal{P}_n\}/U(1) \tag{2.8}$$

Em outras palavras, o grupo de Clifford é o *normalizador* do grupo de Pauli, módulo fatores de fase.[3] Demonstrou-se [CRSS97] que $\mathcal{C}_n$ é gerado por H, CNOT e a *porta fase* P dada por $P|j\rangle = i^j|j\rangle$.

A principal importância deste grupo se deve ao fato de que, como um elemento de um grupo de Clifford (uma *porta de Clifford*) leva operadores de Pauli a operadores de Pauli, circuitos que contêm apenas portas de Clifford possuem uma descrição simples, em termos de sua ação em operadores de Pauli. De fato, estes circuitos podem ser

[3] Devido à relação $e^{i\phi}Cg(e^{i\phi}C)^\dagger = CgC^\dagger$.

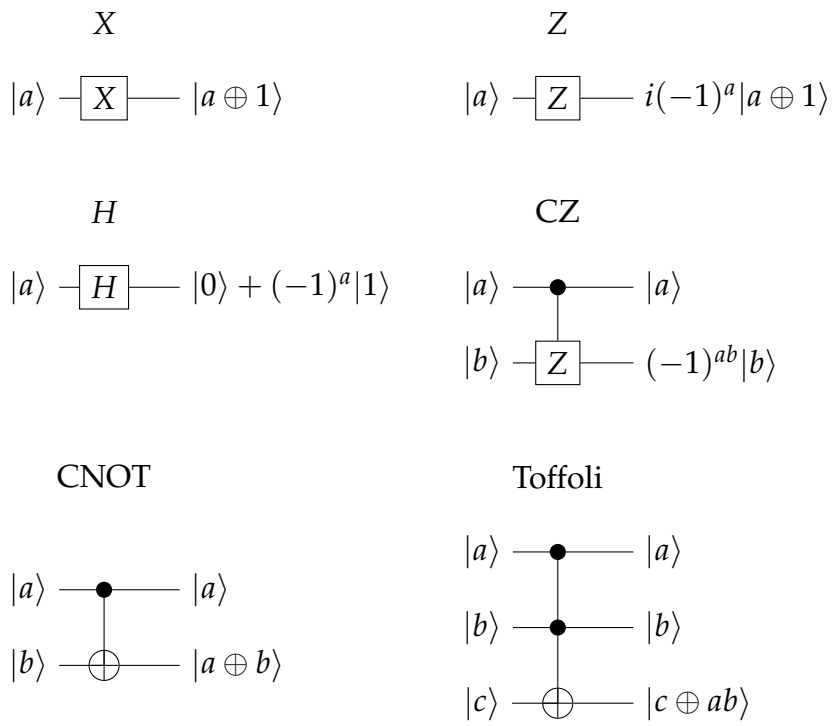


Figura 2.1: Portas quânticas utilizadas neste trabalho.

simulados eficientemente em um computer clássico; esse é o teorema de Gottesman-Knill [Got98a].

O grupo de Clifford *não* é universal para computação quântica—isto é, existem operações unitárias que não podem ser implementadas com circuitos apenas com portas de Clifford. De fato, se isto fosse possível não haveria necessidade para computadores quânticos, uma vez que circuitos de portas de Clifford podem ser simuladas eficientemente num computador clássico. Curiosamente, adicionar uma única porta que está fora do grupo de Clifford (por exemplo, a porta Toffoli) ao conjunto o torna universal [NRS01, Theorem 5.6].

2.1.4 Universalidade clássica e circuitos reversíveis

A porta Toffoli também possui a propriedade importante de ser universal para computação *clássica* [Tof80]. Além disso, a computação pode ser feita de maneira reversível. Isto é, para qualquer função booleana f em n entradas $x_1, \ldots, x_n$, existe um circuito composto unicamente de portas Toffoli, que toma como entradas $x_1, \ldots, x_n$ assim como um conjunto de bits adicionais fixados em 0 ou 1, e tem como saída $x_1, \ldots, x_n, f(x_1, \ldots, x_n)$, acompanhado de alguns bits adicionais fixos.

Para provar a afirmação de que a porta Toffoli é universal, note que a porta Toffoli pode ser usada para implementar a porta NAND, dada por $\mathrm{NAND}(a, b) = ab \oplus 1$, e a

porta FAN-OUT, que duplica a entrada; isto é, ela copia a entrada em um registrador extra. A porta NAND pode ser implementada a partir de uma porta Toffoli, fixando a terceira entrada desta porta em 1. A porta FAN-OUT pode ser implementada fixando uma das entradas de controle da porta Toffoli em 1, e a entrada alvo em 0. Assim, a porta duplica a entrada da outra porta de controle. Pode-se mostrar [Pei80] que o conjunto de portas $\{\text{NAND}, \text{FAN-OUT}\}$ é universal, logo o mesmo é válido para a porta Toffoli. Mas também é possível provar que a porta é universal para computação (reversível) diretamente, como mostraremos abaixo. O argumento é baseado em [Pre98, Chapter 6].

Primeiro, note que uma função reversível com n bits de entrada é uma *permutação* do conjunto de strings binárias. Por exemplo, a porta CNOT com a primeira entrada como controle pode ser vista como uma permutaÇào que permuta as strings 10 e 11 e mantém 00 e 01 fixas. Esse é um exemplo de uma *transposição*: uma permutação onde apenas duas strings são permutadas. A seguir, usaremos a notação $(x\ y)$ para denotar uma transposição das strings x e y. Assim, por exemplo, CNOT é a permutação $(10\ 11)$.

Portanto, para mostrar que a porta Toffoli é universal é suficiente, para qualquer permutação em strings de n bits, construir um circuito envolvendo apenas a porta Toffoli que implemente a permutação, para n arbitrário. Mostraremos que isto pode ser feito para transposições; como qualquer permutação é um produto de transposições, é provada a universalidade.

A porta Toffoli com os dois primeiros bits como controle pode ser escrita como a transposição $(110\ 111)$. Chamaremos uma transposição deste tipo uma transposição de um bit, uma vez que apenas o terceiro bit é modificado. Outras transposições podem ser realizadas aplicando portas X a entradas e saídas de controle. Por exemplo $X_2 T_{12\to 3} X_2$ é a transposição $(010\ 011)$, como mostrado na Figura 2.2. É fácil ver que com este procedimento é possível executar qualquer transposição no terceiro bit, e consequentemente em qualquer bit (basta trocar a posição do bit alvo). Devido ao fato de que qualquer transposição é um produto de transposições de um bit, e a porta X pode ser implementada com uma porta Toffoli (fixando as entradas de controle em 1), é possível realizar qualquer permutação em 3 bits usando apenas portas Toffoli.

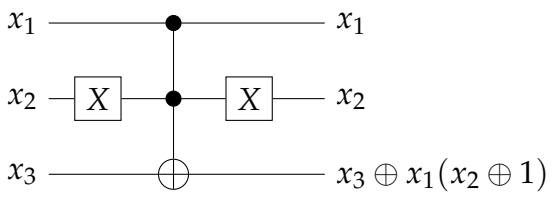


Figura 2.2: Implementação da transposição $(010\ 011)$ com portas Toffoli e X.

Essa abordagem pode ser generalizada para $n > 3$. A ideia é considerar uma porta Toffoli "generalizada" $T^{(n)}(x_1, \ldots, x_n) := x_n \oplus \prod_{i=1}^{n-1} x_i$, que, usando um argumento análogo ao apresentado no parágrafo anterior, pode ser utilizado para implementar permutações arbitrárias em n bits. Adicionalmente, como será mostrado a seguir, $T^{(n)}$ com $n > 3$ pode ser construído usando portas Toffoli (i.e., $T^{(3)}$). Este resultado pode ser contrastado com o fato de que *não* é possível construir uma porta Toffoli a partir de CNOTs ($T^{(2)}$) Isto pode ser verificado a partir da constatação de que circuitos envolvendo apenas CNOTs são lineares em todas as suas entradas, enquanto a porta Toffoli é não-linear em suas entradas de controle.

A construção é feita de maneira indutiva, usando portas Toffoli generalizadas de ordem mais baixa para implementar uma porta de ordem mais alta. A construção para $T^{(4)}$ é mostrada na Figura 2.3. Note que uma entrada extra fixa é usada. Para manter a reversibilidade, ela é retornada ao seu valor original ao final do circuito. Este bit é usado para armazenar o bit de controle "intermediário" x_1x_2 formado pelos dois primeiros bits de controle, que são posteriormente combinados com o terceiro bit de controle na segunda porta Toffoli.

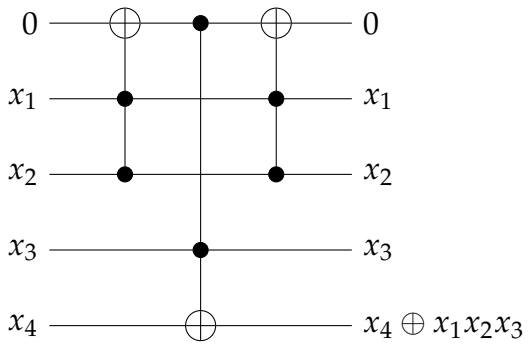


Figura 2.3: $T^{(4)}$ a partir de $T^{(3)}$.

Esta construção é facilmente generalizada àquela da Figura 2.4. Aqui, a porta $T^{(n-1)}$, já construída a partir do processo indutivo iniciado na Figura 2.3, é utilizada, assim como $T^{(3)}$.

Eficiência. A construção apresentada acima usa um número constante de entradas adicionais fixas: é necessário fixar 1 bit em 0 para $T^{(n-1)}$ e 2 bits em 1 para as portas X—note que, como os bits de saída correspondentes também são fixos, eles podem ser reutilizados. Portanto, a implementação impõe um *overhead* aditivo constante em espaço. Além disso, o circuito usa tempo linear no número de bits n. A construção é, portanto, bastante eficiente.

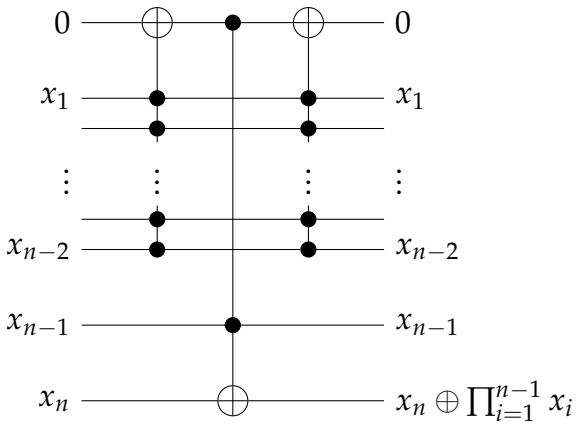


Figura 2.4: $T^{(n)}$ a partir de $T^{(n-1)}$ e $T^{(3)}$.

Como fica a implementação de circuitos irreversíveis? Existe um procedimento genérico (porém ineficiente) para fazê-lo. Como notado anteriormente, o conjunto $\{\text{NAND}, \text{FAN-OUT}\}$ é universal, e ambas as portas podem ser implementadas com uma porta Toffoli cada. Podemos proceder da seguinte maneira: escrevemos o circuito usando apenas NAND e FAN-OUT, e então substituímos estas portas por portas Toffoli. Ao fim da computação, um registrador armazenará a saída do circuito original, mas haverá também várias saídas "lixo"—que não são saídas no circuito original, mas não possuem valor fixo, e, portanto, não podem ser eliminadas (de outra maneira, o circuito não seria reversível). Para tornar o circuito reversível, podemos simplesmente executar o circuito no sentido inversa—isto funciona porque como a porta Toffoli é uma transposição, ela é sua própria inversa.

Essa construção é eficiente em tempo, uma vez que a execução dura o dobro de passos do circuito original. Isto é, se o circuito original leva tempo $O(T)$, o mesmo vale para a simulação reversível. No entanto, ela é bastante ineficiente em espaço: cada porta Toffoli adiciona um *overhead* constante, e há pelo menos uma porta Toffoli por passo na computação. Assim, se o circuito original usa tempo $O(T)$ e espaço $O(S)$, a simulação usa espaço $O(S + T)$.

No entanto, existe uma forma mais eficiente de se fazer a simulação. Usando um *pebble game* [KAI79], Bennett [Ben89] (veja também [Pre98]) mostrou que para qualquer $\varepsilon > 0$, é possível executar a simulação em tempo $O(T^{1+\varepsilon})$ e espaço $O(S \log T)$.

2.1.5 Operadores densidade e canais quânticos

Sistemas cujo estado não é completamente conhecido são descritos por *operadores densidade*. Suponha que um sistema se encontra no estado $|\psi_i\rangle$ com probabilidade p_i. Definimos o operador densidade ρ para o sistema como

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|, \tag{2.9}$$

onde $|\psi_i\rangle\langle\psi_j|$ é o operador que leva $|\phi\rangle$ a $\langle\psi_j|\phi\rangle|\psi_i\rangle$.

Em geral, um operador densidade em $\mathcal{H}$ é um operador positivo com traço unitário. Denotamos o conjunto de tais operadores $\mathcal{B}(\mathcal{H})$. Estados completamente conhecidos, isto é, estados da forma $\rho = |\psi\rangle\langle\psi|$, são chamados *estados puros*, enquanto os outros estados são chamados *estados mistos*.

Operadores densidade são úteis para descrever subssistemas de um sistema maior. Relembramos que o *traço* de um operador densidade $\rho \in \mathcal{B}(\mathcal{H})$ é dado por

$$\mathrm{tr}(\rho) := \sum_i \langle i|\rho|i\rangle, \tag{2.10}$$

onde $\{|i\rangle\}$ é uma base ortonormal de $\mathcal{H}$. Para um operador densidade em um sistema composto $\mathcal{H}_A \otimes \mathcal{H}_B$, também é possível realizar um *traço parcial*, definido da seguinte forma. Se $|\psi_i\rangle, |\psi_{i'}\rangle \in \mathcal{H}_A, |\phi_k\rangle, |\phi_{k'}\rangle \in \mathcal{H}_B$, então o traço parcial tr_B é definido como

$$\mathrm{tr}_B(|\psi_i\rangle\langle\psi_{i'}| \otimes |\phi_k\rangle\langle\phi_{k'}|) := \langle\phi_k|\phi_{k'}\rangle|\psi_i\rangle\langle\psi_{i'}|. \tag{2.11}$$

tr_A é definido de maneira análoga. Por linearidade, a expressão (2.11) também define o traço parcial de um operador densidade arbitrário $\rho_{AB} \in \mathcal{B}(\mathcal{H}_A \otimes \mathcal{H}_B)$. Assim, um subssistema é obtido a partir do traço parcial dos subssistemas complementares.

Essa noção também pode ser utilizada para definir a *purificação* de um operador densidade. Uma purificação de um operador densidade ρ_A é um estado puro $|\psi\rangle_{AB}$ tal que $\mathrm{tr}_B |\psi\rangle\langle\psi| = \rho_A$. Uma purificação sempre existe; um procedimento para obter uma para um operador genérico é dada pela decomposição de Schmidt [NC00, Section 2.5].

Quais operações em estados quânticos são permitidas? Uma condição clara é que elas devem levar operadores densidade a operadores densidade, o que significa que elas precisam ser positivas e preservar o traço, quando aplicadas a operadores densidade. Podemos escrever que uma operação quântica $\mathcal{E}$ satisfaz $\mathcal{E}(\rho) \geq 0$ se $\rho \geq 0$ e $\mathrm{tr}\,\mathcal{E}(\rho) = 1$ se $\mathrm{tr}\,\rho = 1$. Existe uma condição adicional: $\mathcal{E}$ deve ser positiva mesmo quando aplicada a uma purificação de ρ. Neste caso, diz-se que $\mathcal{E}$ é completamente positiva.

Assim, operações quânticas são mapas completamente positivas que preservam o traço. A sigla *CPTP*—de *completely positive, trace-preserving* é comumente usada. Estas funções podem ser escritas na seguinte forma, conhecida como a *representação de Kraus*:

$$\mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger \tag{2.12}$$

onde $\rho, \{E_i\} \in \mathcal{B}(\mathcal{H})$. Como $\mathcal{E}$ preserva o traço, devemos ter $\sum_i E_i^\dagger E_i = \mathbb{1}$.

Assim como operadores densidade podem ser vistos como estados puros em um sistema maior, um mapa CPTP genérico pode ser visto como uma operação *unitária* em um sistema maior. Esta operação é dada pelo *teorema da dilatação de Stinespring* [Sti55] (veja também [NC00, Section 8.2]). O teorema afirma que para qualquer mapa CPTP $\mathcal{E}\colon \mathcal{B}(\mathcal{H}_A) \to \mathcal{B}(\mathcal{H}_B)$, existe um espaço de Hilbert $\mathcal{H}_E$ (o *ambiente*) e uma operação unitária $U\colon \mathcal{H}_A \otimes \mathcal{H}_E \to \mathcal{H}_B \otimes \mathcal{H}_E$ tal que

$$\mathcal{E}(\rho) = \mathrm{tr}_E\, U(\rho_A \otimes |0\rangle\langle 0|_E)U^*. \tag{2.13}$$

A dilatação de Stinespring é particularmente útil na análise de canais quânticos. Ela mostra que um canal quântico ruidoso pode ser visto como uma evolução unitária em um sistema maior. Em particular, de um ponto de vista criptográfico, qualquer informação que vaza a um adversário pode ser obtida do subssistema do ambiente. Esta ideia será usada no Capítulo 3 no contexto de circuitos confiáveis e resistência a vazamento.

2.1.6 Medições quânticas

Uma noção importante no formalismo usado neste trabalho é a de uma *medição*. Medições são comumente usadas para descrever resultados de experimentos; aqui, o foco será no aspecto de uma medição como uma interface entre sistemas quânticos e clássicos.

A forma mais comum é uma medição na base computacional; quando nos referirmos simplesmente a uma medição, este é o tipo pretendido. Um qubit arbitrário $|\psi\rangle$ pode ser escrito como uma combinação linear dos estados da base computacional $|0\rangle, |1\rangle$. Isto é, podemos escrever $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, onde $|\alpha|^2 + |\beta^2| = 1$. Depois de medir $|\psi\rangle$, o estado colapsa em $|0\rangle$, com probabilidade $|\alpha|^2$, ou $|1\rangle$, com probabilidade $|\beta|^2$. Cada estado pós-medição possível é associado a um possível resultado da medição. Por conveniência, rotulamos os resultados como os autovalores do estado correspondente. Assim, se o resultado é $+1$, então o estado pós-medição é $|0\rangle$; se é -1, então o estado é $|1\rangle$.

Não há nada especial na base computacional: poderíamos também realizar uma medição na base de fase $\{|+\rangle, |-\rangle\}$. Como temos $|\psi\rangle = |\alpha+\beta|^2/2|+\rangle + |\alpha-\beta|^2/2|-\rangle$, medir $|\psi\rangle$ na base de fase colapsa o estado em $|+\rangle$, com probabilidade $|\alpha+\beta|^2$, ou $|-\rangle$, com probabilidade $|\alpha-\beta|^2$). Assim como no caso da base computacional, $|+\rangle$ corresponde ao resultado $+1$, e $|-\rangle$ corresponde a -1.

Em geral, medições seguem a chamada *regra de Born*. Uma medição é dada por um conjunto de operadores $\{E_i\}_i$ que atuam no espaço de Hilbert $\mathcal{H}_2$. Os operadores satisfazem $\sum_i E_i^\dagger E_i = \mathbb{1}$. O resultado i da medição do estado $|\psi\rangle$ ocorrem com probabilidade p_i, dada por

$$p_i = \langle\psi|E_i^\dagger E_i|\psi\rangle \tag{2.14}$$

e o estado pós-medição é dado por

$$|\psi_i\rangle := \frac{E_i|\psi\rangle}{\sqrt{p_i}}. \tag{2.15}$$

Para uma medição na base computacional, temos $E_{-1} = |0\rangle\langle 0|, E_{+1} = |1\rangle\langle 1|$. Para um medição na base de fase, temos $E_{-1} = |+\rangle\langle +|, E_{+1} = |+\rangle\langle +|$. Note que medições podem também ser aplicadas a sistemas com múltiplos qubits.

É fácil generalizar a regra de Born a estados mistos. Após medir um operador densidade ρ, a probabilidade de obter o resultado i é dada por $p_i = \mathrm{tr}(E_i^\dagger E_i \rho)$, e o estado pós-medição é dado por

$$\rho_i := \frac{E_i \rho E_i^\dagger}{p_i}. \tag{2.16}$$

Para mais detalhes, consulte [NC00, Section 2.4].

Medição de elementos de $\mathcal{P}_n$. Neste trabalho, usaremos com frequência a operação de medir um operador no grupo de Pauli. A medição de um operador $P \in \mathcal{P}_n$ consiste em aplicar a medição dada por

$$E_+ := \frac{\mathbb{1}+P}{2}, \qquad E_- := \frac{\mathbb{1}-P}{2} \tag{2.17}$$

Note que como $P^\dagger P = P^2 = \mathbb{1}$, temos $E_+^\dagger = E_+, E_-^\dagger = E_-$, isto é, E_+ e E_- são projeções, e $E_+^\dagger E_+ + E_-^\dagger E_- = \mathbb{1}$. Isto mostra que a medição é válida. Note também que o estado pós-medição é um autoestado de P, com autovalor ± 1, cujo sinal depende do resultado da medição.

Finalmente, notamos que medições quânticas podem ser modeladas como um tipo de operação similar a uma operação quântica, com a diferença de que a operação neces-

sita apenas não aumentar o traço (lembre que operações quânticas preservam o traço). Referimos a [NC00, Section 8.2.3] para mais detalhes.

2.2 Códigos corretores de erro quânticos

Considere um sistema S sujeito a ruído do ambiente E. O significado disto é que o sistema evolui da seguinte forma:

$$|\psi\rangle_S|0\rangle_E \to \sum_i E_i|\psi\rangle_A|e_i\rangle_E \tag{2.18}$$

onde os estados $|e_i\rangle$ não são necessariamente ortogonais, e $\{E_i\}_i$ é um conjunto de operadores de Pauli linearmente independentes. Chamaremos cada um deles de um *erro de Pauli*. Diremos que um erro de Pauli E_i possui peso t se ele atua de forma não-trivial (isto é, diferente da identidade) em no máximo t qubits.

Antes de introduzir códigos corretores de erro quânticos (QECCs) formalmente, é útil considerar um exemplo ilustrativo. Primeiro, podemos nos perguntar: corrigir erros quânticos é mesmo possível? Esta questão é razoável porque as ideias básicas utilizadas em correção de erros clássicos resistem a uma tradução clara para o caso quântico. Considere o código clássico mais simples: o código de repetição. Um bit é codificado: 0 é codificado em 000, e 1 em 111. Este código pode corrigir qualquer erro de um único bit: se um bit é corrompido, os dados codificados ainda podem ser recuperados, tomando um voto majoritário sobre a string—de forma que os bits restantes "ganham" a votação.

Podemos tentar generalizar essa ideia ao caso quântico, com um código em que $|\psi\rangle$ é codificado em $|\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle$. No entanto, sabe-se que tal código é impossível, devido ao *no-cloning theorem*: não existe uma operação que leva um estado arbitrário $|\psi\rangle$ a $|\psi\rangle \otimes |\psi\rangle$.

Não obstante, a ideia de um código de repetição é útil no caso quântico. O primeiro passo é reconhecer que, uma vez que erros genéricos podem ser escritos como uma superposição de erros de Pauli—como notado na Seção 2.1.1—podemos nos focar em erros X e Z, também conhecidos como erros de bit e erros de fase, respectivamente.[4]

Se apenas erros de bit ocorrem, eles podem ser corrigidos aplicando a mesma estratégia usada em correção de erros clássicos: codificamos $|0\rangle$ em $|000\rangle$ e $|1\rangle$ em $|111\rangle$. Por outro lado, se temos apenas erros de fase, podemos novamenteaplicar a mesma estratégia, mas usando a base complementar (base de fase) $\{|+\rangle := |0\rangle + |1\rangle, |-\rangle :=$

[4]Já que $Y = iXZ$, um erro Y corresponde a um erro de bit e um erro de fase ocorrendo simultaneamente.

$|0\rangle - |1\rangle\}$. Esta base pode ser obtida a partir da base computacional $\{|0\rangle, |1\rangle\}$ aplicando o operador Hadamard. Devido à relação $HZH^\dagger = X$, se aplicarmos o mapa

$$|+\rangle \to |+++\rangle \tag{2.19}$$

$$|-\rangle \to |---\rangle, \tag{2.20}$$

o código é resistente a um único erro de fase, exatamente como no caso de erros de bit.

No entanto, para códigos quânticos, é necessário proteger contra ambos tipos de erros. Neste caso particular, é suficiente *concatenar* os dois esquemas: isto é, codificamos o qubit usando um código, e posteriormente codificamos os qubits físicos do estado obtido usando o outro código. Mais precisamente, temos as seguintes codificações:

$$|\bar{0}\rangle = (|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \tag{2.21}$$

$$|\bar{1}\rangle = (|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle), \tag{2.22}$$

onde $|\bar{a}\rangle$ representa a codificação do qubit $|a\rangle$. Este código é conhecido como o *código de Shor*, introduzido em [Sho95]. Pode-se verificar que o código protege contra quaisquer erros simples (isto é, em uma só posição) de bit ou de fase. Notamos também que não há violação do *no-cloning theorem*: a codificação não requer a replicação de estados arbitrários. De fato, a codificação de um estado arbitrário é simplesmente uma combinação linear de $|\bar{0}\rangle$ e $|\bar{1}\rangle$, isto é, temos $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \to \alpha|\bar{0}\rangle + \beta|\bar{1}\rangle$.

Agora podemos introduzir a teoria de correção de erros quânticos em mais detalhe. Um código corretor de erros quânticos (QECC) é uma função de $\mathcal{H}_2^k$ a $\mathcal{H}_2^n$, onde $n > k$. Frequentemente, vamos nos referir ao código em termos da imagem desta função; nesta representação, o código é um subespaço de $\mathcal{H}_2^n$, que chamaremos de *espaço do código*. Elementos deste espaço são chamados *palavras de código*. Um erro de Pauli E_i é corrigível se existe uma operação de recuperação R em $A \otimes S$ (onde A é um sistema auxiliar) tal que para todo $|s\rangle$ no espaço do código, temos

$$R|0\rangle \otimes E_i|s\rangle = |i\rangle|s\rangle. \tag{2.23}$$

Devido ao fato de que QECCs tem estrutura de subespaço, um código que corrige erros de Pauli E_1 e E_2 também corrige a superposição $E_1 + E_2$. Lembrando que um erro genérico pode ser sempre escrito como uma superposição de erros de Pauli, é suficiente projetar QECCs que corrigem os erros de Pauli. Normalmente consideraremos códigos que corrigem todos os erros de Pauli com peso menor ou igual a um número fixo t. Logo, esse argumento implica que tal código corrige todos os erros com peso menor ou igual a t.

2.2.1 Propriedades de QECCs

Teorema 1. *Dado um conjunto de erros $\mathcal{E} = \{E_i\}_i$, a condição a seguir é necessária e suficiente para a existência de uma operação de recuperação: para todos $|\alpha\rangle, |\beta\rangle \in \mathcal{C}$, temos*

$$\langle\alpha|E_i^\dagger E_j|\beta\rangle = A_{ij}\langle\alpha|\beta\rangle \tag{2.24}$$

para algum $A_{ij} \in \mathbb{C}$.

Demonstração. Por (2.23), temos

$$\langle\alpha|E_i^\dagger E_j|\beta\rangle = \langle\alpha|\langle 0|R^\dagger R|0\rangle E_i^\dagger E_j|\beta\rangle = \langle i|j\rangle\langle\alpha|\beta\rangle, \tag{2.25}$$

onde a primeira igualdade segue do fato de que R é unitária. Logo (2.24) é uma condição necessária. Para provar a recíproca, notamos primeiro que, por (2.24), A pode ser diagonalizada, isto é, existe uma matriz unitária K e uma matriz diagonal Λ com $\Lambda = K^\dagger AK$. Agora, seja $F_j = \sum_i K_{ij}E_i$. Temos

$$\langle\alpha|F_l^\dagger F_m|\beta\rangle = \sum_{ij} K_{li}K_{jm}\langle\alpha|E_i^\dagger E_j|\beta\rangle \tag{2.26}$$

$$= \sum_{ij} K_{li}^\dagger K_{jm}A_{ij}\delta_{\alpha\beta} \tag{2.27}$$

$$= \lambda_l\delta_{lm}\delta_{\alpha\beta}, \tag{2.28}$$

onde $\lambda_l = \Lambda_{ll}$. Logo, $\{F_j\}$ define um novo conjunto de erros equivalente a $\{E_i\}$, com a diferença de que, agora, $\langle\alpha|F_l^\dagger F_m|\alpha\rangle = 0$ se $l \neq m$. A princípio, também é possível que $\lambda_l = 0$. A seguir, excluiremos a possibilidade de tais F_l, porque eles satisfazem $F_l|\alpha\rangle = 0$ para todo $|\alpha\rangle$, isto é, eles nunca podem ocorrer.

Seja $P = \sum_\alpha |\alpha\rangle\langle\alpha|$ o operador que projeta todo estado no espaço do código; como qualquer matriz, F_jP possui a decomposição polar

$$F_jP = U_j\sqrt{PF_j^\dagger F_jP} = \sqrt{\lambda_j}U_jP. \tag{2.29}$$

para algum unitário U_j.

Defina agora $P_j = U_jPU_j^\dagger$. Com a adição da projeção complementar $P_0 = \mathbb{1} - \sum_j P_j$, o conjunto $\{P_j\}$ define uma medição. Agora, podemos definir a operação de recuperação

$$R|0\rangle \otimes F|\alpha\rangle = \sum_j \frac{1}{\sqrt{\lambda_j}}|j\rangle U_j^\dagger P_j|\alpha\rangle = PU_j^\dagger|\alpha\rangle \tag{2.30}$$

para um erro arbitrário F. Suponha agora que o erro F_i foi aplicado a um estado $|\alpha\rangle$ no espaço do código. Temos

$$R|0\rangle \otimes F_i|\alpha\rangle = \sum_{j\beta} \frac{1}{\lambda_j}|j\rangle|\beta\rangle\langle\beta|F_j^\dagger F_i|\alpha\rangle U_j P \tag{2.31}$$

$$= \sum_{j\beta} |j\rangle|\beta\rangle\delta_{ij}\delta_{\alpha\beta} \tag{2.32}$$

$$= |i\rangle|\alpha\rangle. \tag{2.33}$$

Logo, a condição (2.23) é satisfeita. □

Na demonstração, excluimos F_i's que aniquilam palavras de código. Erros desse tipo ocorrem quando a matriz A tem autovalores repetidos, ou—o que é equivalente—quando há dois erros com a mesma ação nas palavras de código. Voltando ao exemplo do código de Shor, os erros Z_1, Z_2 e Z_3 atuam da mesma maneira nas palavras de código. Dizemos que um código com esta propriedade é *degenerado*. Note que degeneração também depende do conjunto de erros considerado. Note também que o fenômeno é puramente quântico: num código clássico, para uma palavra de código c e erros e_1, e_2 com $e_1 \neq e_2$, temos sempre $c + e_1 \neq c + e_2$.

Suponha agora que $\mathcal{E}$ é o conjunto de erros de Pauli com peso menor ou igual a t. Neste caso, para qualquer par de erros E_i, E_j, o erro $E := E_i^\dagger E_j$ possui peso de no máximo $2t$, e, por (2.24), devemos ter

$$\langle\alpha|E|\alpha\rangle = a_E \tag{2.34}$$

onde a_E depende apenas de E. Esta condição implica que é impossível distinguir palavras de código usando medições em no máximo $2t$ qubits. Por analogia a códigos clássicos, definimos a *distância* de um código como o menor peso para $E \in \mathcal{P}_n$ tal que a condição (2.34) não é satisfeita. Pela discussão acima, um código que corrige t erros tem distância de pelo menos $2t + 1$. Dizemos também que um código que codifica k qubits em n qubits com distância d é um $[[n, k, d]]$-código.

2.2.2 Códigos estabilizadores

Códigos quânticos são comumente descritos em termos do formalismo de estabilizadores, introduzido por Gottesman [Got97].

Seja S um subgrupo abeliano do grupo de Pauli em n qubits $\mathcal{P}_n$ que não contenha $-\mathbb{1}$. O conjunto $\mathcal{C}$ definido por

$$\mathcal{C} = \{|\psi\rangle \in \mathcal{H}_2^n : \forall G_i \in S,\ G_i|\psi\rangle = |\psi\rangle\} \tag{2.35}$$

é um código quântico. Códigos deste tipo são chamados *códigos estabilizadores*.

A ideia é que pares de elementos de $\mathcal{P}_n$ comutam ou anticomutam, e erros podem ser diagnosticados a partir de uma medição dos elementos geradores de S—chamados geradores do estabilizador ou simplesmente *estabilizadores*. Mais precisamente, se um erro E anticomuta com algum $G \in S$, e E é aplicado a uma palavra código $|\psi\rangle$, então temos $E|\psi\rangle = EG|\psi\rangle = -GE|\psi\rangle$. Em outras palavras, $E|\psi\rangle$ é um autoestado de G com autovalor -1. Por outro lado, se nenhum erro ocorreu, então a medição de qualquer estabilizador dará resultado $+1$. Assim, para detectar se um erro E ocorreu, basta medir G. Dessa maneira, medir estabilizadores é análogo a computar a síndrome em um código corretor de erros clássicos. De fato, frequentemente vamos nos referir ao procedimento de medir os estabilizadores como a *medição de síndrome*.

O código de Shor é um código estabilizador cujo grupo estabilizador é gerado pelo seguinte conjunto:

$$Z_1Z_2 \quad Z_1Z_3 \quad Z_4Z_5 \quad Z_4Z_6 \quad Z_7Z_8 \quad Z_7Z_9 \tag{2.36}$$

$$X_1X_2X_3X_4X_5X_6 \quad X_1X_2X_3X_7X_8X_9 \tag{2.37}$$

É fácil verificar que as palavras código em (2.21) são autoestados destes operadores. Os operadores em (2.36) detectam erros de fase, enquanto os operadores em (2.37) detectam erros de bit. Notamos também que há múltiplos erros cujas síndromes são idênticas. Por exemplo, tanto Z_4 quanto Z_5 anticomutam com $X_1X_2X_3X_4X_5X_6$, e comutam com os outros estabilizadores. Isto é devido ao fato de que o código de Shor é degenerado, como notamos em 2.2.1. Todavia, a recuperação ainda é possível, devido ao fato de que Z_4Z_5 é um estabilizador. Isso implica que não há necessidade de se distinguir entre Z_4 e Z_5: se, por exemplo, o erro Z_4 ocorre, então a recuperação também pode ser feita aplicando Z_5.

Outras propriedades de códigos estabilizadores. O código de Shor é um $[[9,1,3]]$-código com 8 estabilizadores. Em geral, um $[[n,k,d]]$-código estabilizador possui $n-k$ estabilizadores. Isto vem do fato de que adicionar um estabilizador ao conjunto de geradores reduz à metade a dimensão do espaço do código, uma vez que se adiciona a restrição de que palavras de código estejam no autoespaço do estabilizador com autovalor $+1$.

É possível que um operador não-trivial fora do grupo estabilizador comute com todos os estabilizadores. Erros desse tipo não podem ser detectados, uma vez que eles atuam de forma não trivial em palavras de código, sem que a sua imagem deixe o espaço do código. Para um subconjunto geral S de um grupo, o conjunto de elementos que comutam com todos os elementos de S é chamado o *centralizador* de S, e denotado por $C(S)$. Se S é o estabilizador, então erros que não podem ser detectados são gerados por operadores que estão no centralizador mas não no estabilizador. A *distância* de um código estabilizador é o peso do operador de peso mínimo em $C(S)\backslash S$.

No contexto de QECCs, é mais comum trabalhar com o *normalizador* $N(S)$. Para um grupo arbitrário G, o normalizador $N(G)$ consiste em todos os operadores que mantêm o grupo invariante sob conjugação, isto é, todos os operadores tais que $\{ngn^\dagger : g \in G\} = G$.

Para o grupo estabilizador, as duas noções são equivalentes, isto é, $N(S) = C(S)$. Este fato pode ser mostrado da seguinte maneira. Primeiro, é óbvio que $C(S) \subseteq N(S)$. Para mostrar que $N(S) \subseteq C(S)$, note que, se $g \in N(S), s \in S$, então $gs = tg$ para algum $t \in S$. Agora, como $s \in \mathcal{P}_n$, temos ou $gs = sg$, ou $gs = -sg$. No primeiro caso, não há mais nada a fazer; no segundo, temos $-sg = tg$, então $t = -s$, i.e., $st = -\mathbb{1}$. Mas isso não pode ocorrer porque $-\mathbb{1} \notin S$; portanto, $gs = sg$.

Elementos do normalizador também podem ser vistos como *operadores lógicos* em palavras código. Mais concretamente, pode-se demonstrar que para um $[[n, k, d]]$-código estabilizador genérico, $N(S)\backslash S$ é gerado por operadores $\overline{X_i}, i = 1, \ldots, k, \overline{Z_i}, i = 1, \ldots, k$, tais que

$$[\overline{X_i}, \overline{X_j}] = 0 \tag{2.38}$$

$$[\overline{Z_i}, \overline{Z_j}] = 0 \tag{2.39}$$

$$[\overline{X_i}, \overline{Z_j}] = 0 \,\text{para}\, i \neq j \tag{2.40}$$

$$\{\overline{X_i}, \overline{Z_j}\} = 0 \tag{2.41}$$

Portanto, esses operadores funcionam da mesma maneira que operadores de Pauli, com a diferença de que eles atuam em palavras código em vez de qubits físicos. Note que, devido à relação dos operadores lógicos com o estabilizador, existe uma certa liberdade em sua escolha. Para o código de Shor, uma escolha possível é $\overline{X} = \bigotimes_{i=1}^{9} Z_i$ e $\overline{Z} = \bigotimes_{i=1}^{9} X_i$. É fácil verificar que as condições acima são satisfeitas e que os operadores atuam nas palavras código da maneira esperada. Para um código estabilizador geral, os operadores lógicos podem ser encontrados escrevendo o código na *forma padrão*; para um procedimento detalhado, veja [NC00, Section 10.5.7], e também [Wil09] para um procedimento alternativo.

Estados estabilizadores. Como vimos, cada estabilizador reduz pela metade a dimensão do espaço do código. Se o número de estabilizadores independentes é igual ao número de qubits, o espaço do código consiste de apenas um estado, chamado *estado estabilizador*. Nem todos os estados são estados estabilizadores, mas esta noção é frequentemente útil. Escrever o estado em termos do conjunto de operadores que o estabilizam[5] pode tornar mais fácil a tarefa de seguir a evolução do estado sob circuitos quânticos. Como notado na Seção 2.1.2, esta noção é especialmente útil para circuitos que consistem apenas de portas de Clifford: como as portas levam operadores de Pauli a operadores de Pauli, é possível seguir a evolução de um estado estabilizador calculando em vez disso a evolução dos operadores de Pauli. Ela também pode ser útil mesmo quando a porta não está no grupo de Clifford; de fato, usaremos essa representação na implementação da porta Toffoli (Seção 4.2.2), apesar do fato de que a porta Toffoli não é uma porta de Clifford.

Notamos que, para uma palavra código arbitrária num estado estabilizador, esta representação pode ser facilmente obtida fixando o autovalor dos operadores lógicos em $+1$. Esta operação efetivamente adiciona os operadores lógicos ao conjunto de estabilizadores.[6]

2.2.3 Códigos CSS

Códigos Calderbank-Shor-Steane (CSS) fornecem uma maneira de construir códigos estabilizadores a partir de códigos lineares clássicos. Antes de introduzi-los, revisamos brevemente a seguir alguns conceitos de correção de erros clássicos.

Um $[n, k, d]$-código[7] é um código clássico que codifica k bits em n bits com distância d. Um código linear pode ser descrito em termos de sua matriz geradora. Um $[n, k, d]$-código possui uma matriz geradora $G \in \{0,1\}^{k \times n}$ com palavras código y dadas por $y = xG$, onde $x \in \{0,1\}^k$ (representado aqui como um vetor linha).

Uma descrição equivalente é dada pela matriz de paridade H, com $n - k$ linhas e n colunas, que satisfaz a condição $yH^T = 0$ para todas as palavras código y. Isto implica $GH^T = 0$. Como o nome indica, cada linha da matriz de paridade corresponde a uma verificação de paridade de um subconjunto da palavra código; o subconjunto é dado pelas entradas da linha com valor 1.

[5] Às vezes chamada a *representação de Heisenberg*; veja [Got98a] para mais detalhes.

[6] De forma recíproca, é possível construir códigos estabilizadores a partir de outros códigos estabilizadores (incluindo estado estabilizadores) removendo estabilizadores. Neste caso é necessário ter certificar-se que as relações (2.38) a (2.41) são satisfeitas.

[7] Note que usamos apenas um par de colchetes, para distinguir códigos clássicos de QECCs.

A última noção que precisamos é a de um código dual. Dado um código C, o seu dual $C^\perp$ é dado pelo espaço ortogonal a C, isto é, $C^\perp = \{x \in \{0,1\}^n : x \cdot y = 0, \forall y \in C\}$. Note que uma matriz geradora para C é uma matriz de paridade para $C^\perp$ e vice-versa.

A ideia de códigos CSS é que estabilizadores podem ser vistos como verificações de paridade. Na construção CSS, toma-se uma matriz de paridade para um código clássico, substituindo todos os 1's na matriz com Zs, e 0's com o operador identidade. Assim, os estabilizadores são linhas da matriz de paridade. O fato de que a matriz aniquila palavras código do código clássico implica que os estabilizadores obtidos desta maneira atuam como a identidade em palavras código. Por outro lado, se uma palavra código do código clássico foi corrompida por um erro, pelo menos uma verificação dará resultado 1 (isto é, a verificação falha); isto implica que a medição do estabilizador correspondente dará resultado -1. Assim, os estabilizadores detectam erros X.

Esses estabilizadores são chamados *estabilizadores tipo* Z, uma vez que eles são um produto de operadores Z. Pode-se aplicar uma estratégia parecida para corrigir erros Z, utilizando estabilizadores tipo X. Mais precisamente, a construção CSS fornece a seguinte receita para construir os estabilizadores. Começamos com dois códigos clássicos: um $[n, k_1, d]$-código C_1 e um $[n, k_2, d]$-código C_2, tais que $C_1 \subseteq C_2$. Os estabilizadores tipo Z correspondem às linhas da matriz de paridade H_2 de C_2, como descrito no parágrafo anterior, enquanto os estabilizadores tipo X correspondem às linhas da matriz de paridade H_1 de $C_1^\perp$, o código dual a C_1. O código dual é um $[n, n-k_1, d]$-código.

No total, temos $n - k_2 + n - (n - k_1) = n + k_1 - k_2$ estabilizadores, o que nos dá um $[[n, k_2 - k_1, d]]$-código. Mas para mostrar que o código é um código estabilizador válido, temos que provar que todo estabilizador tipo X comuta com todo estabilizador tipo Z. Provamos essa afirmação a seguir.

Como C_1 e $C_1^\perp$ são duais, H_1^T é uma a matriz geradora de C_1. A condição $C_1 \subseteq C_2$ implica que para qualquer $x \in \{0,1\}^{k_1}$, temos $H_2(H_1^T x) = 0$. Supondo, sem perda de generalidade, que as colunas de H_1 são linearmente independentes (e analogamente para H_2), isto implica que $H_2 H_1^T = 0$. Este fato significa que cada linha de H_1 é ortogonal a cada linha de H_2. Note agora que estabilizadores tipo Z são dados por Z^u onde U é uma linha de H_2, e estabilizadores tipo X são dados por X^v onde v é uma linha de H_1. Como $ZX = -XZ$, temos $Z^u X^v = (-1)^{u \cdot v} X^v Z^u$, e como $u \cdot v = 0$, isso implica que os estabilizadores comutam.

Operações lógicas. Como notamos acima, para um dado $[n, k_1, d]$-código C_1 e um dado $[n, k_2, d]$-código C_2, a construção CSS nos dá um $[[n, k_2 - k_1, d]]$-código com a

propriedade de que todo estabilizador é tipo Z ou X. Mas qual é a forma das operações lógicas?

Elas podem ser encontradas a partir dos geradores do normalizador $N(S)$. Primeiro, considere um operador da forma Z^u onde u é uma linha do gerador G_1 de $C_1^\perp$. Lembramos que $G_1 H_1^T = 0$, isto é, linhas de G_1 e linhas de H_1 são ortogonais. Uma vez que escolhemos estabilizadores tipo X dados por operadores X^v onde v é uma linha de H_1, isso implica que Z^u comuta com todos os estabilizadores tipo X, e portanto com todo estabilizador, dado que, trivialmente, Z^u comuta com estabilizadores tipo Z. Logo, $Z^u \in N(S)$ para toda linha u em G_1.

Pelo mesmo argumento, operadores da forma X^u onde u é uma linha do gerador G_2 de C_2 estão em $N(S)$. Finalmente, podemos assumir que linhas de G_1 são linearmente independentes, e igualmente para G_2. Isto nos dá $n - k_1$ operadores tipo Z e k_2 operadores tipo X, ou $n + k_2 - k_1$ operadores em total. Já que o código tem $k_2 - k_1$ qubits lógicos e portanto $2k_2 - 2k_1$ operadores lógicos, e $n + k_1 - k_2$ estabilizadores, o normalizador tem dimensão $n + k_1 - k_2 + 2k_2 - 2k_1 = n + k_2 - k_1$. Portanto, esse conjunto de operadores gera o normalizador.

Note que, em particular, os operadores lógicos são produtos de operadores X ou operadores Z. Um procedimento eficiente para encontrar os operadores pode ser encontrado em [Wil09].

2.2.4 O código de Steane

O código de Steane [Ste96] é um código CSS baseado no código de Hamming, que é um $[7, 4, 3]$-código. Este código contém seu próprio dual, isto é, $C^\perp \subseteq C$. Portanto, a restrição imposta pela construção CSS é satisfeita com $C_2 = C_1^\perp = C$. Uma matriz de paridade H é dada por

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{2.42}$$

Logo, o código de Steane tem estabilizadores

$$g_1 = Z_1 Z_3 Z_5 Z_7 \tag{2.43}$$
$$g_2 = Z_2 Z_3 Z_6 Z_7 \tag{2.44}$$
$$g_3 = Z_4 Z_5 Z_6 Z_7 \tag{2.45}$$
$$g_4 = X_1 X_3 X_5 X_7 \tag{2.46}$$
$$g_5 = X_2 X_3 X_6 X_7 \tag{2.47}$$
$$g_6 = X_4 X_5 X_6 X_7 \tag{2.48}$$

Como $C^{\perp}$ é um $[7,3,3]$-código, o código de Steane é um $[[7,1,3]]$-código. Logo, há apenas dois operadores lógicos, $\overline{Z}$ e $\overline{X}$. Uma escolha possível é

$$\overline{Z} = Z_1 Z_2 Z_3 \tag{2.49}$$
$$\overline{X} = X_1 X_2 X_3. \tag{2.50}$$

Capítulo 3

Computação confiável e resistência a vazamentos

Este capítulo explora a noção de computação resistente a vazamentos e sua conexão com computação quântica confiável. Na Seção 3.1, introduzimos a ideia intuitiva de *segurança demonstrável*. Na Seção 3.2, apresentamos um resumo de criptografia abstrata (AC), um *framework* para definir sistemas criptográficos e provar propriedades de segurança sobre eles. Na Seção 3.3, motivamos o estudo de circuitos resistentes a vazamento e fornecemos uma definição de segurança para este cenário em termos do *framework* AC. Na Seção 3.4, usamos esta definição para descrever formalmente diversos modelos para vazamento utilizados na literatura, e a Seção 3.5 fornece alguns exemplos de resistência a vazamento utilizados na prática. Na Seção 3.6, definimos o modelo de circuitos ruidosos usado neste trabalho e também definimos circuitos confiáveis, que são relacionados a circuitos resistentes a vazamento na Seção 3.7.

3.1 Segurança demonstrável

Em um sentido prático, a tarefa de definir segurança começa pela análise do cenário de execução e uma avaliação dos possíveis ataques ao sistema. No entanto, quando o objetivo é propor um protocolo para ser usado em diversos cenários diferentes, é desejável introduzir um certo grau de abstração. O objetivo final da área de criptografia teórica é alcançar segurança *demonstrável* a partir de hipóteses razoáveis. O desafio é propor hipóteses que sejam claras e baseadas em considerações empíricas, e que ao mesmo tempo mantenham geral o cenário de execução.

Tradicionalmente, segurança em criptografia teórica é definida por meio de um *jogo* envolvendo o adversário. Como exemplo, considere segurança de textos cifrados escolhidos (também chamados CCA, de *chosen-ciphertext attack*), definidos para esquemas

de cifra. Em linhas gerais, o jogo CCA se segue da seguinta maneira. O adversário possui acesso a um oráculo de encriptação, isto é, um dispositivo a partir do qual o adversário pode obter o texto cifrado para uma mensagem arbitrário, e um oráculo de decriptação, a partir do qual o adversário pode enviar um texto cifrado arbitrário e receber o texto puro correspondente. Posteriormente, o adversário envia duas mensagens a um desafiador, que envia o texto cifrado de uma das mensagens para o adversário. O adversário vence o jogo se ele consegue adivinhar qual mensagem foi cifrada com probabilidade significativamente melhor que $1/2$.

Com essa definição, fica claro qual tipo de ataque está sendo considerado, mas ela possui a desvantagem de ser bastante específica. Uma consequência deste fato é que apesar de segurança CCA ser considerada uma definição forte de segurança, alcançá-la para um esquema pode não significar muito, se se deseja implementar o esquema em um cenário com um modelo de ataque ligeiramente diferente. De fato, existem diversas variantes da segurança CCA, correspondendo a cenários diferentes.

Um outro problema com definições de segurança baseadas em jogos é que, em geral, elas não podem ser *compostas*. Isto é, combinar a execução de instâncias de dois esquemas seguros—de acordo com a definição—ou até combinar várias instâncias do mesmo esquema não garante que a execução combinada é segura.

Esta objeção despertou o interesse dos criptógrafos em estabelecer uma definição de segurança em que protocolos podem ser compostos mantendo a segurança. O modelo mais conhecido que garante tal propriedade é o de *composabilidade universal* (UC, de *universal composability*), proposto por Canetti [Can01]. No entanto, o *framework* UC é bastante complexo. Definir segurança neste modelo requer a definição precisa do modelo computacional e canais de comunicação utilizados. As hipóteses no modelo em si são mínimas, mas os detalhes adicionais que as definições requerem podem tornar menos claro o significado de segurança.

3.2 Criptografia abstrata

Neste trabalho, usamos o *framework* de *criptografia abstrata*, introduzido por Maurer e Renner [MR11] (veja também [Mau12, PR14]), do qual fornecemos um resumo nesta seção. A ideia é que os blocos básicos das construções sejam especificados apenas em termos de seu comportamento de entrada e saída, independente do funcionamento interno. É claro que, em uma implementação real, é necessário considerar as partes internas destes blocos, mas tais detalhes não são relevantes para a definição de segurança. Estes blocos básicos serão chamados de *recursos*—veremos mais detalhes na Seção 3.2.1, mas já é útil introduzir um exemplo de como eles são usados no framework.

Considere criptografia simétrica, em que dois participantes, Alice e Bob, compartilham inicialmente uma chave secreta. Alice encripta suas mensagens usando a chave e envia o texto cifrado a Bob, que o decripta usando a mesma chave. Neste cenário, supomos que dois recursos estão disponíveis. Um é um recurso que gera uma chave secreta e a distribui às duas partes. O outro é um canal de comunicação autenticado, tal que se garante que a mensagem recebida por Bob é a mesma mensagem enviada por Alice. Note que não supomos que o canal seja confidencial; logo, mensagens enviadas através dele também são enviadas ao adversário.

A construção mais conhecida para criptografia simétrica é o *one-time pad*. Neste *framework*, mostrar que o *one-time pad* é seguro equivale a provar que, em posse dos recursos disponíveis e o protocolo especificado pela construção, é possível construir um canal seguro, em que o adversário não recebe as mensagens enviadas.

3.2.1 Recursos

É útil formular esse cenário em termos de *recursos ideais* e *recursos reais*. Os recursos reais são os recursos disponíveis aos participantes, e recursos ideais realizam as tarefas que os participantes desejam construir. No exemplo do *one-time pad*, o recurso ideal é o canal seguro, enquanto o canal autenticado juntamente com uma chave compartilhada secreta é usada como um recurso real. É importante notar que as noções de "real" e "ideal" são sempre relativas às hipóteses relevantes ao protocolo a ser implementado. Por exemplo, em um protocolo para autenticação, o canal autenticado toma o papel de recurso ideal, e os recursos reais são um canal completamente inseguro juntamente com uma chave secreta.

Neste *framework*, um recurso é um tipo de *sistema*, que é definido como um objeto abstrato que pode ser composto com outros sistemas. Cada sistema é acompanhado de um conjunto de rótulos $\mathcal{I}$, chamado *conjunto de interfaces*. Cada participante tem acesso a uma interface. Interfaces de sistemas diferentes podem ser conectadas para formar novos sistemas.

Recursos são sistemas onde cada interface corresponde a um participante que possui acesso a ela. Como exemplo de um recurso, podemos definir um canal privado entre partes honestas A e B sujeitas a espionagem por uma terceira parte E como um recurso que toma entradas de A e as entrega a B, como mostrado na Figura 3.1. Uma vez que o recurso não dá nenhuma saída a E, o canal é privado por definição. Note que isto é válido independente se E age honestamente ou desonestamente; em criptografia abstrata, o objetivo é emular o comportamento de recursos ideais em todas as situações, não apenas sob a hipótese de que certas partes são honestas e outras desonestas.

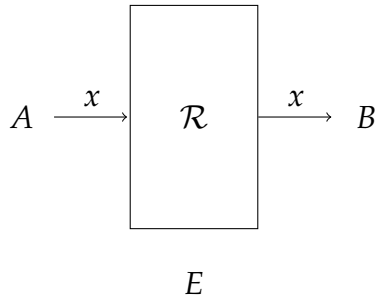


Figura 3.1: Canal privado.

Recursos são transformados em outros recursos por meio de composição com *conversores*. Um conversor é um sistema com duas interfaces: uma interface interna, que o conecta ao recurso, e uma interface externa, que o conecta ao participante. Um *protocolo* especifica um conversor para um conjunto de participantes. Um participante é *honesto* se ele segue o protocolo e é desonesto caso contrário.

Formalmente, recursos com um conjunto de interfaces fixo podem ser definidas como elementos em uma álgebra (chamada de *álgebra criptográfica*), onde novos elementos são obtidos a partir de composição paralela e aplicação de conversores.

Definição 1

Uma álgebra criptográfica $\langle \Phi, \Sigma, \mathcal{I}, \approx \rangle$ é um conjunto Φ de recursos com o conjunto de interfaces $\mathcal{I}$, um conjunto Σ de conversores, e uma relação de equivalência $\approx$ sobre Φ, juntamente com uma operação $f : \Phi \times \Sigma \times \mathcal{I} \to \Phi$, para a qual denotamos $f(\mathcal{R}, \alpha, i)$ por $\alpha_i \mathcal{R}$, e uma operação $\| : \Phi \times \Phi$ (chamada composição paralela), tais que

1. $\alpha_i \beta_j \mathcal{R} \approx \beta_j \alpha_i \mathcal{R}$ para todos $i \neq j$, $\mathcal{R} \in \Phi$ e $\alpha, \beta \in \Sigma$.
2. Existe $\mathrm{id} \in \Sigma$ tal que $\mathrm{id}_i \mathcal{R}$ para todos $i \in \mathcal{I}$ e $\mathcal{R} \in \Phi$.
3. Se $\mathcal{R} \approx \mathcal{S}$ então $\alpha_i \mathcal{R} \approx \alpha_i \mathcal{S}$.
4. Se $\mathcal{R} \approx \mathcal{S}$ então $(\mathcal{R} \| \mathcal{T}) \approx (\mathcal{S} \| \mathcal{T})$ e $(\mathcal{T} \| \mathcal{R}) \approx (\mathcal{T} \| \mathcal{S})$ para todo $\mathcal{T} \in \Phi$.

Definimos as composições paralela e serial de conversores da seguinte maneira:

$$(\alpha\beta)_i \mathcal{R} := \alpha_i(\beta_i \mathcal{R}) \tag{3.1}$$

$$(\alpha \| \beta)_i (\mathcal{R} \| \mathcal{S}) := (\alpha_i \mathcal{R}) \| (\beta_i \mathcal{S}) \tag{3.2}$$

Para incluir a possibilidade de construções que apenas aproximam um recurso ideal desejado, em vez de construi-lo perfeitamente, precisamos de uma noção de distância d entre dois recursos. A distância deve ser uma pseudométrica sobre o conjunto de recursos. Mais precisamente, ela deve satisfazer as seguintes propriedades para conversores arbitrários α e recursos $\mathcal{R}, \mathcal{S}, \mathcal{T}$:

$$d(\alpha\mathcal{R}, \alpha\mathcal{S}) \leq d(\mathcal{R}, \mathcal{S}) \tag{3.3}$$

$$d(\mathcal{R}\|\mathcal{T}, \mathcal{S}\|\mathcal{T}) \leq d(\mathcal{R}, \mathcal{S}) \tag{3.4}$$

Neste trabalho, definiremos a distância como a vantagem máxima que um sistema que tente distinguir entre dois recursos (o *distinguidor*) pode possuir. Um distinguidor é um sistema que conecta todas as interfaces de um recurso e possui uma interface externa que dá um bit como saída. Cada bit corresponde a uma estimativa de qual recurso está interagindo com o distinguidor. Como a escolha é entre dois recursos, dizemos que o distinguidor possui vantagem ε se ele consegue adivinhar o recurso corretamente com probabilidade $1/2 + \varepsilon$. ($\varepsilon = 0$ corresponde a segurança perfeita—neste caso, a melhor escolha que o distinguidor consegue fazer é completamente aleatória.)

Escrevemos $d(\mathcal{R}, \mathcal{S}) \leq \varepsilon$ ou $\mathcal{R} \approx_\varepsilon \mathcal{S}$ se $\mathcal{R}$ e $\mathcal{S}$ podem ser distinguidos com vantagem de até ε. Pode-se mostrar que esta distância é equivalente à distância estatística [PR14, Section 6.1].

Para definir segurança, temos que considerar *recursos filtrados*. Um filtro é um conversor que "bloqueia" parcialmente uma interface externa (veja a Figura 3.2). Em geral, tais recursos são necessários para permitir participantes que podem adquirir acesso adicional ao recurso quando eles agem de maneira desonesta. Por exemplo, imagine que o recurso disponível é um canal de comunicação de A para B e existe um adversário E. Suponha que E tem a opção de deletar mensagens enviadas por A, de maneira que elas nunca são recebidas por B. Se E utiliza essa opção para toda mensagem, é impossível construir um canal privado a partir de tal canal, porque B nunca receberia a mensagem. Mas podemos relaxar nossas exigências: poderíamos exigir que *se* E não deleta a mensagem, então a mensagem recebida por B deve ser igual à mensagem enviada por A.

Essa consideração nos leva a duas condições diferentes: uma para o caso em que E é honesto e outra para o caso em que E é desonesto. No primeiro caso, a interface de E é filtrada, e se exige que o protocolo faça a funcionalidade desejada, enquanto no segundo caso, somente exigimos sigilo, isto é, nenhuma informação deve vazar a E.

No caso geral onde há n partes e qualquer uma delas pode ser desonesta, temos 2^n condições diferentes, uma para cada escolha de partes honestas. A seguir, temos uma

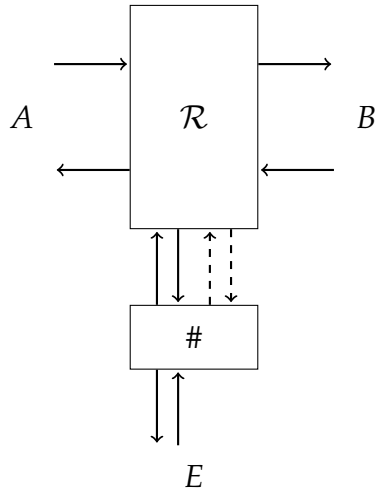


Figura 3.2: Recurso filtrado.

definição de segurança que é suficiente para implicar a definição dada em [MR11].

Definição 2 Sejam $\mathcal{R}_\phi$ e $\mathcal{S}_\psi$ recursos com o mesmo conjunto de interfaces $\mathcal{I}$ e sejam $\{\phi_i\}_{i\in\mathcal{I}}$,$\{\psi_i\}_{i\in\mathcal{I}}$ conjuntos de filtros. Para um protocolo $\pi = \{\pi_i\}_{i\in\mathcal{I}}$, escrevemos $\mathcal{R} \xrightarrow{(\pi,\varepsilon)} \mathcal{S}$ se existe um conjunto de conversores $\{\sigma_i\}_{i\in\mathcal{I}}$ tais que para todo $\mathcal{P} \subseteq \mathcal{I}$, temos

$$\pi_\mathcal{P}\phi_\mathcal{P}\mathcal{R} \approx_\varepsilon \sigma_{\overline{\mathcal{P}}}\psi_\mathcal{P}\mathcal{S}. \tag{3.5}$$

Por [MR11, Theorem 2], esta definição implica a definição de segurança dada no trabalho citado. Consequentemente ela possui a propriedade de *composabilidade*. Isto é, temos o seguinte resultado:

1. Se $\mathcal{R}_\alpha \xrightarrow{(\alpha,\varepsilon)} \mathcal{S}_\beta$ e $\mathcal{S}_\beta \xrightarrow{(\beta,\varepsilon')} \mathcal{T}_\gamma$ então $\mathcal{R}_\alpha \xrightarrow{(\alpha\circ\beta,\varepsilon+\varepsilon')} \mathcal{T}_\gamma$.
2. Se $\mathcal{R}_\alpha \xrightarrow{(\pi,\varepsilon)} \mathcal{S}_\gamma$ e $\mathcal{R}'_\beta \xrightarrow{(\pi',\varepsilon')} \mathcal{S}'_\delta$ então $\mathcal{R}_\alpha \| \mathcal{R}'_\beta \xrightarrow{(\pi\|\pi',\varepsilon+\varepsilon')} \mathcal{S}_\gamma \| \mathcal{S}'_\delta$.

Para nosso tratamento de resistência a vazamentos, precisamos apenas considerar a Definição 2 no caso restrito de duas partes, isto é, o conjunto de interfaces possui apenas dois elementos, e no máximo um deles é desonesto. Neste caso, a definição toma uma forma particularmente simples. A Condição 3.5 é reduzida a duas equações, e, devido ao fato de que no máximo uma parte é desonesta, precisamos de filtros em apenas uma interface, que chamamos de E na definição a seguir.

Definição 3 [Caso especial da Definição 2] Sejam $\mathcal{R}$ e $\mathcal{S}$ recursos com o conjunto de interfaces $\mathcal{I} = \{A, E\}$. Dizemos que o protocolo $\pi = \{\pi_A, \pi_E\}$ constrói $\mathcal{S}$ a

partir de $\mathcal{R}$ de forma segura com distância ε (denotado $\mathcal{R} \xrightarrow{(\pi,\varepsilon)} \mathcal{S}$) se existe um conversor σ_E (chamado simulador) tal que

$$\pi_A \pi_E \mathcal{R} \approx_\varepsilon \mathcal{S} \tag{3.6}$$

$$\pi_A \mathcal{R} \approx_\varepsilon \sigma_E \mathcal{S}. \tag{3.7}$$

Como discutimos anteriormente, na definição acima, $\mathcal{R}$ toma o papel do recurso real, uma vez que assumimos que o recurso está disponível. Por outro lado, $\mathcal{S}$ é o recurso ideal.

3.3 Definição de segurança para resistência a vazamentos

Agora podemos fornecer uma definição de computação resistente a vazamentos introduzindo os recursos ideal e real. Mas primeiro, vamos considerar um cenário prático em que se deseja ter resistência a vazamentos, e descrever a tarefa desejada em um sentido concreto.

3.3.1 Motivação

Smart cards são circuitos integrados que têm sido amplamente usados para autenticação e armazenamento de dados sensíveis. Eles são portadores móveis de informação como, por exemplo, dinheiro e históricos médicos. Dado que *smart cards* são projetados de forma a serem portáveis, eles estão sujeitos a diversos ataques físicos. Ataques possíveis incluem medir o tempo e a corrente elétrica usadas durante a operação (análise de consumo). Logo, resistência a vazamentos é essencial ao projeto de *smart cards*.

Mais precisamente, um *smart card* armazena dados internos, fornece uma interface para entradas externas, executa computações que dependem dos dados internos e entrada externa e retorna a saída através de uma interface de saída. Um dos objetivos é que se o *smart card* é entregue a um adversário, que pode enviar entradas e receber saídas do cartão, ele não deveria ser capaz de obter qualquer informação sobre os seus dados internos, além do que pode ser aprendido da saída regular da computação. Esse objetivo é o que entendemos por resistência a vazamentos. No entanto, enquanto *smart cards* são normalmente projetados de forma a ser resistentes contra ataques específicos, estamos interessados em classes maiores de ataques. Ademais, *smart cards* são dispositivos que executam uma função fixa, enquanto nosso objetivo é ter resistência a vazamentos para funções genéricas.

Nosso cenário geral para vazamento é o seguinte: primeiro, uma parte honesta (Alice) dá como entrada um circuit $\mathcal{C}$ que computa uma função f. Ela também dá como entrada uma string secreta y. Depois, o adversário (Eva) recebe acesso ao circuito "como caixa preta", isto é, ela pode—de maneira interativa—enviar entradas ao circuito e receber saídas. A função computada pelo circuito depende das entradas enviadas por Eva e da string secreta escolhida por Alice, isto é, se Eva entra com x, ela recebe $f(x, y)$. Eva também recebe uma descrição do circuito. Adicionalmente, a cada interação Eva obtém bits que vazam de acordo com o modelo de vazamento. Grosso modo, o objetivo é que Eva não deve aprender nenhuma informação sobre y além do que ela receberia com a saída $f(x, y)$. Este objetivo será formalizado abaixo mediante a introdução do recurso ideal, enquanto o cenário em que há vazamento corresponde ao recurso real.

3.3.2 Recurso ideal

Tendo em mente a descrição acima, o cenário é o seguinte. Ao início da execução, Alice dá como entrada o circuito e a string secreta. Posteriormente, uma descrição do circuito é dada a Eva, que pode executar o circuito como uma caixa preta.

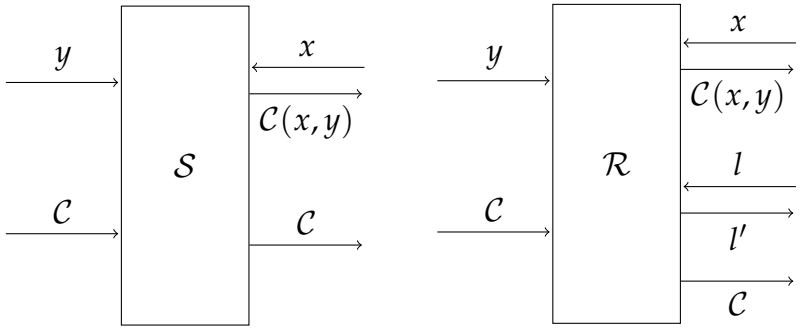
Definimos o recurso ideal $\mathcal{S}$ da seguinte forma. Inicialmente, Alice dá como entrada y e uma descrição de um circuito $\mathcal{C}$ que computa uma função f. Para uma entrada x enviada por Eva, ela recebe a saída correspondente $\mathcal{C}(x, y)$. A funcionalidade ideal também dá a descrição de $\mathcal{C}$ a Eva. O recurso é mostrado na Figura 3.3a.

Esta definição implica que informação sobre y somente pode vazar através de $f(x, y)$ e $\mathcal{C}$. Como um exemplo concreto, seja $\mathcal{C}$ um circuito que implementa um algoritmo de encriptação que encripta uma entrada x usando a chave secreta y. Para estabelecer segurança para um esquema criptográfico, supõe-se que a chave secreta é completamente secreta do ponto de vista de Eva. No exemplo, esta suposição é garantida pela forma em que o recurso ideal é definido, uma vez que a única informação sobre y que pode vazar para Eva é através do texto cifrado $f(x, y)$.

Este exemplo também ilustra que segurança e resistência a vazamento são objetivos separados: não há restrições impostas em f, logo, mesmo se a função revela alguma informação sobre y, a resistência a vazamentos não é violada. A condição de resistência a vazamentos apenas garante que não é vazada nenhuma informação adicional sobre y.

3.3.3 Recurso real

Informalmente, o recurso real (que denotamos por $\mathcal{R}$) é uma versão do recurso ideal que vaza informação, isto é, informação adicional se torna disponível a Eva. Como anteriormente, a interface de Alice a permite dar como entrada um segredo y e um circuito $\mathcal{C}$ que implementa uma função f. A interface de Eva a permite enviar entradas x e receber as saídas correspondentes $\mathcal{C}(x, y)$. Adicionalmente, ela pode enviar *pedidos de vazamento* l, recebendo vazamentos l'. Assim como no recurso ideal, Eva também recebe $\mathcal{C}$. A ideia é que Eva usa $\mathcal{R}$ como uma caixa preta, mas também recebe informações adicionais a partir dos vazamentos, que podem revelar algo sobre o segredo. Este cenário é representado na Figura 3.3b.



(a) (b)

Figura 3.3: (a) Recurso ideal para resistência a vazamentos e (b) recurso real. Em ambos os casos, Alice tem acesso à interface mostrada à esquerda e Eva tem acesso à interface mostrada à direita.

Para satisfazer as condições na Definição 3, é necessário um protocolo π para construir o recurso ideal $\mathcal{S}$ a partir do recurso real $\mathcal{R}$. A primeira condição na Definição 3 é apresentada na Figura 3.4.

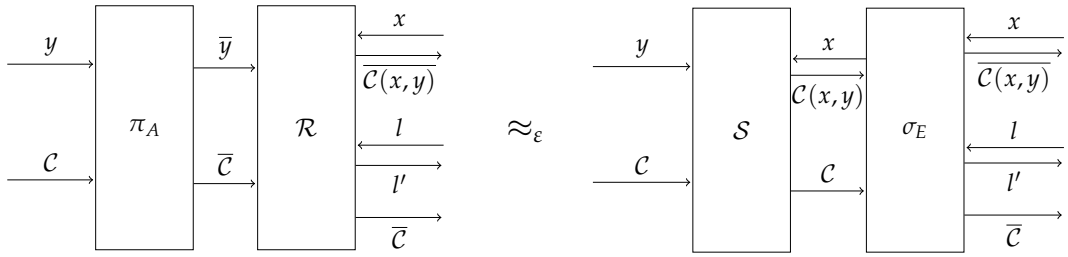


Figura 3.4: A primeira condição na Definição 3 aplicada a resistência a vazamentos. Para provar que o conversor π_A é parte de um protocolo resistente a vazamentos π, é necessário mostrar a existência de um simulador σ_E tal que o recurso ideal $\mathcal{R}$ executado em conjunto com π_A é indistinguível da execução do recurso ideal $\mathcal{S}$ com σ_E.

Nossa descrição do recurso real captura vazamento em uma forma bem geral, no sentido de que $\mathcal{C}, l$ e l' podem ser arbitrários. No entanto, para que a resistência a vazamentos seja possível, é necessário especificar a forma em que os pedidos l são escolhidos. Assim, construímos esquemas que são resistentes a vazamento com uma estrutura particular. Para formalizar esta noção, vamos tornar o cenário apresentado acima mais concreto, restringindo as interações da seguinte maneira. Eva interage com o recurso em diversas rodadas. A cada rodada, ela entra com pares (x, l) e recebe saídas $(f(x, y), l')$.[1]

A estratégia particular usada para escolher os pedidos de vazamento l é chamado um *modelo de vazamento*. O modelo de vazamento é um conjunto de pedidos de vazamento permitidos. Um pedido de vazamento é uma distribuição de probabilidade sobre funções de vazamento l_i. Cada l_i é uma função do conjunto de valores atribuídos aos fios no circuito; denotamos este conjunto por $W_{\mathcal{C}}(x, y)$ (quando as entradas ao circuito são x e y). Eva envia $l = \{(l_i, p_i)\}$. A função de vazamento a ser executada é escolhida pelo recurso de acordo com as probabilidades dadas pelo pedido. Isto é, l_i é escolhida com probabilidade p_i. Se a função l_i é escolhida, Eva recebe $l' = l_i(W_{\mathcal{C}}(x, y))$. Note que a saída de l_i pode incluir ou não o índice i, dependendo do modelo de vazamento.

Como exemplo, Eva poderia escolher a cada rodada um número limitado de fios do circuito, cujos valores vazam para ela. (Esses são os "probing attacks" considerados em [ISW03].) Ou poderíamos também restringir Eva a funções de vazamento que são computadas com circuitos de profundidade constante [FRR+10]. Em todo caso, definimos compiladores resistentes a vazamento com respeito a um modelo de vazamento em particular da seguinte maneira. Na definição abaixo, $\mathcal{S}$ e $\mathcal{R}$ são respectivamente os recursos ideal e real definidos acima.

Definição 4 O protocolo π é um ε-compilador resistente a vazamentos contra o modelo de vazamentos L se ele constrói $\mathcal{S}$ a partir de $\mathcal{R}$ de forma segura com distância ε, onde os pedidos de vazamento são obtidos de L.

Observação 1. Na construção do protocolo $\pi = \{\pi_A, \pi_E\}$, focamos nossa atenção no conversor π_A. Isto é devido ao fato de que para um dado π_A apropriado, é sempre trivial construir um conversor π_E tal que a Condição (3.7) na Definição 3 é satisfeita. Construimos π_E da seguinte forma. π_E retransmite entradas x de Eva para $\mathcal{R}$ e decodifica a saída codificada $\overline{\mathcal{C}(x, y)}$, que é posteriormente transmitida a Eva. π_E também decodifica o circuito codificado $\mathcal{C}$ antes de passá-lo a Eva. Crucialmente, π_E não for-

[1] Poderíamos também especificar que a saída da função seja uma função de l; esta situação capturaria *tampering*—em que Eva pode introduzir falhas nos fios do circuito [IPSW06, AK96].

nece a interface de vazamento (que recebe l e envia l') a Eva, efetivamente protegendo o vazamento do acesso por ela.

3.4 Modelos de vazamento

Abaixo apresentamos três exemplos de modelos de vazamentos que são frequentemente considerados na literatura. Em particular, construimos um esquema explícito para o modelo que chamamos de vazamento independente (Capítulo 5). Todos os modelos apresentados aqui são modelos de "wire probing", em que a saída da função de vazamento consiste dos valores em um conjunto de fios no circuito.

Vazamento independente. Vazamento independente é caracterizado pela possibilidade de que cada fio no circuito vaze, cada um com uma probabilidade fixa p. Este modelo foi considerado em [DDF14] para provar resistência a vazamentos de um esquema contra um modelo diferente, chamado vazamento ruidoso (o qual descreveremos posteriormente). Vazamento independente é chamado "wire probing" no trabalho citado.

Podemos formalizar vazamento independente da seguinte maneira. Seja n o número de fios no circuito. Rotulamos cada fio com um índice i onde $1 \leq i \leq n$. Agora, seja w uma string binária de tamanho n e seja $l_w(x, y)$ o conjunto de valores dos fios vazados quando o circuito tem (x, y) como entrada. Se $w_i = 1$, atribuimos a $(l_w)_i(x, y)$ o valor do i-ésimo fio, enquanto se $w_i = 0$, atribuimos $(l_w)_i(x, y) = \perp$. Para cada rodada de interação, a probabilidade p_w de que a função de vazamento l_w foi escolhida é dada por $p_w = \Pr(X = |w|)$, onde X segue uma distribuição binomial com parâmetros n e p. O modelo de vazamento L é então $L = \{(l_w, p_w) \colon w \in \{0,1\}^n\}$. (Note que existe apenas um pedido de vazamento possível.)

Vazamento adversarial. Uma variante do modelo de vazamento independente é considerar o cenário onde o adversário escolhe um número t de fios para vazar. Este modelo é frequentemente chamado o *t-wire probing model* (e chamado *t-threshold-probing* em [DDF14]) e foi considerado inicialmente em [ISW03]. Intuitivamente, o modelo é uma versão mais forte do vazamento independente: para vazamento independente com probabilidade de vazamento p em um circuito com n fios, $t = np$ vazam em média. No entanto, o fato de que, para vazamento adversarial, o adversário escolhe os fios a vazar torna a tarefa de proteger o circuito mais difícil.

Formalmente, o tratamento também é similar. O conjunto de funções de vazamento $\{l_w\}$ é o mesmo que no caso de vazamento independente, mas agora há vários pedidos

de vazamento possíveis, um para cada função. Dado um pedido de vazamento, a função é escolhida de maneira determinística, então temos $p_w = 1$ para cada l_w. Isto é, o modelo de vazamento L é definido como $L = \{\{(l_w, 1)\} : w \in \{0,1\}^n, |w| \leq t\}$.

Vazamento ruidoso. Nesta variante, considerada em [FRR+10, PR13, DDF14], todos os valores dos fios vazam, mas o adversário recebe uma versão ruidosa dos valores. Isto é, cada bit é virado com probabilidade p (o nível de ruído). Formalizamos esta noção definindo funções de vazamento l_w onde $l_w(x, y)$ é o valor de todos fios no circuito, exceto pelos fios i com $w_i = 1$, cujos valores são virados. As probabilidades p_w são dadas por $p_w = \Pr(X = |w|)$ onde X segue uma distribuição binomial com parâmetros n e $1 - p$. O modelo de vazamento é dado por $L = \{(l_w, p_w) : w \in \{0,1\}^n\}$. Note que ao contrário do modelo de vazamento independente, o adversário não aprende quais fios vazam.

Convertendo vazamento independente a vazamento ruidoso. Uma maneira alternativa de interpretar os modelos de vazamento independente e ruidoso é visualizar o vazamento como a saída de um *canal*. Mais precisamente, para vazamento independente com probabilidade de vazamento p, podemos ver a função de vazamento como o envio do valor de cada fio por um canal do tipo *erasure*, onde cada valor é eliminado com probabilidade $1 - p$. A saída do canal é então dada para o adversário. Similarmente, para vazamento ruidoso com nível de ruído p, o valor de cada fio é enviado por um canal binário simétrico com probabilidade de erro p, e a saída é dada ao adversário.

Dada esta observação, podemos mostrar que um protocolo que é resistente a vazamento independente também é resistente a vazamento ruidoso, para algum nível de ruído. Temos o seguinte resultado.

Lema 1. *Seja π um ε-compilador resistente a vazamentos contra vazamento independente com probabilidade de vazamento p. Então π é um ε-compilador resistente a vazamentos contra vazamento ruidoso para qualquer nível de ruído p' que satisfaça $(1 - p)/2 < p' < 1/2$.*

Demonstração. Como notado na Observação 1, apenas precisamos mostrar a primeira condição na Definição 3. Sejam $\mathcal{R}_i, \mathcal{R}_n$ os recursos ideais com pedidos de vazamento dados pelos modelos de vazamento independente e ruidoso, respectivamente. Como observado, estes modelos possuem somente um pedido de vazamento possível, que denotamos respectivamente por l_i e l_n. Mostraremos como construir um simulador σ_n tal que $\pi_A \mathcal{R}_n = \sigma_n \pi_A \mathcal{R}_i$. Como por hipótese $\pi_A \mathcal{R}_i \approx_\varepsilon \sigma_i \mathcal{S}$ para algum simulador σ_i, temos $\pi_A \mathcal{R}_n \approx_\varepsilon (\sigma_n \sigma_i)\mathcal{S}$.

Obtemos vazamento ruidoso a partir de vazamento independente enviando as saídas do canal *erasure* através de um canal que torna o canal composto um canal biná-

rio simétrico. Seja $p' = p(1-q) + (1-p)/2$ onde $0 < q < 1/2$. Para o modelo de vazamento independente, cada bit de saída na função de vazamento é a saída de um canal binário de *erasure* com probabilidade de apagamento $1-p$, isto é, o canal U caracterizado pelas probabilidades condicionais $U(x|x) = p, U(\perp|x) = 1-p$ para $x \in \{0,1\}$. Para o modelo de vazamento ruidoso, cada bit de saída é a saída de um canal binário simétrico com probabilidade de erro $1-p'$, isto é, o canal W dado por $W(x|x) = p', W(1 \oplus x|x) = 1-p'$ para $x \in \{0,1\}$. Agora, seja V o canal com espaço de entrada $\{0,1,\perp\}$ e espaço de saída $\{0,1\}$ dado por $V(x|x) = 1-q, V(1 \oplus x|x) = q, V(x|\perp) = 1/2$ para $x \in \{0,1\}$. Concatenando U e V, obtemos W.

Assim, dado um simulador σ_i para $\mathcal{R}_i$, construimos um simulador σ_n para $\mathcal{R}_n$ da seguinte maneira. σ_n envia entradas x da interface externa à interface interna e envia saídas $f(x,y)$ da interface interna à interface externa. Ao receber l_n na interface externa, σ_n envia l_i à interface interna. Ao receber um vazamento l' na interface interna, σ_c envia todos os bits l'_i através do canal V e envia para a interface externa a string l'_n, que consiste de todos os bits dados como saída pelo canal. Assim, podemos gerar vazamento ruidoso executando vazamento independente através de σ_n. Isto mostra que $\mathcal{R}_n = \sigma_n \mathcal{R}_i$. □

Componentes livres de vazamento. Muitas das construções para resistência a vazamento exigem que uma pequena componente do circuito seja confiável ou *livre de vazamento*, isto é, uma componente cujos fios internos não vazam para Eva. Para a nossa construção, isto inclui fios de saída da componente. Incorporamos esta exigência em nossa definição de resistência a vazamento adicionando a restrição de que a função de vazamento l não pode depender de quaisquer fios de uma componente livre de vazamentos, incluindo os fios que saem dela.

A única componente livre de vazamentos que usamos neste trabalho é uma fonte de bits aleatórios: uma porta que não toma nenhuma entrada e dá como saída um bit uniformemente distribuído (que, por hipótese, não vaza ao adversário). Esta componente é utilizada na tradução de um circuito quântico a um circuito clássico. Deixamos em aberto a questão de se esta restrição é realmente necessária em geral. Notamos que nossa restrição é diferente da restrição utilizada em [FRR+10]: exigimos apenas bits uniformemente distribuídos, enquanto a transformação no trabalho citado exige bits distribuídos de acordo com uma distribuição arbitrária (embora fixa). Por outro lado, os fios de saída da componente livre de vazamentos usada por eles podem vazar, enquanto neste trabalho exigimos que componentes livres de vazamento não vazam antes de interagir com outras componentes.

3.5 Resistência a vazamentos na prática

Modelos de vazamento frequentemente se tornam populares na comunidade devido a sua conveniência matemática, mas, em última análise, a utilidade de um modelo é determinada por sua aplicabilidade em cenários reais. Dado que esta qualidade pode ser difícil de avaliar, os responsáveis por implementações práticas frequentemente ignoram a análise teórica completamente e projetam suas próprias medidas para proteger contra ataques de canal lateral. Terminaremos esta seção com uma breve discussão de exemplos de ataques de canal lateral que têm sido aplicados contra implementações utilizadas na prática, assim como algumas das ténicas utilizadas para proteger contra estes ataques. Embora as considerações a seguir não se apliquem diretamente a este trabalho, usaremos a oportunidade para fazer algumas comparações entre as técnicas utilizadas e o esquema proposto neste trabalho.

3.5.1 Análise de consumo

Ataques de análise de consumo[2] são baseados no fato de que dispositivos consumem energia durante a execução, e que o gráfico da corrente utilizada ao longo do tempo mostra padrões diferentes para operações diferentes. Como ataques de análise de consumo são relativamente baratos, gerais e impossíveis de detectar, eles são bastante relevantes à implementação de dispositivos criptográficos. De fato, desde o trabalho inicial por Kocher *et al.* [KJJ99], uma grande parte da literatura em ataques de canal lateral tem focado neste tipo de ataque.

Como exemplo simples de funcionamento de um ataque de análise de consumo, considere uma implementação de um algoritmo que envolve iterar sobre os n bits da chave secreta, e onde a cada passo i da iteração (para $1 \leq i \leq n$), a sequência de operações a ser executadas depende do valor do i-ésimo bit. Neste caso, é fácil adivinhar o valor da chave secreta observando o padrão do consumo de energia ao longo da iteração.

Embora este tipo de ataque possa ser bastante poderoso, ele é frequentemente difícil de implementar, uma vez que medições de energia tendem a ser bastante ruidosas, o que é particularmente importante quando o dispositivo consome pouca energia. Para superar esta limitação, um método diferente é proposto em [KJJ99]: análise de consumo diferencial—ou *differential power analysis* (DPA), em inglês. No trabalho citado, um ataque específico contra o algoritmo de chave simétrica DES é descrito, mas o mesmo método funciona contra AES, e há variações para criptossistemas diferentes.

[2] *Power analysis*, em inglês.

De maneira resumida, DPA funciona da seguinte maneira. Seja $m_1, \ldots, m_k$ uma lista de textos em claro e sejam $P_1, \ldots, P_k$ as energias médias calculadas a partir dos gráficos de energia correspondente. Suponha agora que um bit de um valor intermediária na execução do algoritmo depende apenas do texto claro m e uma parte da chave, uma string de s bits que denotaremos por K. Denotamos o valor intermediário por $L(m, K)$. Após escolher um valor para K, calculamos $L(m_1, K), \ldots, L(m_k, K)$. Para $i \in \{0, 1\}$, seja M_i o conjunto de textos em claro m para os quais $L(m, K) = i$.

A ideia é que se K não é parte da chave verdadeira, então i corresponde apenas ao valor intermediário obtido na execução real para cerca de metade dos textos em claro (para um número suficientemente grande de textos). O resultado é que os gráficos de energia correspondentes a textos em M_0 são estatisticamente indistinguíveis dos padrões para textos em M_1. Assim, se calcularmos a diferença entre a energia média sobre textos em M_0 e textos em M_1, esperamos que o sinal resultante esteja próximo de zero.

Se, por outro lado, K é parte da chave verdadeira, então i corresponde ao valor intermediário para todos os textos em claro. Neste caso, a diferença vai ser significativamente maior que zero. Logo podemos obtemer o valor correto de K.

3.5.2 Ataques acústicos

É longa a história de ataques de canal lateral que exploram os *sons* produzidos por um dispositivo. Dispostivos de ciframento eletromecânicos como máquinas que utilizam rotores, amplamente usados durante a Segunda Guerra Mundial e a Guerra Fria, são particularmente vulneráveis a ataques deste tipo. O motivo é que os discos rotatórios utilizados produzem sons característicos que dependem dos dados em que o dispositivo opera. Estes sons podem ser medidos por um grampo de telefone, por exemplo. Um exemplo de tal ataque é descrito na autobiografia do ex-funcionário de contrainteligência da MI5 Peter Wright [WG88].

Em máquinas modernas, um grampo pode ser utilizado para cronometrar a duração do tempo entre o pressionamento de uma tecla e outra em um teclado [AA04, ZZT09]. Esta informação pode frequentemente ser usada para determinar o texto que foi digitado, desta forma vazando informação como, por exemplo, senhas [SWT01].

Recentemente, Genkin *et al.* [GST13] proporam um ataque acústico que funciona em sistemas operacionais executando em computadores pessoais. O ataque foi aplicado à popular implementação de RSA no *software* GnuPG. (Desde a publicação, a vulnerabilidade foi corrigida.)

3.5.3 *Probing*

Outra opção é utilizar pontas de prova para ler os valores em fios individuais no circuito—*probing attacks*. Este tipo de ataque é frequentemente negligenciado, por ser considerado custoso. Contudo, *probing attacks* são extremamente poderosos, uma vez que eles permitem uma observação direta da computação. Este tipo de ataque foi inicialmente considerado em um trabalho por Anderson e Kuhn [AK96, And08]; veja também [KK99] para uma discussão de *probing attacks* e possíveis estratégias de defesa.

3.5.4 Contramedidas

Ao longo do tempo, várias modificações de *hardware* com o intuito de proteger contra ataques de canal lateral tem sido propostas. Algumas delas—particularmente no caso de *wire probing*—envolvem blindar o *hardware* do ambiente, enquanto outras—como no caso de análise de consumo—possuem como ideia principal desacoplar a informação vazada de outros dados sendo processados.

Uma opção é utilizar um conjunto de portas em que cada operação consome uma quantidade fixa de energia. Isto é alcançado por *dual-rail pre-charge (DRP) logic* [TAV02]. Outra opção é injetar aleatoriedade no sistema. Esta técnica é conhecida como mascaramento (*masking*). Em sua forma mais simples, a data é mascarar os dados mediante a substituição de cada bit num registrador de dados pelo XOR do bit com um bit aleatório, isto é, cada bit b é substituído por $s = b \oplus r$ onde r é um bit aleatório. É claro que para "desmascarar" o valor, é necessário preservar tanto s quando r, mas a principal propriedade é que os dois são independentes de b quando considerados separadamente. Baseado nesta ideia simples, é possível projetar esquemas em que a quantidade de dados vazados é significativamente reduzida.

Também é possível combinar lógica *dual-rail* e mascaramento. Esta possibilidade é explorada em [PM05]. De fato, mascaramento é uma técnica bastante geral: ela é utilizada por vários esquemas teóricos para resistência a vazamento. Mascaramento também é utilizado implicitamente no esquema proposto neste trabalho, apesar de que, como veremos, chegaremos à técnica por uma rota bem diferente.

3.6 Computadores confiáveis

Um circuito quântico ruidoso $\mathcal{E}$ é uma implementação de uma transformação unitária U que atua nos subsistemas S (o subsistema de *dados*) e E (o *ambiente*). Uma vez que, na prática—atualmente e para o futuro imaginável—somente podemos esperar computar

em circuitos ruidosos, é desejável desenvolver um método para executar computações em um circuito ruidoso de maneira *confiável*, utilizando o circuito para executar operações codificadas de tal maneira que a qualquer ponto da computação, o estado condificado dos dados esteja próximo ao estado de um circuito ideal que implementa U perfeitamente.

Mais precisamente, seja $\mathcal{C}$ um circuito quântico com entradas e saídas em $\mathcal{H}_2^{\otimes k}$. Assumimos que o circuito atua em entradas clássicas; isto é feito preparando um estado conhecido $|0\rangle^{\otimes k}$ e posteriormente codificando uma entrada clássica x neste estado, obtendo o estado denotado por $|x\rangle$. Os estados de entrada possíveis $|x\rangle$ formam a base computacional. Assim, para qualquer x, o circuito implementa a ação da transformação U_x sobre $|0\rangle^{\otimes k}$.

Uma simulação confiável para $\mathcal{C}$ opera da seguinte maneira. Seja $\tilde{\mathcal{C}}$ um circuito quântico ruidoso que atua em $\mathcal{H}_2^{\otimes n}$. O circuito é sujeito a um ruído especificado por um *modelo de ruído* $\mathcal{N}$. O formato de modelos de ruído é definido na Seção 3.7. O subsistema de dados é inicializado no estado $|0\rangle^{\otimes n}$. Posteriormente, a entrada $x \in \{0,1\}^k$ é codificada em um código corretor de erros com tamanho n; denotamos a entrada codificada por $|\bar{x}\rangle$. A execução procede da mesma forma que no circuito ideal $\mathcal{C}$, com a diferença de que as portas são substituídas por portas codificadas (isto é, operações em dados codificados). Assim, podemos comparar os estados de $\mathcal{C}$ e $\tilde{\mathcal{C}}$ em um ponto arbitrário da computação. Adicionalmente, após cada passo da computação, executa-se um passo de correção de erros, com o objetivo de manter o estado do circuito no espaço do código. Dizemos que $\tilde{\mathcal{C}}$ é *ε-confiável* contra o ruído $\mathcal{N}$ se para cada passo, o estado de $\mathcal{C}$ é igual ao valor lógico do estado de $\tilde{\mathcal{C}}$, exceto com probabilidade ε.

Notamos que a maneira pela qual utilizamos confiabilidade é ligeiramente diferente do tratamento usual. Como definido acima (e igualmente ao tratamento comum), as entradas são codificadas como parte do circuito. Mas nossos circuitos também recebem entradas codificadas adicionais. Isto é feito puramente por conveniência e não torna a nossa definição mais forte.

3.7 Circuitos confiáveis são resistentes a vazamento

Nesta seção mostramos que para um circuito clássico arbitrário com vazamento de acordo com um modelo de vazamento L, podemos visualizar o circuito como um circuito quântico com um modelo de ruído correspondente $\mathcal{N}$, e que se o circuito quântico é confiável com respeito a $\mathcal{N}$ ele também é resistente a vazamento do tipo L. Neste sentido, primeiramente precisamos introduzir uma definição concreta para modelos de ruído. Um modelo de ruído quântico geral é uma operação $\mathcal{N}$ em estados quânti-

cos em $\mathcal{H}_2^{\otimes n}$ e toma a forma

$$\mathcal{N}(\rho) = \sum_k p_k E_k \rho E_k^\dagger, \tag{3.8}$$

onde E_k são operadores arbitrários que levam o espaço $\mathcal{H}_2^{\otimes n}$ a ele mesmo e $p_k \geq 0$ com $\sum_k p_k = 1$. Se E_k tem a forma $E_k = \bigotimes_{i=1}^n Z_i^{a_{i,k}}$ for $a_{i,k} \in \{0,1\}$ e Z dado por $Z|x\rangle = (-1)^x|x\rangle$, dizemos que $\mathcal{N}$ é um modelo de *ruído de fase*.

O lema a seguir relaciona modelos de vazamento clássico e modelos de ruído quântico.

Lema 2. *Para qualquer circuito $\mathcal{C}$ executando com um um pedido de vazamento $\{l_j, p_j\}_{j=1}^m$ tirado de um modelo de vazamento L, existe um modelo de ruído $\mathcal{N}$ tal que $\mathcal{C}$, visto como um circuito quântico, é um circuito sujeito ao ruído $\mathcal{N}$. Adicionalmente, $\mathcal{N}$ tem a forma*

$$\mathcal{N}(\rho) = \frac{1}{d} \sum_{j=1}^m p_j \sum_{k=0}^{d-1} F_j^k \rho (F_j^k)^\dagger, \tag{3.9}$$

onde F_j^k é o operador que leva $|s\rangle$ a $\omega^{k l_j(s)}|s\rangle$ e ω é uma raiz d-ésima primitiva da unidade.

Agora podemos estabelecer a conexão formal entre resistência a vazamentos e computação quântica confiável:

Teorema 2. *Para todo circuito $\mathcal{C}$ e modelo de vazamento L, existe um modelo de ruído $\mathcal{N}$ como especificado no Lema 2 tal que para todo implementação $\tilde{\mathcal{C}}$ de $\mathcal{C}$, ε-confiável contra $\mathcal{N}$, com função de codificação $y \mapsto |\bar{y}\rangle$, o protocolo $\pi = \{\pi_A, \pi_E\}$ tal que, dado $(y, \mathcal{C})$ como entrada, dá como saída $(|\bar{y}\rangle, \tilde{\mathcal{C}})$, e π_E é como especificado na Observação 1 da Definição 4, é um $2\sqrt{\varepsilon}$-compilador resistente a vazamentos contra L.*

Demonstração do Lema 2. Primeiro considere o caso em que L é tal que a cada rodada, uma única função de vazamento l é escolhida. Seja S o subsistema representando as atribuições a fios $s := W_{\mathcal{C}}(x, y)$ no circuito e seja E o subsistema de Eva. Em termos quânticos, a ação do vazamento corresponde à transformação

$$|s\rangle^S \otimes |0\rangle^E \quad \to \quad |s\rangle^S \otimes |l(s)\rangle^E \tag{3.10}$$

para cada s. Para determinar a ação do vazamento no sistema S, considere um estado arbitrário $|\psi\rangle^S = \sum_s \sqrt{p_s}|s\rangle^S$ para alguma distribuição de probabilidade p_s. Após aplicar a transformação e realizar o traço parcial no subsistema E, o sistema $\rho := |\psi\rangle\langle\psi|$ é

transformado em

$$\rho \quad \to \quad \mathcal{N}(\rho) = \sum_{s} \sum_{s' : l(s') = l(s)} \sqrt{p_s p_{s'}} |s\rangle\langle s'|, \tag{3.11}$$

uma vez que a coerência entre partes do sistema com valores diferentes de l é perdida. Mas este também é o estado resultante se a transformação fosse

$$|s\rangle^S \otimes |0\rangle^E \quad \to \quad \frac{1}{\sqrt{d}} \sum_k \omega^{kl(s)} |s\rangle^S \otimes |k\rangle^E = \frac{1}{\sqrt{d}} \sum_k F^k |s\rangle^S \otimes |k\rangle^E, \tag{3.12}$$

onde d é o tamanho da saída de l, ω é uma d-ésima raiz primitiva da unidade, e F^k é o operador que leva $|s\rangle$ a $\omega^{kl(s)}|s\rangle$. A equivalência entre as transformações (3.11) e (3.12) pode ser verificada tomando o traço parcial de E em (3.12). Logo, o modelo de ruído pode ser expresso como

$$\mathcal{N}(\rho) = \frac{1}{d} \sum_{k=0}^{d-1} F^k \rho (F^k)^\dagger. \tag{3.13}$$

Considere agora o caso em que a função de vazamento é escolhida probabilisticamente de um conjunto $\{l_j\}_{j=1}^m$, onde l_j é escolhido com probabilidade p_j. Neste caso, um modelo de ruído correspondente é uma combinação convexa de operadores de ruída da forma em (3.13). Isto é, podemos escrever

$$\mathcal{N}(\rho) = \frac{1}{d} \sum_{j=1}^{m} p_j \sum_{k=0}^{d-1} F_j^k \rho (F_j^k)^\dagger \tag{3.14}$$

onde F_j^k é o operador que leva $|s\rangle$ a $\omega^{kl_j(s)}|s\rangle$. □

O lema técnico a seguir é um caso especial do Teorema 3 em [KSW08]. O lema mostra que um canal próximo em operação (no sentido da *norma de operadores* $\|\cdot\|_\infty$) vaza uma quantidade de informação insignificante ao ambiente.[3]

Lema 3. *Seja $T_B \colon \mathcal{B}(\mathcal{H}_A) \to \mathcal{B}(\mathcal{H}_B)$ o canal quântico com entrada clássica e canal complementar $T_E \colon \mathcal{B}(\mathcal{H}_A) \to \mathcal{B}(\mathcal{H}_E)$, e seja $T \colon \mathcal{B}(\mathcal{H}_A) \to \mathcal{B}(\mathcal{H}_B)$ o "canal ideal" com $T(\rho) = U\rho U^*$, onde $U \colon \mathcal{H}_A \to \mathcal{H}_B$ is é uma operação unitária, e suponha que $\|T_B - T\|_\infty \leq \varepsilon$. Então temos*

$$\|T_E - S\|_\infty \leq 2\sqrt{\varepsilon}, \tag{3.15}$$

onde S é um canal completamente despolarizante, isto é, $S(\rho) := \dfrac{\mathrm{tr}(\rho)}{2} \mathbb{1}_E$.

[3] A mesma conclusão pode ser alcançada utilizando o princípio da incerteza derivado em [RS14].

Demonstração. Note primeiro que o canal ideal T não introduz ruído. Logo, podemos definir a operação inversa $T^{-1}(\rho) := U^*\rho U$ tal que $T^{-1}T = TT^{-1} = \mathbb{1}_A$. Agora, se $\|T_B - T\|_\infty \leq \varepsilon$, então $\|T^{-1}T_B - \mathbb{1}_A\|_\infty \leq \varepsilon$. Logo

$$\inf_D \|DT_B - \mathbb{1}_A\|_\infty \leq \|T^{-1}T_B - \mathbb{1}_A\|_\infty \leq \varepsilon, \tag{3.16}$$

Onde o ínfimo é tomado sobre todos os canais $D\colon \mathcal{B}(\mathcal{H}_B) \to \mathcal{B}(\mathcal{H}_A)$. Por [KSW08, Theorem 3][4], temos

$$\|T_E - S\|_\infty \leq \varepsilon. \tag{3.17}$$

(A entrada a T_B e T_E é clássica, logo as normas CB e de operador são equivalentes.) □

Demonstração do Teorema 2. Cada pedido de vazamento de L possui um modelo de ruído correspondente, dado pela expressão no Lema 2. A implementação confiável do circuito $\mathcal{C}$ inclui um circuito compilado $\overline{\mathcal{C}}$, assim como um método para codificar entradas. O conversor π_A recebe o segredo y acompanhado de um circuito como entradas, e dá como saída o segredo codificado $|\overline{y}\rangle$ e $\overline{\mathcal{C}}$. Seguindo a construção, entradas x recebidas de E são codificadas como $|\overline{x}\rangle$, e o circuito quântico é utilizado para computar $|\overline{\mathcal{C}(x,y)}\rangle$. Posteriormente, a saída decodificada é enviada de volta a E. Logo, temos um cenário como mostrado na Figura 3.3b.

Agora temos que mostra a existência de um simulador σ tal que a condição (3.6) (Figura 3.4) seja satisfeita. Provaremos que o simulador existe mostrando que para cada passo da computação, o vazamento recebido por E é independente do estado atual do circuito (isto é, os valores intermediários codificados por seus fios).

Pelo Lema 2, o modelo de vazamento resulta em ruído $\mathcal{N}$ no circuito quântico $\mathcal{C}$. Como $\mathcal{C}$ é ε-confiável contra $\mathcal{N}$, não pode haver erros na informação quântica codificada em qualquer ponto da computação, exceto com probabilidade ε—caso contrário, o estado do circuito confiável seria diferente do estado do circuito ideal. Podemos visualizar este cenário nos termos do Lema 3. O circuito ruidoso define um canal ruidoso T_B, e o canal complementar correspondente T_E descreve o canal (no pior caso, para o usuário legítimo) para o adversário. O circuito ideal corresponde a computar a transformação unitária U. Aplicando o Lema 3, temos que o estado a cada ponto do circuito, que denotamos por $|s\rangle$, é independente de $l(W_{\mathcal{C}}(x,y))$ (exceto com probabilidade $2\sqrt{\varepsilon}$).

Finalmente, notamos que para o caso de um adversário adaptativo, l' pode depender em entradas e saídas anteriores, assim como vazamentos anteriores, uma vez que a escolha de l pode depender destes valores. Porém, devido ao fato de que estes valores

[4]Note que, aqui, utilizamos a representação de Schrödinger, enquanto o artigo citado utiliza a representação de Heisenberg. Como estamos interessados somente em sistemas de dimensão finita, as duas descrições são equivalentes.

também são disponíveis ao simulador, e que l' é independente do valor lógico de s, l' pode ser gerado pelo simulador. □

Capítulo 4

Tolerância a falhas

Este capítulo explora em detalhe as principais técnicas por trás de circuitos tolerantes a falhas, usados para implementar computação confiável (definida na Seção 3.6). Na Seção 4.1, introduzimos as ideias básicas da área de tolerância a falhas em geral. Na Seção 4.2, exploramos o caso particular de códigos corretores de erros quânticos com distância 3, que serão utilizados em nossa implementação de circuitos resistentes a vazamento no Capítulo 5.

4.1 Circuitos tolerantes a falhas

Nossa implementação concreta de um esquema resistente a vazamento usa a construção dada em [AGP05]. Nesta seção, daremos um resumo das ideias básicas em tolerância a falhas e as técnicas utilizadas.

Uma implementação tolerante a falhas de um circuito ruidoso simula um circuito quântico ideal, isto é, um circuito onde erros não são possíveis. Os dados são primeiramente codificados em um código corretor de erros quânticos. Este cenário é o mesmo que o apresentado na Seção 3.6. Posteriormente, operações são executadas nos qubits lógicos. Estas operações serão denotadas *gadgets*. *Gadgets* simulam—utilizando apenas portas ruidosas—as portas do circuito ideal. Erros podem ser causados por uma porta defeituosa, e também podem ser introduzidos a fios no circuito. Devido ao fato de que, em última análise, estamos apenas interessados em vazamento dos fios, vamos considerar somente erros em fios.

Após a execução de um *gadget*, é executada a correção de erros. Um obstáculo com esta abordagem é que a correção não pode ser executada perfeitamente, devido ao fato de que, uma vez que todas as componentes do circuito são ruidosas, o passo de correção de erros também pode introduzir erros. Consequentemente, teremos que usar *gadgets* que se comportam bem na presença de erros; em particular, eles não devem

propagar erros excessivamente, de maneira a tornar a correção de erros difícil ou até impossível.

Como não é possível corrigir todos os erros perfeitamente, prosseguimos mediante o uso de concatenação. Mais precisamente, a simulação é feita de maneira recursiva, dividida em níveis. No primeiro nível, os *gadgets* utilizam componentes físicas e ruidosas. Como não podemos supor que a simulação é perfeita, estes *gadgets* também são ruidosos. No segundo nível, os *gadgets* do primeiro nível são tratados como portas físicas, a ser simulados da mesma maneira. Esta simulação recursiva prossegue até que um nível seja alcançado em que a probabilidade de erro—isto é, a probabilidade de que a simulação difira da execução ideal—é desprezível. Crucialmente, esta estratégia só pode ser bem-sucedida se a simulação no primeiro nível é menos ruidosa que o circuito físico. Esta possibilidade é positiva para uma variedade de modelos de ruído.

Grosso modo, o critério que utilizaremos para determinar a tolerância a falhas de *gadgets* é aplicar o *gadget* a um qubit lógico—ou a um grupo de qubits lógicos, se o *gadget* tem múltiplos qubits como entrada—com um número de erros corrigíveis e verificar se os erros na saída ainda são corrigíveis. Diremos que um *gadget* é *tolerante a falhas* se ele satisfaz esta propriedade. É claro que, se erros nas entradas não são corrigíveis, não há esperança de que a computação seja correta; se, no entanto, os erros são corrigíveis e o *gadget* é tolerante a falhas, então a computação será bem-sucedida, sob a hipótese de que o passo de correção de erros é executado corretamente.

A seguir, expandiremos essa ideia no caso específico das componentes que precisaremos implementar. Para um tratamento mais geral e formal em tolerância a falhas, veja [Got09].

4.1.1 Portas lógicas

Vamos aplicar esta ideia em termos mais concretos a portas lógicas. Seja $\mathcal{C}$ o espaço do código e, para um dado r, seja $\mathcal{C}_r$ o subespaço gerado por todos os estados da forma $Q|\psi\rangle$, onde $|\psi\rangle \in \mathcal{C}$ e Q é um erro de Pauli com peso de no máximo t. Para uma porta lógica com n entradas, temos a seguinte condição: se a entrada i está em $\mathcal{C}_{r_i}$ e o *gadget* para a porta tem s falhas, então a saída i está em $\mathcal{C}_r$ onde $r = s + \sum_i r_i$, supondo $s \leq t$. Esta condição garante que mesmo quando um erro em um bloco de entrada pode propagar a outros blocos, ele nunca pode propagar *dentro* de um bloco.

Note que esta condição não garante que todas as saídas estejam próximas à saída correta. Ela apenas garante que elas estão próximas ao espaço do código. É necessária uma outra condição: aplicar um *gadget* com s falhas às entradas consideradas no parágrafo anterior e então aplicar uma decodificação perfeita—isto é, supondo uma

correção de erros perfeita—deve ter o mesmo resultado que decodificar as entradas e aplicar a porta física (sem ruído) correspondente.

A principal ideia por trás de *gadgets* tolerantes a falhas é aplicar portas de maneira *transversal*. Mais precisamente, para *gadgets* com um qubit lógico como entrada, uma porta é aplicada a cada qubit físico no bloco. É fácil ver por que um *gadget* implementado desta forma é tolerante a falhas: uma vez que o *gadget* envolve somente portas físicas de um qubit, não pode haver propagação de erros dentro do bloco, logo a primeira condição é satisfeita.

E como satisfazer a segunda condição? Vimos (Seção 2.2.2) que para um $[[n, k, d]]$-código estabilizador com um conjunto estabilizador S, o grupo de operadores lógicos, dado por $N(S)/S$, é isomórfico a $\mathcal{P}_k$, o grupo de Pauli em k qubits. Ademais, como $N(S) \subset \mathcal{P}_n$, os operadores lógicos são produtos de operadores de Pauli. Isto mostra que operadores de Pauli lógicos podem ser implementados transversalmente.

Para portas com múltiplas entradas, a definição de transversalidade é que o *gadget* apenas utiliza portas físicas em que o i-ésimo qubit de um bloco apenas interage com o i-ésimo qubit de outros blocos, para todos os qubits físicos i. É fácil verificar que *gadgets* com esta propriedade são tolerantes a falhas.

Transversalidade pode ser verificada testando a porta lógica em cada entrada possível e observando que a operação desejada é executada. Para códigos estabilizadores, é possível também tomar a rota de verificar como o código é transformado sob conjugação pelo código transversal. Mais precisamente, a porta transversal deve preservar o estabilizador, e a ação nos operadores lógicos do código deve ser a mesma da porta a ser implementada.

Por exemplo, seja S o estabilizador para um $[[n, k, d]]$-código estabilizador e sjea F uma porta de 2 qubits, e $G = F^{\otimes n}$. Suponha que queremos mostrar que $G = \overline{F}$, isto é, G implementa a operação lógica F. Para este fim, verificamos se $G(S \otimes S)G^\dagger = S \otimes S$. Isto pode ser feito computando a ação em geradores do estabilizador individuais. Adicionalmente, temos que verificar se $G(P_1 \otimes P_2)G^\dagger = \overline{F}(P_1 \otimes P_2)\overline{F}^\dagger$, onde $P_1, P_2 \in \{\overline{X}_i, \overline{Z}_i\}_{i=1}^k$.

Embora portas transversais sejam uma ferramenta poderosa em implementações tolerantes a falhas, sabe-se que não existe um conjunto de portas que é simultaneamente transversal e universal [EK09]. Como consequência, se se deseja implementar um computador universal tolerante a falhas, é necessário implementar pelo menos uma porta que não é transversal, ou contornar o resultado do trabalho citado mediante a aplicação de operações não unitárias.

Condições para tolerância a falhas de uma porta lógica. Aqui descrevemos as condições para tolerância a falhas formalmente. Estas condições são tomadas de [AGP05, Section 9]. Na definição a seguir, utilizamos a noção de um *decodificador ideal*: um *gadget* conceitual que toma um estado codificado com um número arbitrário de erros, corrige-os, e dá como saída o estado decodificado.

Definição 5 Uma implementação lógica L de uma porte G que atua em n entradas codificadas de um QECC que corrige t erros é tolerante a falhas se as condições seguintes são satisfeitas: se a entrada i tem s_i erros, para $1 \leq i \leq n$, e L tem r falhas, com $s := r + \sum_{i=1}^{n} s_i \leq t$, então:

1. após L, cada saída tem no máximo t erros, e
2. aplicar primeiro L e então o decodificador ideal a cada saída dá os mesmos resultados que aplicar um decodificador ideal em cada entrad e então G nas saídas do decodificador.

A primeira condição é uma maneira formal de dizer que erros não são propagados excessivamente. A segunda condição é um condição mais técnica, necessária para garantir que a porta lógica implementa G corretamente mesmo na presença de falhas, embora ela possa introduzir erros aos estados codificados.

4.1.2 Correção de erros

Para um código estabilizador, para computar a síndrome é necessário medir os autovalores dos geradores do estabilizador. O estado $|\psi\rangle$ que codifica os dados está no autoespaço de cada gerador; como eles têm autovalores ± 1, para um gerador do estabilizador arbitrário g, temos $g|\psi\rangle = (-1)^a|\psi\rangle$ onde $a \in \{0, 1\}$ é o bit da síndrome. Se não há erros, então todos os bits da síndrome são 0.

Correção de erros de Shor. Uma forma tolerante a falhas de medir a síndrome é utilizar o esquema de correção de erros introduzido por Shor [Sho96]. Este método não será utilizado em nossa implementação, mas é um exemplo ilustrativo de problemas que podem surgir em circuitos ruidosos. Considere primeiro o seguinte procedimento para medir a síndrome[1]. Preparamos um estado auxiliar $|+\rangle$ e aplicamos g à entrada de dados. A aplicação é condicionada no valor (na base computacional) de $|+\rangle$. A

[1] O mesmo procedimento pode ser utilizado para medir um operador unitário arbitrário com autovalores ± 1; para mais detalhes, veja [Got09, Section 4.4].

saída (supondo que não tem falhas) é $|\psi\rangle(|0\rangle + (-1)^a|1\rangle)$. Depois, medimos o qubit auxiliar na base de fase. Se obtemos $|+\rangle$ então $a = 0$, e se obtemos $|-\rangle$ então $a = 1$.

Este circuito *não* é tolerante a falhas: uma falha no qubit auxiliar pode propagar para vários qubits no bloco de dados. No entanto, uma modificação simples o torna tolerante a falhas. Tendo em vista nossa discussão anterior sobre portas transversais, queremos que cada qubit no bloco de dados interaja com um qubit diferente. No esquema de Shor, isto é feito utilizando como estado auxiliar o chamado *cat state* $|\text{cat}_n\rangle = |0\rangle^{\otimes n} + |1\rangle^{\otimes n}$; o procedimento é conhecido como o método de *cat state*. O estado interage com os qubits de dados de maneira transversal, isto é, o i-ésimo qubit no *cat state* controla a componente de g que é aplicada ao o i-ésimo qubit. Desta maneira, para um $[[n, k, d]]$-código estabilizador, utilizamos um *cat state* de n qubits.

O caso para $n = 4$ é mostrado na Figura 4.1. A figura mostra o caso geral em que o operador $U := \bigotimes_{i=1}^{4}$ é medido; na correção de erros de Shor, U é um gerador do estabilizador.

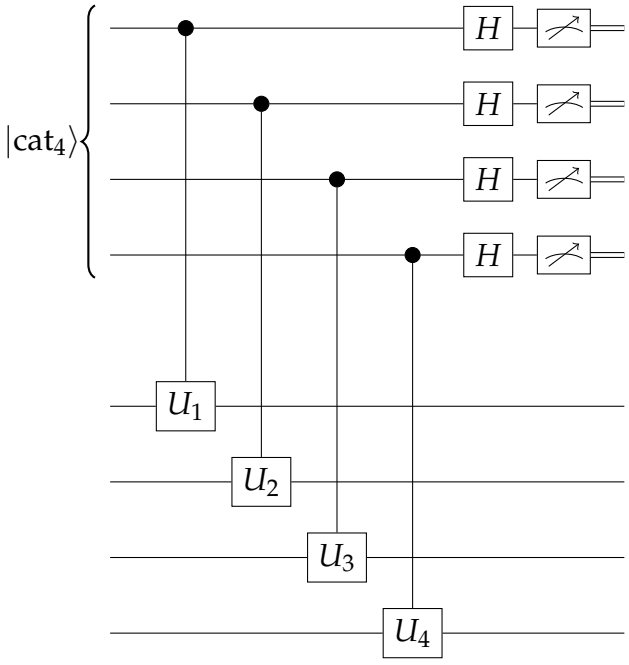


Figura 4.1: Medição do operador $U = \bigotimes_{i=1}^{4}$ com um *cat state*.

Ainda precisamos abordar a questão de se há uma falha simples no qubit auxiliar, a medição da síndrome dará um resultado incorreto. No entanto, podemos repetir a medição várias vezes, melhorando a acurácia a cada medição. Este procedimento funciona, mas é um pouco inconvenient por ser bastante ineficiente: para uma probabilidade de erro ε, são necessárias $O(\log 1/\varepsilon)$ tentativas [Sho96]. No entanto, para o propósito deste trabalho, pode-se assumir que as medições são perfeitas, então não é ne-

cessário repetir a medição da síndrome. Apesar da ineficiência do método de Shor em geral, ele tem sido usado em trabalhos recentes sobre tolerância a falhas, como [Got13].

Correção de erros de Steane. Para códigos CSS, é possível utilizar o método de correção de erros de Steane [Ste97]. Lembramos (Seção 2.2.3) que códigos CSS possuem estabilizadores tipo Z e tipo X, que são utilizados para corrigir erros X e erros Z, respectivamente. A correção de erros de Steane procede da seguinte forma. Suponha que podemos preparar um estado auxiliar $|+\rangle$ codificado, que denotamos por $|\overline{+}\rangle$. Executamos um CNOT codificado—o que, para códigos CSS, pode ser feito de maneira transversal—com o bloco de dados como controle e o estado auxiliar como alvo. Depois, medimos todos os qubits do bloco auxiliar na base computacional. Uma vez que a porta CNOT propaga erros de bit do controle para o alvo, o bloco auxiliar terá os mesmos erros de bit que o bloco de dados. Adicionalmente, como medir um estabilizador tipo Z (responsável por detectar erros de bit) em um código CSS se resume a adicionar os resultados (módulo 2) de um conjunto de medições na base computacional, os bits da síndrome podem ser recuperados medindo cada qubit na base computacional.[2]

Para corrigir erros de fase, aplicamos a mesma técnica, mas na base complementar. Mais precisamente, preparamos um estado auxiliar $|\bar{0}\rangle$, e executamos um CNOT lógico com o estado auxiliar como controle e o bloco de dados como alvo, antes de medir cada qubit do estado auxiliar na base de fase. A porta CNOT propaga erros de fase do alvo para o controle, e os resultados das medições na base de fase são utilizados para reconstruir os autovalores dos estabilizadores tipo X. O procedimento completo é mostrado na Figura 4.2.

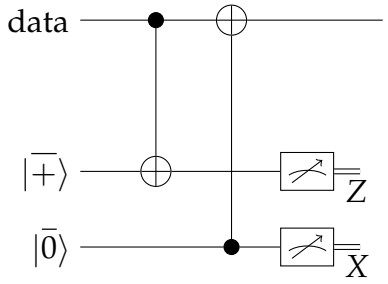


Figura 4.2: Correção de erros de Steane. As portas CNOT e as medições são executadas de maneira transversal.

Note que ainda não demos uma definição formal de tolerância a falhas para correção de erros em geral. Voltaremos a este ponto na próxima seção.

[2]Uma abordagem equivalente é visualizar a síndrome como verificações de paridade em um código clássico, que podem ser obtidas pelos resultados clássicos da medição.

Finalmente, notamos que se os dados interagem com o estado auxiliar apenas através de portas Clifford, não é necessário aplicar a recuperação imediatamente: como notado na Seção 2.1.3, uma vez que portas Clifford levam erros de Pauli a erros de Pauli, pode-se propagar os erros ao long do circuito e corrigir o erro obtido através deste processo ao fim do circuito. Pelo teorema de Gottesman-Knill [Got98a], este procedimento pode ser feito de forma eficiente com um computador clássico. Logo, esta técnica pode ser utilizada para melhorar a eficiência da implementação tolerante a falhas. Para portas fora do grupo de Clifford, esta técnica não pode ser usada; este fato será relevante quando a tradução clássica da implementação for feita no Capítulo 5.

4.1.3 Preparação de estados

A preparação de estados pode ser vista como um caso especial de uma porta lógica em que não há entrada. A condição para tolerância a falhas é reduzida à seguinte condição:

Definição 6 Um *gadget* de preparação de estados que dá como saída estados codificados em um QECC que corrige t erros é dito tolerante a falhas se ele satisfaz a seguinte condição: se o *gadget* tem $r \leq t$ falhas, então sua saída tem no máximo r erros.

Preparação de estados tolerante a falhas pode ser executada começando com um estado arbitrário no qual é aplicada medição de síndrome, seguida pela medição de um operador lógico. O procedimento de medição de síndrome projeta o estado no espaço de código, e a medição do operador lógico projeta o estado no autoespaço do operador. O resultado da medição nos dá o autovalor, isto é, nos diz qual estado foi preparado. Por exemplo, ao medir $\overline{Z}$, temos $|\bar{0}\rangle$ ou $|\bar{1}\rangle$. Um estado arbitrário pode ser obtido mediante a aplicação de uma operação codificada.

Assim, preparação de estados é reduzida ao problema de correção de erros. No entanto, como vimos na Seção 4.1.2, a correção de erros requer a preparação de estados auxiliares. Em particular, o método de Steane utiliza $|\bar{0}\rangle$ e $|\overline{+}\rangle$. À primeira vista, parece que temos um problema de ovo-e-galinha. Felizmente, existem técnicas para resolvê-lo, explorando o fato de que a correção de erros apenas necessita da preparação de estados auxiliares, cujo uso não é necessário no restante da computação.

Considere a preparação do *cat state* $|\mathrm{cat}_n\rangle = |0\rangle^{\otimes n} + |1\rangle^{\otimes n}$. Notamos primeiro que erros de fase não acumulam—dois erros de fase são tão ruins quanto um. Erros de bit, por outro lado, podem propagar excessivamente. Para este fim, utilizamos a seguinte estratégia. Usamos um qubit auxiliar para testar se dois qubits são iguais, como

mostrado na Figura 4.3. Se o *cat state* foi preparado corretamente, então eles devem ser o mesmo, independente de qual par foi escolhido. Portanto, se eles são diferentes, descartamos o estado. Repetindo o procedimento um número suficientemente alto de vezes, podemos ter certeza de que o *cat state* preparado está correto.

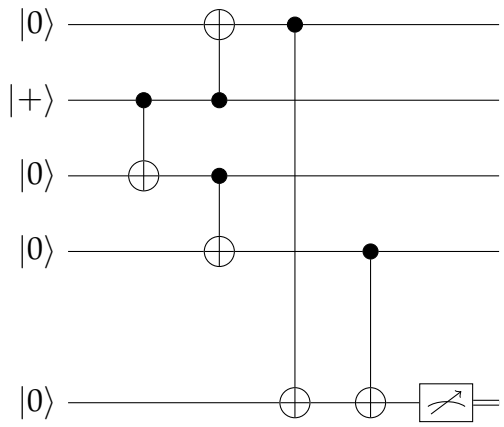
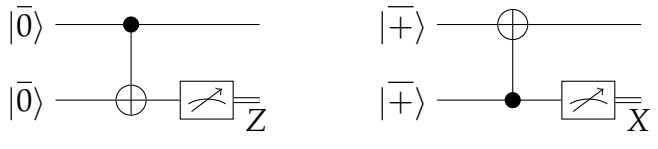


Figura 4.3: Verificação do *cat state*.

Para correção de erros de Steane, precisamos preparar $|\bar{0}\rangle$ and $|\overline{+}\rangle$. Lembramos que $|\bar{0}\rangle$ é usado como controle para um CNOT codificado, e depois medido na base de fase. Assim, os únicos erros que podem propagar do estado auxiliar para o bloco de dados são erros de bit. Estes erros podem ser detectados mediante a execução de um passo de verificação, antes da interação com o bloco de dados. A verificação é feita preparando um segundo estado auxiliar $|\bar{0}\rangle$, seguido de um CNOT com o primeiro estado auxiliar como controle e o estado auxiliar de verificação como alvo. Posteriormente, o estado de verificação é medido na base computacional. Se encontrarmos erros, descartamos o estado auxiliar. Assim como no caso do *cat state*, para garantir a preparação correta do estado, repetimos este procedimento um número alto de vezes.

Para $|\overline{+}\rangle$, um procedimento similar é utilizado, mas na base complementar, como mostrado na Figura 4.4. Após o passo de verificação, os estados são utilizados no circuito de correção de erros, como mostrado na Figura 4.2.



(a) (b)

Figura 4.4: Verificação de estados auxiliares utilizados na correção de erros de Steane ($|\bar{0}\rangle$ em (a), e $|\overline{+}\rangle$ em (b)).

Decodificação de estados auxiliares. Além de considerações de eficiência, estes procedimentos possuem possuem a grande desvantagem de serem *não determinísticos*. Esta propriedade é particularmente ruim para nossa aplicação de tolerância a falhas, porque queremos circuitos clássicos que são confiáveis na presença de erros de fase, e devido ao fato de que erros de fase nunca aparecem em circuitos clássicos, nossos circuitos nunca podem realizar decisões condicionadas nos resultados de medições que não podem ser realizadas. Esta objeção não torna nossa tarefa necessariamente impossível, uma vez que não precisamos destes resultados para realizar computação. Não bostante, precisamos da capabilidade de realizar verificação de estados auxiliares de maneira determinística.

Tal esquema foi proposto em [DA07]. No trabalho citado, verificação determinística de estados auxiliares é desenvolvida como parte de uma investigação de tolerância a falhas com medições lentas—em ambos métodos que vimos aqui, o restante da computação, a efetividade da implementação é massivamente reduzida se não temos acesso a medições rápidas. A solução elimina indeterminismo, de uma maneira que pode ser aplicada ao nosso cenário. Aqui, apresentamos um resumo da abordagem tratada no trabalho citado.

A ideia é executar a verificação de estados auxiliares apenas depois de sua interação com os dados. Adicionalmente, em vez de descartar toda a computação no caso em que um número excessivo de erros é encontrado—como consequência natural do fato de que preparação de estados auxiliares é feito de uma maneira que não é tolerante a falhas—, o esquema corrige estes erros. Isto é possível pelo seguinte motivo. Intuitivamente, o objetivo da tolerância a falhas é prevenir que falhas simples que causam erros supostos corrigíveis—a partir da suposição de que estamos usando um QECC com tal propriedade—propaguem em erros que não são mais corrigíveis. Mas como estados auxiliares são estados fixos, preparados com circuitos conhecidos, é possível retraçar erros não-corrigíveis na saída aos erros originais, e recuperar os erros corrigíveis causados por falhas.

Primeiro, apresentamos o procedimento para um QECC geral que pode corrigir t erros. Posteriormente, discutimos o caso do código de Steane. Seguindo a terminologia de [DA07], chamaremos o método de *decodificação de estados auxiliares*. (No artigo citado, o termo *ancilla verification*—verificação de estados auxiliares—é utilizado somente para os procedimentos não determinísticos.)

O esquema é mostrado na Figura 4.5. Primeiro, o estado auxiliar é preparado com um circuito de codificação (não tolerante a falhas) Enc. Depois, o estado interage com o bloco de dados. Posteriormente, a informação de amplitude (isto é, da base computacional) é copiada a $2t$ cópias de $|0\rangle^n$; isto é, ela é codificada em um código clássico de

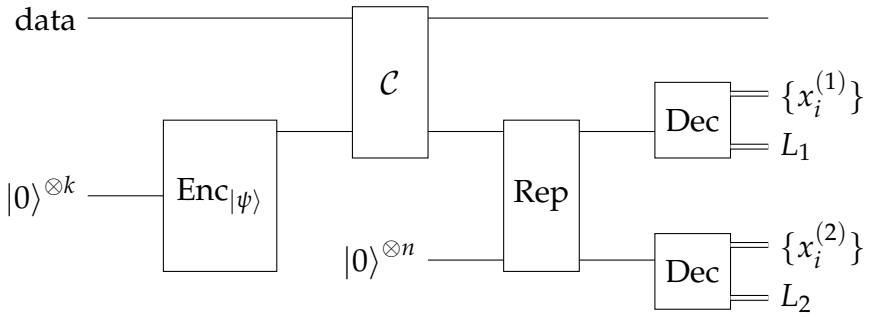


Figura 4.5: Decodificação de estados auxiliares no caso $t = 1$.

repetição de $2t$ bits. Esta codificação é representada por Rep na figura. Por clareza, o caso particular de $t = 1$ é mostrado, em que Rep é simplesmente um CNOT do estado auxiliar codificado a $|0\rangle^n$; no caso geral, há $2t - 1$ CNOTs adicionais. Cada um deles tem o estado auxiliar codificado como controle e um estado fresco como alvo.

Depois, cada estado codificado é decodificado de forma separada, revertendo a operação de Enc. Decodificar a j-ésima cópia dá como saída os padrões de síndrome $s_j := \{x_i^{(j)}\}_i$, e os autovalores de um operador lógico L_j. Os padrões da síndrome e os autovalores podem ser vistos como palavras código do código clássico de repetição. Isto é, escrevemos $\overline{x} := \{s_j\}_j, \overline{L} := \{L_j\}_j$. Devido às possíveis falhas no procedimento de decodificação, as palavras código podem ter erros. Elas são decodificadas por meio de um voto de maioria.

Note que os padrões da os padrões de síndrome não vão necessariamente diagnosticar erros que afetam menos que t qubits. De fato, uma vez que o procedimento de codificação não é tolerante a falhas, um conjunto de menos que t falhas pode propagar excessivamente, resultando em um estado auxiliar com mais de t erros. A ideia é que projetamos os procedimentos de codificação e decodificação de maneira que o conjunto de síndromes possíveis nos permite distinguir entre as falhas possíveis, assim fazendo os erros corrigíveis.

Para provar que o método funciona, temos que mostrar que a execução da correção de erros utilizando estados auxiliares preparados desta maneira é tolerante a falhas. A seguinte definição para tolerância a falhas para correção segue a dada em [AGP05, Section 9].

Definição 7 Um esquema de correção de erros (EC) que corrige t falhas é tolerante a falhas se ele satisfaz as seguintes condições:

1. se EC $r \leq t$ falhas, ele leva uma entrada arbitrária a uma saída que difere do espaço do código em no máximo r erros.

2. se a entrada tem s erros e EC tem r falhas com $s + r \leq t$, a saída tem no máximo s erros.

A segunda propriedade é garantida pela forma que a decodificação é feita. O código clássico de repetição é aplicado para distinguir entre falhas na codificação e falhas na decodificação. A única maneira de a síndrome ser incorreta é se houvesse pelo menos t decodificadores com falhas. Mas como supomos que o circuito tem no máximo t falhas, o único caso com o qual temos que nos preocupar é se há exatamente t decodificadores com falhas. Mas neste caso sabemos que por hipótese não há falhas no codificador, então não é necessário fazer correção de erros.

A primeira propriedade é satisfeita se a aplicação da correção de erros da maneira tradicional, utilizando verificação não determinista, também a satisfaz. Este é o caso para os métodos de Shor e de Steane. Estes métodos também satisfazem a propriedade se usarmos verificação determinística, pelo motivo exposto a seguir. Supondo que a correção de erros tradicional satisfaz a primeira propriedade, podemos substituir a verificação não determinística por decodificação do estado auxiliar. Em ambos os casos, supomos que há r falhas. A única diferença entre os dois cenários é que no segundo caso, não podemos garantir que o estado auxiliar não possui mais que r erros, que então podem propagar excessivamente para o bloco de dados, de forma que o bloco estará a mais de r erros do espaço do código. No entanto, através da aplicação da decodificação do estado auxiliar, podemos retraçar estes erros às falhas originais, cuja quantidade é (por hipótese) de no máximo r. Assim, se o procedimento original satisfaz a primeira propriedade, ela também a satisfaz se a verificação de estados auxiliares é substituída pelo esquema de decodificação.

4.1.4 Medição

Do ponto de vista de tolerância a falhas, medições podem ser vistas como um caso especial de uma porta lógica, onde a saída é clássica. A condição para tolerância a falhas para um *gadget* de medição pode ser definida da seguinte forma.

Definição 8 Um *gadget* de medição para um QECC que corrige t erros é dito tolerante a falhas se ele satisfaz a seguinte condição: se a entrada tem r erros e a medição tem s falhas com $r + s \leq t$, então ele dá os mesmos resultados que uma medição ideal.

Como observado na Seção 4.1.2, uma variante do método de Shor pode ser utilizado para medir um operador unitário arbitrário, simplesmente substituindo os estabilizadores a serem medidos pelos operadores desejados, os quais neste caso seriam os operadores lógicos $\overline{Z}$ e $\overline{X}$.

No caso do código CSS, existe um procedimento particularmente simples para computar a medição tolerante a falhas de $\overline{Z}$ e $\overline{X}$. Como vimos na Seção 2.2.4, os operadores lógicos são operadores de Pauli do tipo Z ou X. Sem perda de generalidade, podemos supor que $\overline{Z}$ é tipo Z e $\overline{X}$ é tipo X. (Caso contrário, podemos simplesmente dar novos rótulos às palavras código.)

Assim, na medição de $\overline{Z}$, precisamos nos preocupar com erros de bit, porque estes são os erros que podem alterar o autovalor de $\overline{Z}$. Mas para um código CSS, estes erros são detectados pela medição dos estabilizadores tipo Z. Logo, se quisermos medir $\overline{Z}$, podemos simplesmente medir todos os qubits na base computacional. A partir dos resultados da medição, obtemos o valor de $\overline{Z}$, e ademais podemos utilizá-los para computar os autovalores dos estabilizadores tipo Z. Note que esta operação pode ser feita classicamente. Sob a hipótese razoável de que operações clássicas podem ser feitas de maneira confiável, e dado que a parte quântica da medição descrita é efetivamente uma operação transversal, o procedimento é tolerante a falhas.

4.2 Tolerância a falhas com códigos de distância 3

Nesta seção, apresentamos um resumo da construção em [AGP05]. Desta construção derivaremos um esquema concreto resistente a vazamentos (Capítulo 5). Devido ao fato de que a implementação completa não é necessária para resistência a vazamentos, discutiremos apenas a componentes utilizadas.

O código utilizado é o código de Steane $[[7,1,3]]$ concatenado. Abaixo, apresentamos os *gadgets* utilizados no primeiro nível de concatenação. Posteriormente, vamos explicar brevemente como eles podem ser utilizados junto com a concatenação de códigos para alcançar computação confiável.

4.2.1 Portas lógicas

Lembre-se que, intuitivamente, uma porta lógica é tolerante a falhas se ela não propaga erros excessivamente. Para códigos de distância 3—que corrigem 1 erro—, tolerância a falhas para portas lógicas é reduzida a um conjunto de condições particularmente simples. As condições para tolerância a falhas são dadas abaixo. Elas são baseadas nas definições de [AGP05, Section 2].

Definição 9 Seja L uma porta lógica para um código que corrige 1 erro. Dizemos que L é tolerante a falhas se satisfaz as seguintes condições:

1. Se L não tem falhas, aplicar L a uma entrada com no máximo mmmm erro produz uma saída com no máximo um erro em cada bloco, e
2. se L tem no máximo uma falha, aplicá-lo a uma entrada sem erros produz uma saída com no máximo um erro em cada bloco.

É fácil mostrar que portas transversais satisfazem estas condições. Para a primeira condição, transversalidade implica que um erro pode propagar a no máximo um qubit por bloco. Para a segunda condição, uma falha em uma porta pode causar erros apenas nos qubits aos quais a porta é aplicada, isto é, no máximo um qubit por bloco.

As portas lógicas que usamos são X, Z, CNOT, CZ e Toffoli. Como mostraremos abaixo, exceto pela porta Toffoli, todas estas portas são transversais.

Como observado na Seção 4.1.1, para um $[[n,k,d]]$-código estabilizador arbitrário, os operadores lógicos $\overline{X}_i, \overline{Z}_i$ (para $i = 1,\ldots,k$) estão em $\mathcal{P}_n$. Isto é, eles podem ser implementados por portas transversais. Em particular, para o código de Steane (Seção 2.2.4), é fácil verificar que $\overline{X} = X^{\otimes 7}, \overline{Z} = Z^{\otimes 7}$.

Transversalidade *não* vale para a porta CNOT em geral, mas podemos mostrar que ela é transversal para qualquer $[[n,k,d]]$-código CSS, isto é, $\overline{\text{CNOT}} = \text{CNOT}^{\otimes n}$. Pela Seção 4.1.1, temos que mostrar que

$$\text{CNOT}^{\otimes n}(S \otimes S)(\text{CNOT}^{\otimes n})^\dagger = S \otimes S \tag{4.1}$$

$$\text{CNOT}^{\otimes n}(P_1 \otimes P_2)(\text{CNOT}^{\otimes n})^\dagger = \overline{\text{CNOT}}(P_1 \otimes P_2)\overline{\text{CNOT}} \tag{4.2}$$

onde $P_1, P_2 \in \{\overline{X}_i, \overline{Z}_i\}_{i=1}^k$.

Para provar (4.1), apenas temos que mostrar que a função f que envia $g_1 \otimes g_2 \in S \otimes S$ a $\text{CNOT}^{\otimes n}(g_1 \otimes g_2)(\text{CNOT}^{\otimes n})^\dagger$ é uma permutação em $S \otimes S$. Neste sentido, note que

$$\text{CNOT}(X \otimes \mathbb{1})\text{CNOT}^\dagger = X \otimes X \qquad \text{CNOT}(\mathbb{1} \otimes X)\text{CNOT}^\dagger = \mathbb{1} \otimes X \tag{4.3}$$

$$\text{CNOT}(\mathbb{1} \otimes Z)\text{CNOT}^\dagger = Z \otimes Z \qquad \text{CNOT}(Z \otimes \mathbb{1})\text{CNOT}^\dagger = Z \otimes \mathbb{1}. \tag{4.4}$$

Agora, note que para um código CSS, os geradores dos estabilizadores são produtos de operadores X ou Z. Por (4.3) e (4.4), temos $f(g_X \otimes \mathbb{1}) = g_X \otimes g_X, f(\mathbb{1} \otimes g_X) = \mathbb{1} \otimes g_X$ se g_X é um gerador tipo X, e $f(\mathbb{1} \otimes g_Z) = g_Z \otimes g_Z, f(g_Z \otimes \mathbb{1}) = g_Z \otimes \mathbb{1}$ se g_Z é um gerador tipo Z.

Escrevendo um elemento arbitrário $g \in S \otimes S$ como um produto de geradores do estabilizador, é fácil ver a partir destas relações que $f(g) \in S \otimes S$, e também que existe $h \in S \otimes S$ tal que $f(h) = g$. Isto prova (4.1).

Para os operadores lógicos, como $\overline{X}, \overline{Z}$ podem ser escolhidos como produtos de operadores X e Z, respectivamente, aplicar (4.3) e (4.4) nos dá (4.2).

Omitomos a prova de que CZ é transversal, mas notamos que ela pode ser facilmente derivada da demonstração apresentada acima utilizando o fato de que $\mathrm{CZ} = (\mathbb{1} \otimes H)\mathrm{CNOT}(\mathbb{1} \otimes H)$.

4.2.2 Porta Toffoli

A porta Toffoli é mais difícil de implementar, devido ao fato de que ela não é transversal. No entanto, veremos que ela pode ser implementada por um circuito com portas transversais, medições—cuja tolerância a falhas será discutida na Seção 4.2.3—e um estado auxiliar especial, que chamaremos de *estado de Toffoli*.

Antes de apresentar o circuito, introduziremos uma maneira de executar operadores lógicos utilizando medições. A teoria completa é concretizada em mais detalhes em [Got98b]; veja também [Gai08] para a sua aplicação na porta Toffoli.

Seja $|\psi\rangle$ um estado estabilizado por um conjunto S de operadores unitários que comutam mutuamente—isto é, para quaisquer $g_1, g_2 \in S$, temos $[g_1, g_2] = 0$. Aqui, estes operadores *não* precisam estar no grupo de Pauli. Seja agora $g \in S$ e suponha que exista um operador A que anticomuta com g Podemos utilizar este fato para transformar $|\psi\rangle$ em um autoestado de A com autovalor $+1$. Isto é feito da seguinte maneira. Primeiro, medimos A. Se o resultado da medição é $+1$, não há mais o que fazer; caso contrário, aplicamos g ao estado pós-medição.

Este procedimento funciona porque, como descrito na Seção 2.1.6, o estado pós-medição $|\psi'\rangle$ é dado por $|\psi\rangle_+ := P_+|\psi\rangle$ ou $|\psi\rangle_- := P_-|\psi\rangle$, onde $P_\pm := (\mathbb{1} \pm A)/2|\psi\rangle$, dependendo do resultado da medição. Se o estado é $|\psi\rangle_+$, temos

$$A|\psi\rangle_+ = \frac{A + A^2}{2}|\psi\rangle = \frac{A + \mathbb{1}}{2}|\psi\rangle = |\psi\rangle_+. \tag{4.5}$$

Caso contrário a aplicação de g nos dá

$$g|\psi\rangle_- = \frac{g - gA}{2}|\psi\rangle = \frac{\mathbb{1} + A}{2}g|\psi\rangle = P_+|\psi\rangle = |\psi\rangle_+. \tag{4.6}$$

O que acontece a operadores lógicos, isto é, elementos de $N(S)/S$? Se $L \in N(S)/S$ comuta com A, então autoestados de L ainda serão autoestados de L após o procedimento. Se, por outro lado, L anticomuta com A, então em geral o estado final não será

autoestado de L, mas um cálculo simples mostra que ele será um autoestado de gL com o mesmo autovalor. Logo, o procedimento efetivamente leva L a gL[3].

Agora estamos prontos a abordar o problema de executar a porta Toffoli. A ideia é primeiro calcular a ação da porta Toffoli nos operadores lógicos. Então, começando das entradas e usando o estado de Toffoli, produzimos um estado que é estabilizado por um conjunto de operadores tal que, após aplicar o procedimento de medição descrito nos parágrafos anteriores, os operadores lógicos são transformados na maneira apropriada.

Seja T a porta Toffoli. A ação de T em um operador lógico L pode ser obitda computando a ação de $TLT^\dagger$ em um estado da base computacional e usando linearidade. Por exemplo, temos

$$\begin{aligned} TX_1|x,y,z\rangle &= T|(x\oplus 1),y,z\rangle && (4.7)\\ &= |(x\oplus 1),y,z\oplus xy\oplus y\rangle && (4.8)\\ &= (\mathrm{CNOT}_{2\to 3}X_1)T|x,y,z\rangle, && (4.9)\end{aligned}$$

logo $TX_1T^\dagger = \mathrm{CNOT}_{2\to 3}X_1$. Repetindo isto para todo operador lógico, obtemos:

$$\begin{aligned} TX_1T^\dagger &= \mathrm{CNOT}_{2\to 3}X_1 \qquad TZ_1T^\dagger = Z_1 && (4.10)\\ TX_2T^\dagger &= \mathrm{CNOT}_{1\to 3}X_2 \qquad TZ_2T^\dagger = Z_2 && (4.11)\\ TX_3T^\dagger &= X_3 \qquad TZ_3T^\dagger = \mathrm{CZ}_{1\to 2}Z_3 && (4.12)\end{aligned}$$

O estado de Toffoli é dado por $|\Theta\rangle = |000\rangle + |100\rangle + |010\rangle + |111\rangle$. Este estado é estabilizado por $g_1 = \mathrm{CNOT}_{2\to 3}X_1$, $g_1 = \mathrm{CNOT}_{1\to 3}X_2$, $g_3 = \mathrm{CZ}_{1\to 2}Z_3$. Seja $|\psi\rangle$ um estado arbitrário em 3 partes. Na Figura 4.6, a entrada é dada por $|xyz\rangle$, mas trataremos o caso geral diretamente. Logo o estado produto $|\phi\rangle := |\Theta\rangle|\psi\rangle$ tem o mesmo conjunto de estabilizadores. Adicionalmente, os operadores lógicos para $|\phi\rangle$ são $\overline{X}_i = X_{i+3}, \overline{Z}_i = Z_{i+3}$ para $i = 1,\ldots,3$, onde X_j, Z_j denota a aplicação de X ou Z ao j-ésimo qubit de $|\phi\rangle$, respectivamente.

[3]Note que como não nos restringimos a operadores de Pauli, é possível que o operador lógico não comuta ou anticomuta com A, mas não precisamos considerar esta possibilidade aqui.

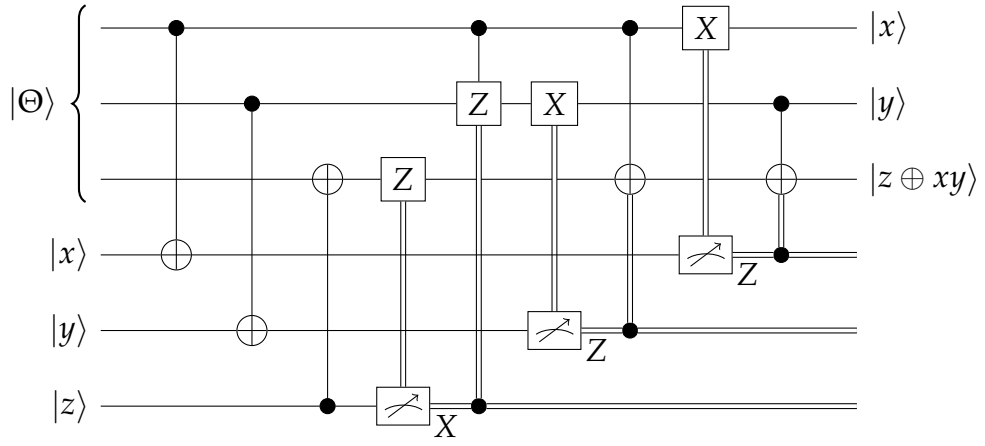


Figura 4.6: Porta Toffoli aplicada ao estado da base computacional $|xyz\rangle$.

Após a aplicação de $\text{CNOT}_{1\to4}, \text{CNOT}_{2\to5}$ e $\text{CNOT}_{6\to3}$, os novos estabilizadores são

$$g_1' = \text{CNOT}_{2\to3} X_1 X_4 \tag{4.13}$$
$$g_2' = \text{CNOT}_{1\to3} X_2 X_5 \tag{4.14}$$
$$g_3' = \text{CZ}_{1\to2} Z_3 Z_6, \tag{4.15}$$

e os operadores lógicos são

$$\overline{X}_1' = X_4 \qquad \overline{Z}_1' = Z_1 Z_4 \tag{4.16}$$
$$\overline{X}_2' = X_5 \qquad \overline{Z}_2' = Z_2 Z_5 \tag{4.17}$$
$$\overline{X}_3' = X_3 X_6 \qquad \overline{Z}_3' = Z_6 \tag{4.18}$$

Posteriormente, medimos Z_4, Z_5 e X_6. Note que estes operadores anticomutam com os estabilizadores g_1', g_2' e g_3', respectivamente. Utilizando o procedimento de medição descrito, podemos forçar o estado a um autoestado dos operadores com autovalor $+1$, aplicando os estabilizadores condicionados no resultado da medição: se obtemos -1, então aplicamos o estabilizador. Este passo é mostrado no circuito da Figura 4.6. Logo, o estado final tem estabilizadores $g_1'' = Z_4, g_2'' = Z_5$ e $g_3'' = X_6$.

Ademais, note que Z_4, Z_5 e X_6 comutam com todos os operadores lógicos, com três exceções: Z_4 anticomuta com $\overline{X}_1'$, Z_5 anticomuta com $\overline{X}_2'$ e X_6 anticomuta $\overline{Z}_3'$. Como

resultado, os operadores lógicos são atualizados a

$$\overline{X}_1'' = g_1' X_4 = \mathrm{CNOT}_{2\to 3} X_1 \qquad \overline{Z}_1'' = Z_1 \tag{4.19}$$

$$\overline{X}_2' = g_2' X_5 = \mathrm{CNOT}_{1\to 3} X_2 \qquad \overline{Z}_2' = Z_2 \tag{4.20}$$

$$\overline{X}_3' = X_3 \qquad \overline{Z}_3' = g_3' Z_6 = \mathrm{CZ}_{1\to 2} Z_3 \tag{4.21}$$

(Eliminamos aqui as ocorrências de Z_4, Z_5 e X_6, uma vez que eles estabilizam o estado final.) Como esta é a ação desejada nos operadores lógicos, o circuito da Figura 4.6 implementa a porta Toffoli.

4.2.3 Medições

Para um código de distância 3, a Definição 8 é reduzida às seguintes condições:

1. Se a entrada tem um erro e o *gadget* de medição não tem falhas, ele dá os mesmos resultados que uma medição ideal.
2. Se a entrada não tem erros e o *gadget* de medição tem uma falha, ele dá os mesmos resultados que uma medição ideal.

Em geral, uma simples medição destrutiva não é suficiente para garantir a segunda condição, mas existe um procedimento que funciona para um código estabilizador geral e um operador geral, utilizando o método *cat state* da correção de erros de Shor desenvolvido na Seção 4.1.2 (Figure 4.1). Ali, as medições têm de ser repetidas diversas vezes, para executar a correção de erros de maneira tolerante a falhas; mas aqui, é suficiente repetir as medições três vezes, executando correção de erros (tolerante a falhas) após cada medição. Tomamos o resultado do *gadget* de medição como o vencedor da votação de maioria entre as três medições.

É fácil ver que a primeira condição é satisfeita; para verificar que o procedimento também satisfaz a segunda condição, note primeiro que se há uma falha, ela deve estar em uma das três medições ou em um dos dois *gadgets* de correção de erros. Se a falha está na primeira medição, a saída terá no máximo um erro, uma vez que o método *cat state* funciona transversalmente. Então, o estado será corrigido pelo *gadget* de correção de erros, e, uma vez que o restante do circuit não tem falhas, os dois outros resultados estarão corretos. Um argumento similar pode ser feito para o caso em que a falha está na segunda ou terceira medição.

Se, por outro lado, a falha aparece em um dos dois *gadgets* de correção de erros, então, pela hipótese de tolerância a falhas, a sua saída terá no máximo um erro. Isto é, o resultado da medição que segue a correção de erros estará incorreta. Mas em

todo caso, os dois outros resultados estarão corretos. Portanto a segunda condição é satisfeita.

4.2.4 Preparação de estados

Pela Definição 6, um *gadget* de preparação de estados tolerante a falhas para o código de Steane deve preparar estados que têm no máximo um erro. Os estados que queremos preparar são $|\bar{0}\rangle$, $|\overline{+}\rangle$, o "estado de Shor" $|\text{even}_7\rangle$, e o estado de Toffoli $|\overline{\Theta}\rangle$. Também poderíamos obter $|\overline{+}\rangle$ a partir de $|\bar{0}\rangle$, mais vamos apresentar esta preparação como um procedimento separado porque mais tarde será mais fácil visualizar como executar a tradução clássica.

Notamos que, exceto por $|\bar{0}\rangle$, todos estes estados são usados apenas como *estados auxiliares*; isto é, eles são utilizados somente para executar algum procedimento. Mais precisamente, utilizamos $|\bar{0}\rangle$ e $|\overline{+}\rangle$ para correção de erros de Steane, e $|\overline{+}\rangle$ e $|\text{even}_7\rangle$ no procedimento de preparação para $|\overline{\Theta}\rangle$. Esta distinção é importante, porque para estados auxiliares podemos utilizar o método de decodificação de estados auxiliares da Seção 4.1.3.

No entanto, para $|\bar{0}\rangle$, há um uso adicional: utilizamos este estado para codificar entradas. Isto pode ser feito simplesmente executando um CNOT com a entrada (clássica) como controle e $|\bar{0}\rangle$ como alvo. Para preparar $|\bar{0}\rangle$, podemos aplicar o método de correção de erros de Steane a uma entrada arbitrária, como discutido em 4.1.3, seguido por uma medição de $\overline{Z}$. Após executar a correção de erros, projetamos o estado no espaço do código, e o resultado da medição de $\overline{Z}$ nos diz se a preparação foi de $|\bar{0}\rangle$ ou de $|\bar{1}\rangle$.

Medir $\overline{Z}$ exige preparar um *cat state*; no entanto, note que, para o código de Steane, $\overline{Z} = Z^{\otimes 7}$. Pela Figura 4.1.2, medir $\overline{Z}$ em uma entrada $|\psi\rangle$ se resume a medir os primeiros 7 qubits do estado$|\Psi\rangle := (H^{\otimes 7} \otimes \mathbb{1})\text{CZ}_{i,i+7}(|\text{cat}_7\rangle|\psi\rangle)$. Devido à equivalência $\text{CZ}_{12} = (H \otimes \mathbb{1})\text{CNOT}_{2\to 1}(H \otimes \mathbb{1})$ (Seção 2.1.2), temos

$$|\Psi\rangle = (H^{\otimes 7} \otimes \mathbb{1})(H^{\otimes 7} \otimes \mathbb{1}) \bigotimes_{i=1}^{7} \text{CNOT}_{i+7\to i}(H^{\otimes 7} \otimes \mathbb{1})|\text{cat}_7\rangle|\psi\rangle \tag{4.22}$$

$$= \bigotimes_{i=1}^{7} \text{CNOT}_{i+7\to i}|\text{even}_7\rangle|\psi\rangle. \tag{4.23}$$

(Lembre-se que $|\text{even}_n\rangle = H^{\otimes n}|\text{cat}_n\rangle$.) Portanto, a medição é equivalente a preparar um estado de Shor, executar um CNOT com a entrada como controle e o estado de Shor como alvo e depois medir o estado de Shor. O circuito completo é mostrado na Figura 4.7.

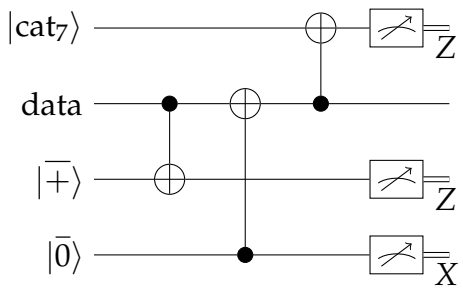


Figura 4.7: Preparação de $|\overline{0}\rangle$. Todos os CNOTs e medições são feitas transversalmente.

Preparação de estados auxiliares. Utilizamos o método de decodificação de estados auxiliares discutidos na Seção 4.1.3. Na realidade, como mostrado em [DA07], no caso do código de Steane é possível simplificar o esquema da Figura 4.5.

O circuito é mostrado na Figura 4.8. O circuito para $\text{Enc}_{|0\rangle}$ é mostrado na Figura 4.9a, enquanto o circuito de decodificação $\text{Dec}_{|0\rangle}$ é mostrado na Figura 4.9b.

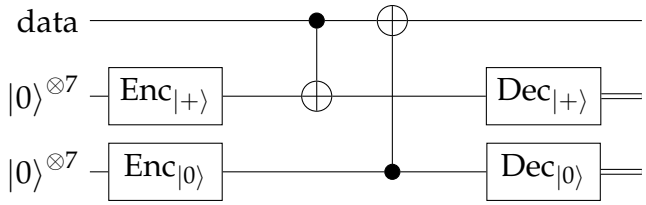
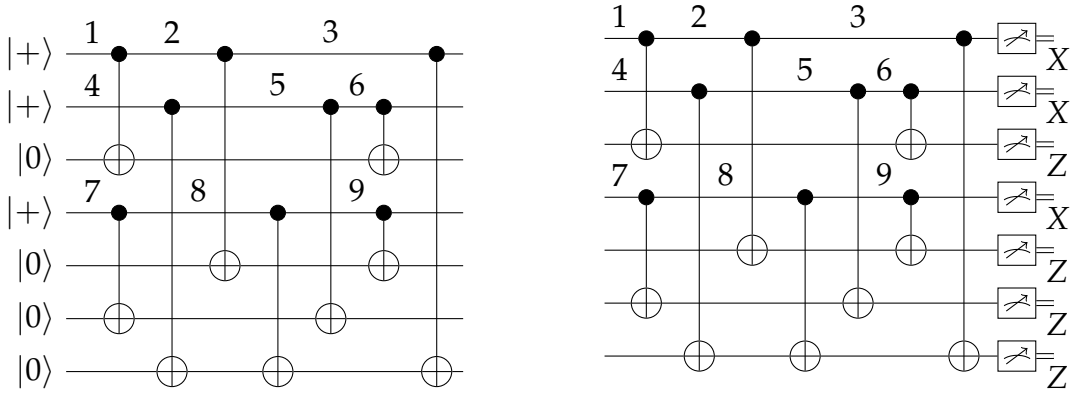


Figura 4.8: Correção de erros de Steane sem verificação.

Primeiro, note que as saídas das medições X na Figura 4.9b dão os autovalores dos estabilizadores tipo X, de forma que se erros de fase no bloco de dados podem ser diagnosticados e corrigidos. Logo, se não há falhas, o procedimento de correção de erros funciona corretamente.

Para mostrar que o esquema é tolerante a falhas, temos que mostrar que cada falha simples no circuito causa um erro (possivelmente envolvendo múltiplos qubits) que podem ser diagnosticados após a decodificação. Lembre-se (Seção 4.1.3) de que, para $|\overline{0}\rangle$, somente precisamos nos preocupar com erros de bit. Logo, uma vez que o circuito da Figura 4.9a contém apenas portas CNOT, que propagam erros de bit a erros de bit e erros de fase a erros de fase, somente temos que procurar por posições no circuito onde um erro de bit simples propaga de forma a causar múltiplos erros de bit.

Uma vez que a porta CNOT propaga erros de bit apenas de controle a alvo, não é necessário se preocupar com falhas no terceiro, quinto, sexto ou sétimo fio, uma vez que estes fios nunca são utilizados como controle. Como consequência, falhas nestes fios somente podem causar erros em um qubit. Assim, restam apenas as posições na Figura 4.9a numeradas de 1 a 9. Note primeiro que erros de bit nas posições 1, 4 ou 7 não causam nenhum erro, porque $|+\rangle$ é um autoestado de X. Os outros padrões de



(a) (b)

Figura 4.9: (a): Circuito de codificação para $|\overline{0}\rangle$ no código de Steane. O circuito que codifica $|\overline{+}\rangle$ é o mesmo, com a diferença de que alvos e controles são revertidos, e estados $|0\rangle$ são revertidos em $|+\rangle$ e vice-versa. (b): Circuito de codificação para $|\overline{0}\rangle$ no código de Steane. Por construção ,o resultado da medição Z no terceiro qubit dá o autovalor do operador lógico $\overline{Z} = Z_1Z_2Z_3$. O circuito de decodificação $|\overline{+}\rangle$ é o mesmo, com a diferença de que alvos e controles são revertidos e medições na base Z são revertidos em medições na base X e vice-versa.

erro são os seguintes:

$$2 : X_1X_5X_7 = X_3 \tag{4.24}$$

$$3 : X_1X_7 \tag{4.25}$$

$$5 : X_2X_3X_6 = X_7 \tag{4.26}$$

$$6 : X_2X_3 \tag{4.27}$$

$$8 : X_4X_5X_7 = X_6 \tag{4.28}$$

$$9 : X_4X_5, \tag{4.29}$$

onde as igualdades se devem ao fato de que $X_1X_3X_5X_7$, $X_2X_3X_6X_7$ e $X_4X_5X_6X_7$ são estabilizadores para o código de Steane. Logo, os únicos erros de múltiplos qubits possíveis são X_1X_7, X_2X_3 e X_4X_5. Propagando estes padrões através do circuito na Figura 4.9b nos dá as síndromes

$$X_1X_7 \to X_3X_5 \tag{4.30}$$

$$X_2X_3 \to X_6X_7 \tag{4.31}$$

$$X_4X_5 \to X_6X_7. \tag{4.32}$$

(Uma vez que X_1, X_2 e X_4 são medidos na base X, eles podem ser descartados.) Note que X_2X_3 e X_4X_5 geram a mesma síndrome, mas devido ao fato de que $X_2X_3X_4X_5$ é

um estabilizador do código, o mesmo operador de recuperação pode ser aplicado com sucesso a ambos, como notado na Seção 2.2.1.

Para provar tolerância a falhas, temos também que mostrar que nenhuma falha simple no circuito de decodificação pode levar o procedimento de recuperação a introduzir múltiplos erros no bloco de dados. Isto pode acontecer somente se uma falha, propagada através do circuito, causa um dos padrões em (4.30)–(4.32). Na Figura 4.9b, as posições numeradas indicam falhas que podem ser propagadas a múltiplos erros. Os padrões obtidos são

$$1 : X_3X_5X_7 \tag{4.33}$$
$$2 : X_5X_7 \tag{4.34}$$
$$3 : X_7 \tag{4.35}$$
$$4 : X_3X_6X_7 \tag{4.36}$$
$$5 : X_3X_6 \tag{4.37}$$
$$6 : X_3 \tag{4.38}$$
$$7 : X_5X_6X_7 \tag{4.39}$$
$$8 : X_5X_7 \tag{4.40}$$
$$9 : X_5 \tag{4.41}$$

Logo, podemos distinguir entre falhas simples no codificador e falhas no decodificador.

Para completar a descrição, os circuitos de codificação e decodificação para $|\overline{+}\rangle$ na Figura 4.10a e na Figura 4.10b, respectivamente. Eles podem ser obtidos a partir dos circuitos para $|\bar{0}\rangle$ trocando $|0\rangle$ por $|+\rangle$ (e vice-versa) e controle por alvo nas portas CNOT (e vice-versa), e medições Z por medições X (e vice-versa). Como resultado, podemos reutilizar a análise feita assima, uma vez que erros de fase propagam da mesma forma que erros de bit propagam no caso de $|\bar{0}\rangle$.

Estado de Shor. O estado de Shor em n qubits $|\text{even}_n\rangle$ é dado pela superposição uniforme de todos os estados $|i\rangle$ onde i é uma string binária de n bits com peso par. O estado pode ser obtido a partir do *cat state* em n qubits (introduzid na Seção 4.1.3) aplicando a porta Hadamard transversalmente; isto é, $|\text{even}_n\rangle = H^{\otimes n}|\text{cat}_n\rangle$. Este estado será utilizado na preparação no estado auxiliar de Toffoli, necessário para a implementação da porta Toffoli (Section 4.2.2).

Utilizamos o método de decodificação de estados auxiliares para preparar o estado de Shor de 7 qubits $|\text{even}_7\rangle$. O circuito de codificação é mostrado na Figura 4.11a.

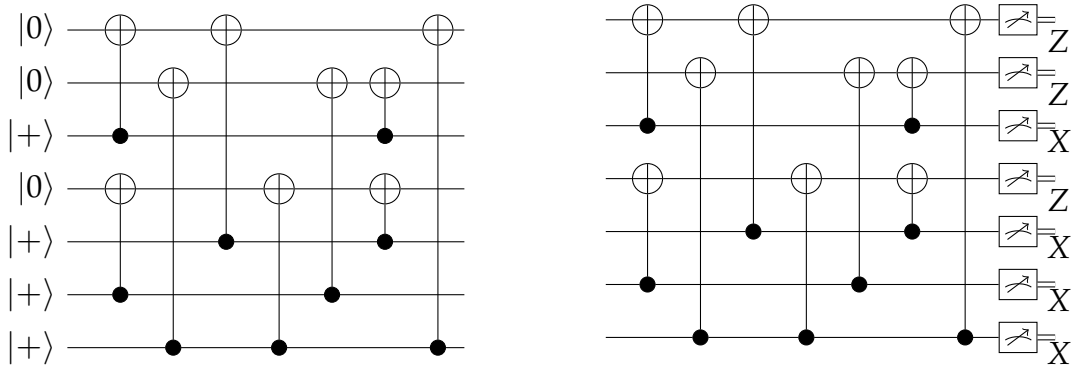


Figura 4.10: Encoding (a) and decoding (b) circuits for $|\overline{+}\rangle$ in the $[[7,1,3]]$ code.

Depois de interagir com o bloco de dados, medimos sua síndrome utilizando o circuito na Figura 4.11b de forma a diagnosticar uma possível falha na preparação. Estes circuitos são baseados nos circuitos apresentados em [DA07], onde o caso de $|\text{cat}_7\rangle$ é apresentado.

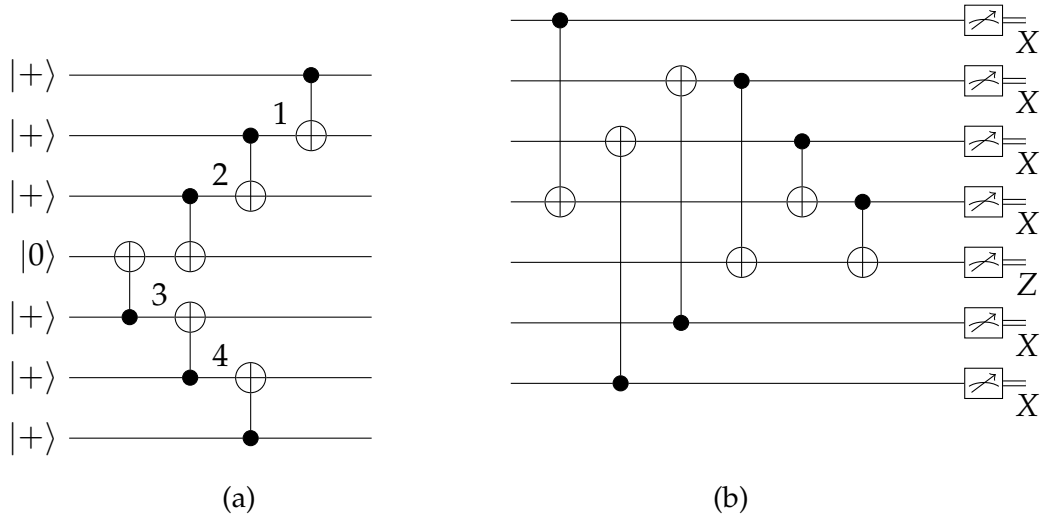


Figura 4.11: (a) Preparação do estado de Shor $|\text{even}_7\rangle$. Os números indicam posições onde os erros introduzidos causam múltiplos erros na saída. (b) Medição da síndrome para o estado de Shor.

É fácil verificar que se não há falhas, o circuito da Figura 4.11a prepara o estado corretamente. Como fizemos na preparação de $|\overline{0}\rangle$ e $|\overline{+}\rangle$, para obter tolerância a falhas, precisamos mostrar que existe um procedimento de decodificação tal que todo tipo de erro possível introduzido na codificação do estado de Shor leva a diferentes síndromes, que podem então ser utilizadas para corrigir erros que podem ter sido causados no circuito.

Primeiro, vamos analisar todos os possíveis padrões de erros Z que resultam de uma falha simples que pode ocorrer durante a codificação. Esses erros são mostrados pelos números na Figura 4.11a. Todas as outras possibilidades levam a um erro simples na saída, ou ao mesmo padrão de erros que um dos erros numerados causaria.

Os padrões de erro, na ordem dada pela figura, são Z_1Z_2, $Z_1Z_2Z_3$, $Z_5Z_6Z_7$ and Z_6Z_7. O circuito de decodificação, mostrado na Figura 4.11b, propaga estes erros a $Z_1Z_2Z_6$, $Z_1Z_2Z_3Z_6Z_7$, $Z_2Z_4Z_6Z_7$ e Z_6Z_7 respectivamente. Logo, todos os padrões de erro possíveis dão síndromes diferentes. Como consequência, estes erros são corrigíveis.

Estado de Toffoli. O estado de Toffoli é preparado utilizando o circuito da Figura 4.12. Todas as portas do circuito são executadas de maneira transversal, logo o circuito é tolerante a falhas.

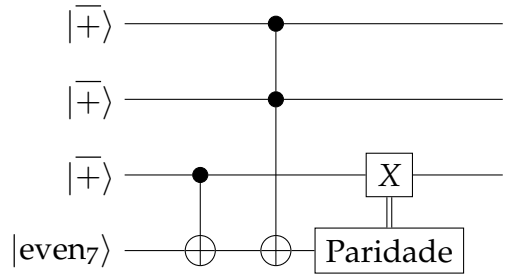


Figura 4.12: Preparação do estado de Toffoli. A caixa rotulada "Paridade" corresponde a medir a entrada na base Z de maneira transversal, seguida de computar a paridade da string de saída.

Para mostrar que o circuito prepara o estado de Toffoli $|\overline{\Theta}\rangle$, é útil visualizar $|\text{even}_7\rangle$ como uma palavra código de um QECC. Mais concretamente, $|\text{even}_7\rangle$ codifica 0, enquanto o estado "ímpar" $|\text{odd}_7\rangle$, dado pela superposição uniforme de todos os $|i\rangle$ onde i é uma string binária de 7 bits com peso ímpar, codifica 1.

É fácil verificar que se o CNOT transversal e a porta Toffoli no circuito da Figura 4.12 executam as operações codificadas corretamente, então o circuito prepara $|\overline{\Theta}\rangle$:

$$|\overline{+++}\rangle|\text{even}_7\rangle \to |\overline{+}\rangle|\overline{+}\rangle(|\bar{0}\rangle|\text{even}_7\rangle + |\bar{1}\rangle|\text{odd}_7\rangle) \tag{4.42}$$

$$\to (|\overline{000}\rangle + |\overline{100}\rangle + |\overline{010}\rangle + |\overline{111}\rangle)|\text{even}_7\rangle + \tag{4.43}$$

$$+ (|\overline{001}\rangle + |\overline{101}\rangle + |\overline{011}\rangle + |\overline{110}\rangle)|\text{odd}_7\rangle \tag{4.44}$$

$$= |\overline{\Theta}\rangle|\text{even}_7\rangle + \overline{X}_3|\overline{\Theta}\rangle|\text{odd}_7\rangle, \tag{4.45}$$

onde as duas flechas se referem à aplicação do CNOT transversal e à aplicação da porta Toffoli indicadas no circuito, respectivamente. Assim, a partir de uma medição da paridade da quarta entrada e, no caso em que o resultado da medição é ímpar (correspondendo a $|\text{odd}_7\rangle$), a aplicação de uma porta X codificada na terceira entrada produz $|\overline{\Theta}\rangle$.

Para mostrar que o CNOT transversal executa a operação codificada, precisamos verificar que a saída no alvo do CNOT é $|\text{even}_7\rangle$ se o controle é $|\overline{0}\rangle$ e $|\text{odd}_7\rangle$ se o controle é $|\overline{1}\rangle$, e que a porta Toffoli dá como saída $|\text{odd}_7\rangle$ se, e somente se, ambos os qubits de controle são $|\overline{1}\rangle$, e $|\text{even}_7\rangle$ caso contrário. Para este fim, precisamos utilizar uma propriedade do código de Steane. Mais precisamente, $|\overline{0}\rangle$ é uma superposição uniforme das strings de peso par: as palavras código do $(7,4)$-código de Hamming. Por outro lado, $|\overline{1}\rangle$ é uma superposição uniforme das palavras código de peso ímpar.

Agora, para uma palavra código de Hamming fixa x com peso par, podemos mostrar que $\text{CNOT}_{1\to 2}^{\otimes 7}|x\rangle|\text{even}_7\rangle = |\overline{x}\rangle|\text{even}_7\rangle$, da seguinte maneira. Note que a função $f_x(y) := x \oplus y$ é injetora: se $y_1 \neq y_2$, então $f_x(y_1) \neq f_x(y_2)$. Logo, como a função leva o conjunto de strings de peso par a ele mesmo, a função é uma permutação. Por linearidade, temos $\text{CNOT}_{1\to 2}^{\otimes 7}|\overline{0}\rangle|\text{even}_7\rangle = |\overline{0}\rangle|\text{even}_7\rangle$.

Para o caso de $|\overline{1}\rangle$, podemos usar um argumento similar. Embora a função f_x (onde, agora, x é uma palavra código de Hamming de peso ímpar) não seja mais uma permutação, dado que ela leva strings de peso par a strings de peso ímpar, ela ainda é uma bijeção, porque o conjunto de strings de peso par e o conjunto de strings de peso ímpar têm o mesmo tamanho. Isto prova nossa afirmação de que o CNOT transversal executa a operação codificada.

Para a porta Toffoli, temos uma situação parecida, mas note agora que o alvo não é mais uma palavra código de Hamming, e sim o *produto* de duas palavras. Mostraremos que o produto transversal de $|\overline{x}\rangle$ e $|\overline{y}\rangle$ é uma superposição de strings de peso ímpar se, e somente se, $x = y = 1$. Caso contrário, ela é uma superposição de strings de peso par. Com este fato, podemos simplesmente reutilizar os argumentos dos parágrafos anteriores.

O argumento chave para mostrar esta afirmação é que o código de Steane se baseia em um código clássico $\mathcal{C}$ que contém o seu próprio dual, $\mathcal{C}^{\perp}$[4]. Seja H a matriz de paridade do código $\mathcal{C}$. Temos $H\mathbf{c} = 0$ para todo $\mathbf{c} \in \mathcal{C}$. Isto é, toda palavra código é ortogonal a toda linha H_i de H. Uma vez que $\mathcal{C}^{\perp} \subset \mathcal{C}$, este fato também se aplica às palavras código de $\mathcal{C}^{\perp}$; em particular ele se aplica a H_i. Isto é, cada H_i é ortogonal a si mesmo, o que somente pode ocorrer se H_i tem peso par. Portanto, todas as palavras código em $\mathcal{C}^{\perp}$ têm peso par. Ademais, elas possuem sobreposição par—isto é, possuem

[4]Veja a Seção 2.2.3 para uma revisão de códigos corretores de erro clássicos.

1 na mesma posição em um número par de posições—com cada H_i (caso contrário, eles não seriam ortogonais), e, consequentemente, entre elas mesmas.

Logo, $|\bar{0}\rangle$ é a superposição de todas as palavras código com peso par de $\mathcal{C}$, as quais possuem sobreposição par entre elas mesmas, pelo argumento acima. O estado $|\bar{1}\rangle$ é a superposição uniformedas palavras código em $\mathcal{C} - \mathcal{C}^{\perp}$; todas estas palavras têm peso ímpar. Uma vez que elas também satisfazem a verificação de paridade dada por H, elas também possuem sobreposição par com as linha de H, e portanto também com as palavras código em $|\bar{0}\rangle$. Mas elas possuem sobreposição *ímpar* com elas mesmas. Para provar este fato, seja $x, y \in \mathcal{C} - \mathcal{C}^{\perp}$. Se x e y tivessem sobreposição par, então $x + y \in \mathcal{C}^{\perp}$ (por ter peso par), e portanto x e $x + y$ teriam sobreposição par—uma contradição. Isto prova nossa afirmação.

4.2.5 O *Accuracy Threshold Theorem*

Nesta seção, exploramos a questão de como combinar *gadgets* de tolerância a falhas para alcançar computação confiável. Primeiro, precisamos estabelecer alguns pontos gerais. Precisamos de uma família de códigos, cada uma com um conjunto apropriado de portas lógicas. Os códigos são parametrizados por um inteiro n. Utilizamos os códigos em um protocolo. O objetivo é mostrar que para um modelo específico de ruído, o protocolo dá saídas próximas às saídas dadas por um circuito ideal, tal que o erro se aproxima de 0 com $n \to \infty$.

Para tornar as coisas mais concretas, apresentamos aqui uma implementação para o código de Steane concatenado e o modelo de ruído independente. Neste modelo, para cada posição no circuito, uma falha é inserida com uma probabilidade fixa. A família de códigos é obtida a partir do código de Steane via concatenação. Assim, as portas lógicas apresentadas na Seção 4.2.1 funcionam para todo código neste família. Depois, apresentaremos um argumento informal para justificar a confiabilidade da implementação. Para uma demonstração rigorosa, veja [AGP05].

O fato de que estamos utilizando um código concatenado torna o argumento particularmente simples, uma vez que somente precisamos analisar o primeiro nível da concatenação: é fácil generalizar o argumento para um nível arbitrário k. Adicionalmente, como consideramos um código de distância 3, não há muitos casos a considerar na demonstração, o que torna a análise mais simples.

Utilizaremos a noção de um *retângulo* e de um *retângulo estendido*. Seguindo a terminologia de [AGP05], eles serão denotados por 1-Rec e 1-exRec, respectivamente. Um 1-Rec consiste na aplicação de uma porta lógica a uma entrada codificada (possivelmente com múltiplos blocos), e posteriormente aplicar correção de erros a cada bloco

da saída. Um 1-exRec consiste em aplicar correção de erros a cada bloco da entrada, e depois aplicar um 1-Rec.

Um esboço da demonstração é o seguinte. Dizemos que um 1-exRec é *bom* se ele possui no máximo uma falha, e que um 1-Rec é *correto* se aplicá-lo a uma entrada com no máximo um erro por bloco produz uma saída com no máximo um erro por bloco. É fácil mostrar que se todos os 1-Rec's são corretos, a implementação é confiável. Além disso, também podemos mostrar que se um 1-exRec é bom, então o 1-Rec contido nele é correto. Uma vez que um 1-exRec não é bom se, e somente se, ele tem pelo menos duas falhas, isto reduz a probabilidade de falha a $O(\varepsilon^2)$ (onde a constante depende do tamanho do 1-exRec). Portanto, se quisermos generalizar esta afirmação a um nível arbitrário k de concatenação, podemos reduzir a probabilidade de fracasso[5] a qualquer nível desejado.

Vamos redefinir as condições da Definição 7 e da Definição 9, especificamente para o caso de um código de distância 3. Aqui, EC se refere ao *gadget* de correção de erros, e L se refere a um *gadget* para uma operação lógica arbitrária.

1. Se EC não tem falhas, ele leva uma entrada arbitrária a uma palavra código.
2. Se EC tem uma falha, ele leva uma entrada arbitrária a uma saída que difere de uma palavra código em no máximo um erro.
3. Se EC não tem falhas, ele leva uma entrada com no máximo um erro a uma saída sem erros.
4. Se EC tem no máximo uma falha, ele leva uma entrada sem erros a uma saída com no máximo um erro.
5. Se L não tem falhas, aplicá-lo a uma entrada com no máximo um erro produz uma saída com no máximo um erro em cada bloco.
6. Se L tem no máximo uma falha, aplicá-lo a uma entrada sem erros produz uma saída com no máximo um erro em cada bloco.

Antes de provar que o 1-Rec contido em um 1-exRec bom é correto, vamos primeiro visualizar como este fato é utilizado. Suponha que todos os 1-exRec's são bons. A implementação inicia preparando estados codificados. Cada um dos *gadgets* de preparação pode ser visto como um caso especial de uma porta lógica, onde não há entradas. Pela propriedade 6 indicada acima, cada bloco de saída tem no máximo um erro. Posteriormente, a cada passo do circuito, um 1-Rec é aplicado a cada bloco. (No caso em

[5] A probabilidade de fracasso é a probabilidade de que o circuito dê um resultado errado, isto é, diferente do circuito ideal. Utilizamos esta terminologia para diferenciar este tipo de erro de uma falha física numa componente do circuito.

que nenhuma operação é aplicada a um qubit lógico, EC é aplicado, o que corresponde ao 1-Rec para a porta identidade.) Uma vez que todo 1-Rec é correto, todo bloco de saída terá no máximo um erro. Ao final da computação, cada bloco é medido, produzindo a saída final, juntamente com a informação da síndrome, que então pode ser utilizada para corrigir o erro.

Agora podemos seguir com a demonstração. Afirmamos a seguinte proposição.

Proposição 1. *Se um 1-exRec tem no máximo uma falha, o 1-Rec contido nele leva entradas com no máximo um erro a saídas com no máximo um erro.*

Demonstração. Lembre-se que um 1-exRec é bom se ele tem no máximo uma falha. Esta falha pode estar em um dos ECs que são aplicados diretamente à entrada, ou no 1-Rec. No primeiro caso, temos que, pela propriedade 2 acima, cada entrada ao 1-Rec está a no máximo um erro de uma palavra código. Uma vez que o 1-Rec não tem falhas, temos que, pelas propriedades 3 e 5, a saída do 1-Rec não tem erros.[6]

No segundo caso, nenhum dos ECs aplicados à entrada tem falhas. Logo, pela propriedade 1, cada uma das saídas é uma palavra código. Para o 1-Rec, temos duas possibilidades: ou a porta lógica L tem uma falha, ou um dos ECs aplicados à saída da porta tem uma falha. Se L tem uma falha, como nenhuma de suas entradas tem erros, pela propriedade 6 temos que a saída tem no máximo um erro por bloco. Então, pela propriedade 3, a saída dos ECs aplicados à saída da porta lógica não tem erros. Se, por outro lado, um destes ECs tem uma falha, então, pela propriedade 4, um dos blocos de saída terá no máximo um erro. Isto prova a proposição. □

Como generalizamos a noção de bom e correto a níveis arbitrários de concatenação? Primeiro, precisamos introduzir *gadgets* de nível k. Os *gadgets* executam as operações de porta lógica, correção de erros, preparação de estados e medição. Um *gadget* de nível 2 pode ser construído a partir de um *gadget* de nível 1 mediante a substituição de toda porta física utilizada no *gadget* de nível 1 pelo seu 1-Rec correspondente. Em geral, um *gadget* de nível k é construído a partir de um *gadget* de nível $(k-1)$ substituindo cada porta física por seu 1-Rec correspondente. A seguir, falaremos de k-Ga's e k-ECs. Estes são os *gadgets* de nível k para portas lógicas e correção de erros, respectivamente. Um k-Rec é dado por um k-Ga seguido por um k-EC em cada bloco de saída, e um k-exRec é dado por um k-EC em cada bloco de entrada seguido por um k-Rec.

A definição de um *bom* k-exRec não é completamente trivial. Intuitivamente, gostaríamos de dizer que um k-exRec é ruim (isto é, não é bom) se ele tem dois $(k-1)$-exRec's. Isto é, um $(k-1)$-exrec ruim é o análogo (em nível k) a uma falha em um

[6]Note que, neste argumento, é irrelevante se a saída dos ECs aplicados à entrada é correta. O que importa é que a saída do 1-Rec tenha no máximo um erro relativo à sua entrada.

1-exRec. O problema é que para $k > 1$, $(k-1)$-exRec's podem se sobrepor. Mais precisamente, os ECs aplicados à saída de um exRec são os mesmos ECs aplicados à entrada do exRec subsequente. Assim, $(k-1)$-exRec's ruins não são eventos independentes. Isto não é um problema para $k = 1$ porque por hipótese, falhas são independentes. Este fato é crucial no argumento de que a implementação de nível 1 reduz a probabilidade de fracasso a $O(\varepsilon^2)$. Assim, necessitamos de uma noção de bom e ruim que incorpora a independência. Não apresentaremos as definições formais aqui, ou uma prova de que k-exRec's bons contêm k-Rec's corretos[7], mas discutiremos brevemente a intuição por trás das definições generalizadas.

Queremos dizer que um k-exRec é ruim se ele contém dois $(k-1)$-exRec's ruins *independentemente*. Em que ocasiões isto pode ocorrer para dois $(k-1)$-exRec's que se sobrepõem? Seja $k = 2$ e considere o caso de dois 1-exRec's. Um cenário possível é o seguinte: existem duas falhas no primeiro 1-exRec, e nenhum deles está nos ECs aplicados à saída, e duas outras falhas no segundo 1-exRec, e nenhum deles está nos ECs aplicados à entrada. Neste caso, ambos 1-exRec's têm duas falhas, e nenhum deles está nos ECs que se sobrepõem. Neste caso, podemos dizer que os dois eventos são independentes.

E se há falhas nos ECs que se sobrepõem? Considere um exemplo onde ambos 1-exRec's têm 2 falhas, mas um deles está num EC sobreposto; isto é, há 3 falhas em total. Esta situação é mostrada na Figura 4.13. É fácil ver que se excluirmos EC_2 do primeiro 1-exRec, então 1-Ga_1 estará correto. Logo, podemos dizer que o primeiro 1-exRec é *bom* quando os ECs sobrepostos são removidos. Logo, na verdade temos apenas um 1-exRec ruim: o segundo 1-exRec, que inclui EC_2.

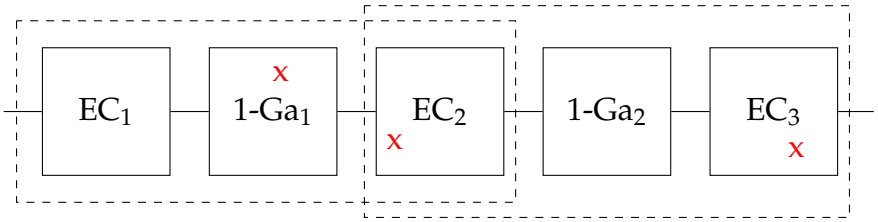


Figura 4.13: Dois 1-exRec's sobrepostos com 3 falhas.

Esta intuição corresponde, na verdade, à noção correta: ela funciona para níveis mais altos de concatenação. Dizemos que dois k-exRec's ruins são independentes se eles não se sobrepõem ou se o primeiro k-exRec é ruim se os k-ECs são removidos. Além dissso, um k-exRec é ruim se ele tem dois $(k-1)$-exRec's ruins independentes, e um k-exRec é bom se ele não é ruim. Veja [AGP05, Section 5.2] para uma demonstração que dois k-exRec's ruins não independentes são equivalentes a um único k-exRec ruim.

[7] Referimos o leitor a [AGP05, Seções 3 e 5].

Dizemos que um k-Rec é *correto* se ele preserva corretude. Isto é, ele leva estados corrigíveis a estados corrigíveis.[8] Para 1-Rec's, isto significa levar entradas com no máximo um erro a saídas com no máximo um erro. Para $k > 1$, podemos utilizar uma definição recursiva. Lembre-se que para um estado codificado em um código concatenado, um bloco de nível k é construído a partir de um bloco de nível $k-1$ codificando cada qubit físico nele. Alternativamente, podemos também visualizar um bloco de nível k como formado a partir de blocos de nível $k-1$. Dizemos que um bloco de nível k é corrigível se cada um dos blocos que o formam é corrigível. Um bloco de nível 1 é corrigível se ele tem no máximo um erro.

Com estas definições, é possível provar que um k-Rec contido em um k-exRec bom é correto. Como vimos, se todo k-exRec é bom, então a computação é confiável. Assim, a probabilidade de fracasso do circuito é obtida da probabilidade de que um k-exRec é ruim, que denotamos por $\varepsilon^{(k)}$. Esta probabilidade é obtida contando o número de pares de $(k-1)$-exRec's independentes. Como a probabilidade é maior para o maior k-exRec, a probabilidade de fracasso para o maior k-exRec fornece uma cota superior para a probabilidade de fracasso do circuito.

Agora, note que, devido à maneira como k-exRec's são construídos, o número de pares de $(k-1)$-exRec's independentes é o mesmo que o número de pares de posições no 1-exRec correspondente. Temos $\varepsilon^{(1)} \leq A\varepsilon^2$, onde A é o número de pares de posições no maior 1-exRec, e ε é a probabilidade de que uma falha ocorra. Para $k > 1$, temos $\varepsilon^{(k)} \leq A(\varepsilon^{(k-1)})^2$. Isto dá

$$\varepsilon^{(k)} \leq \frac{1}{A}(A\varepsilon)^{2^k}. \tag{4.46}$$

Logo, se $\varepsilon < 1/A$, a probabilidade de que um k-exRec é ruim se aproxima de 0 com $k \to \infty$. A probabilidade de fracasso p_{failure} pode ser limitada por cima por

$$p_{\text{failure}} \leq L\varepsilon^{(k)} \leq \frac{L}{A}(A\varepsilon)^{2^k}, \tag{4.47}$$

onde L é o número de posições no circuito.

Esta afirmação pode ser utilizada para provar que a computação é confiável. Lembre-se que um circuito é δ-confiável se para todo passo, o estado da computação é igual ao ideal, exceto com probabilidade δ. Em outras palavras, se executarmos uma medição ideal no estado do circuito ruidoso em um ponto arbitrário, a distribuição de probabilidade dos resultados da medição, que denotamos por P_{real}, é δ-próxima (em distância estatística) à distribuição das saídas de uma medição ideal executada no circuito ideal, denotada por P_{ideal}. Agora, devido ao fato de que P_{real} é igual a P_{ideal} exceto com probabilidade p_{failure}, temos $\delta \leq p_{\text{failure}}$. Por outro lado, este fato nos dá uma estimativa

[8] Adicionalmente, o k-Rec deve implementar a operação codificada.

no tamanho de k, dado δ: para que o circuito seja δ-confiável, é suficiente ter

$$2^k = \frac{\log(\varepsilon_0 L/\delta)}{\log(\varepsilon_0/\varepsilon)} \tag{4.48}$$

onde $\varepsilon_0 = A^{-1}$, e log denota o logaritmo em base 2. Finalmente, temos uma cota superior no tamanho do circuito. Se o maior 1-Rec tem l posições e profundidade d, então o número de posições L' e profundidade D do circuito satisfazem $L' \leq L\, l^k$ e $D' \leq D\, d^k$. Isto é, temos

$$L' \leq L \left(\frac{\log \frac{\varepsilon_0 L}{\delta}}{\log \frac{\varepsilon_0}{\varepsilon}} \right)^{\log l} \tag{4.49}$$

$$D' \leq D \left(\frac{\log \frac{\varepsilon_0 L}{\delta}}{\log \frac{\varepsilon_0}{\varepsilon}} \right)^{\log d} \tag{4.50}$$

Logo, computação confiável é alcançada com um *overhead* polilogarítmico, dado que $\varepsilon < \varepsilon_0$. O parâmetro ε_0 é conhecido como o *accuracy threshold*. Uma estimativa pode ser obtida contando o número de pares de posições no maior 1-exRec. Esta estimativa pode ser melhorada considerando a observação de que nem todos os pares de posições falhos podem causar fracasso. Em [AGP05], uma cota de $\varepsilon_0 \geq 2.73 \times 10^{-5}$ é obtida. Podemos resumir o resultado da seguinte maneira

Teorema 3. *Seja $\delta > 0$. Para um circuito quântico arbitrário $\mathcal{C}$ com L posições e profundidade D, existe $\varepsilon > 0$ e um circuito equivalente $\mathcal{C}'$, obtido via concatenação, tal que o circuito ruidoso $\tilde{\mathcal{C}}'$[9] é δ-confiável contra ruído independente com falhas que ocorrem com probabilidade ε. Adicionalmente, $\mathcal{C}'$ tem $L' = O(L \operatorname{polylog}(L/\varepsilon))$ posições, e profundidade $D' = O(D \operatorname{polylog}(L/\varepsilon))$. O parâmetro ε pode ser limitado por cima por $1/A$, onde A é o número de pares de posições no maior 1-exRec no primeiro nível de concatenação do circuito $\mathcal{C}'$.*

[9] Como definido na Seção 3.6.

Capítulo 5

Resistência a vazamentos concreta a partir de tolerância a falhas

Nosso objetivo é ter resistência a vazamentos *clássica*. No entanto, técnicas de tolerância a falha nos dão um circuito que, em geral, não possui uma tradução simples a um circuito clássico. Na Seção 5.1, mostraremos como transformar um conjunto particular de componentes quânticas em componentes clássicas. Isto é, para cada componente temos um circuito clássico que imita o circuito quântico, num sentido que será especificado de maneira mais precisa. Depois, na Seção 5.2, aplicaremos esta tradução aos *gadgets* apresentados na Seção 4.1. Utilizando o Teorema 2, que relaciona computação confiável a computação resistente a vazamentos, a construção resultante será resistente a vazamentos.

5.1 Tradução clássica de circuitos quânticos

Para mostrar a equivalência entre os circuitos quânticos e clássicos, analisamos cada ocomponente no seguinte cenário: supomos que a componente quântica tem entradas na base computacional—isto é, as entradas são clássicas—, e que após a execução da componente as saídas são medidas na base computacional. Com isso, mostramos que, para cada componente, existe um circuito clássico que, dadas as mesmas entradas, dá as mesmas saídas que a componente quântica, após a medição. Depois usamos estas componentes para construir *gadgets* codificados, que serão utilizados para implementar a porta Toffoli. Como as componentes têm uma tradução clássica, os *gadgets* e a porta Toffoli ganham uma tradução automaticamente.

Para que possamos combinar a tradução clássica com o Teorema 2, precisamos de uma forma de relacionar o vazamento no circuito quântico ao vazamento na tradução clássico. Em um certo sentido, queremos que o vazamento no circuito clássico não seja

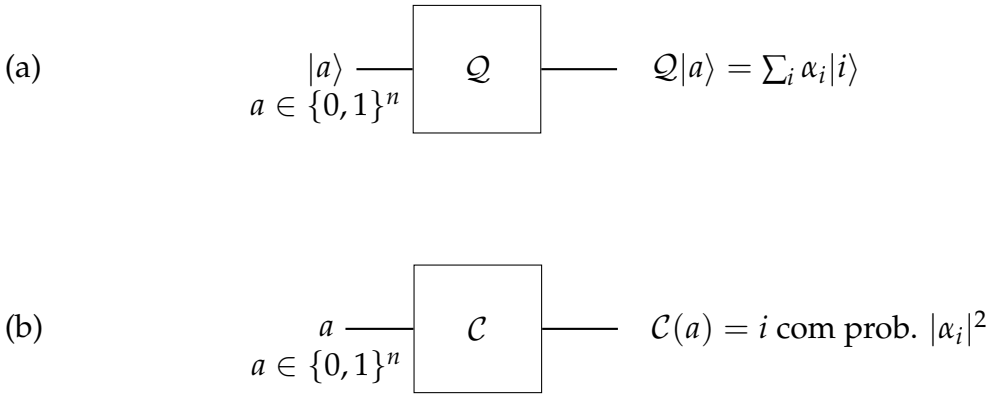


Figura 5.1: (a) Circuito quântico e (b) sua tradução clássica.

"mais forte" que no circuito quântico. Por exemplo, se tomarmos vazamento independente, não desejamos ter um circuito clássico com mais fios que o circuito quântico, devido ao fato de que o circuito clássico vazaria mais informação em média.

Definir um critério para modelos de vazamento gerais parece não ser trivial. Nossa solução ao problema é utilizar circuitos clássicos que a cada passo são equivalentes aos circuitos quânticos, pelo menos de um ponto de vista clássico. Isto é, se medirmos o circuito quântico na base computacional, devemos obter a mesma distribuição de probabilidade para as saídas que no circuito clássico. Em particular, isto significa que os circuitos quânticos e clássicos têm a mesma "forma", isto é, eles têm os mesmos fios. Eles podem diferir em quais portas são usadas, mas tais portas somente podem ser utilizadas para afetar a fase do estado quântico—estas ações são invisíveis do ponto de vista clássico.

Antes de prosseguir com o argumento, vamos primeiro descrever um truque que utilizamos na preparação do estado $|+\rangle$. O circuito é mostrado na Figura 5.2. A parte mostrda dentro do retângulo tracejado é suposto não ter ruído. Isto é, não há falhas em nenhum de seus fios. É fácil ver que o circuito é correto (isto é, ele dá como saída $|+\rangle$): antes da medição, o estado é $|+\rangle|0\rangle + |-\rangle|1\rangle$, de forma que a fase do primeiro qubit é invertida se o resultado da medição é -1.

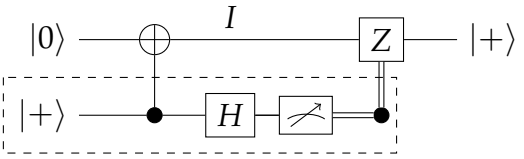


Figura 5.2: Preparação de $|+\rangle$. As componentes dentro do retângulo tracejado são supostas não ter ruído.

Por que utilizamos este procedimento? O motivo é que é impossível, através de

uma medição na base computacional—isto é, a partir do ponto de vista clássico—diferenciar entre $|+\rangle$ e $|-\rangle$. Este fato quebraria a correspondência entre vazamento e falhas. Utilizando o circuito da Figura 5.2, podemos garantir que $|+\rangle$ foi preparado. O uso da componente sem ruído é crucial aqui. Como veremos, o equivalente clássico consiste em preparar um bit aleatório livre de vazamentos, como introduzido na Seção 3.4. Provaremos o seguinte resultado.

Teorema 4. *Seja $\mathcal{C}$ um circuito quântico que aceita estados clássicos como entrada e contém apenas portas X, Z, CNOT, CZ e Toffoli, preparação de $|0\rangle$ e $|+\rangle$ e medição nas bases Z e X. Então, cada porta G no circuito pode ser substituída por um circuito clássico G' com os mesmos fios que G. Adicionalmente, supondo uma fonte de bits aleatórios livre de vazamentos, existem procedimentos para a preparação de $|0\rangle$ e $|+\rangle$ e medição nas bases Z e X tais que cada um deles pode ser substituído por um circuito clássico com os mesmos fios. Se todas estas substituições forem feitas, então o circuito clássico resultante $\mathcal{C}'$ dá as mesmas saídas que a aplicação de $\mathcal{C}$ seguida de uma medição das saídas na base computacional.*

Demonstração. Das portas utilizadas na construção, X, CNOT e Toffoli são clássicas. Consequentemente, suas traduções clássicas são triviais. A porta Z inverte a fase da entrada. Como supomos que entradas e saídas são clássicas, esta porta corresponde na tradução clássica à porta identidade. O mesmo vale para CZ.

Temos que preparar dois qubits físicos: $|0\rangle$. A tradução clássica da preparação de $|0\rangle$ corresponde a inicializar um registrador como 0. Para determinar a tradução de preparação de $|+\rangle$, note que a medição imediata na base computacional dá um bit aleatório. Assim, podemos traduzir a preparação de $|+\rangle$ ao cenário clássico preparando 0 e adicionando um bit aleatório r a ele, que, por hipótese, é gerado de maneira livre de vazamentos[1].

A saída não corresponde univocamente a $|+\rangle$; se tivéssemos preparado $|-\rangle$, teríamos a mesma saída, Mas podemos mostrar que do ponto de vista de um adversário clássico que pode visualizar todos os fios no circuito, exceto pela componente livre de vazamentos, este procedimento de preparação é equivalente ao procedimento que prepara $|+\rangle$, mostrado na Figura 5.2.

O segundo fio, mostrado dentro do retângulo tracejado, está na parte livre de vazamentos do circuito. O estado no ponto I é $|+\rangle\langle+| + |-\rangle\langle-|$. Este é o estado utilizado no verdadeiro circuito clássico. A figura mostra como poderíamos corrigir este estado utilizando a componente livre de vazamentos, de forma que, ao final, obtemos $|+\rangle$. Como a operação de correção não tem efeito na amplitude, ela pode ser omitida. Portanto, se utilizarmos o circuito da Figura 5.2 para a preparação de $|+\rangle$ no circuito $\mathcal{C}$, a

[1]Consequentemente, o valor não está disponível ao adversário.

tradução clássica consiste em inicializar um registrador a 0 e depois adicionar um bit aleatório livre de vazamentos ao registrador.

A medição na base X pode ser feita de maneira similar. Medir X projeta o estado $|\psi\rangle$ em um dos autoespaços do operador; os operadores de projeção são dados por

$$P_{\pm} = \frac{1}{2}(\mathbb{1} \pm X) \tag{5.1}$$

e o estado pós-medição ρ é dado por

$$\rho = P_+|\psi\rangle\langle\psi|P_+ + P_-|\psi\rangle\langle\psi|P_- \tag{5.2}$$

$$= \frac{1}{2}\sum_{i=0}^{1} X^i|\psi\rangle\langle\psi|X^i. \tag{5.3}$$

Assim, sob a hipótese de que a entrada $|\psi\rangle$ é um estado da base computacional, podemos executar a medição na base X adicionando $|+\rangle$ ao estado (via CNOT), o que é equivalente a inverter a amplitude da entrada aleatoriamente, e depois medi-la. Analogamente à preparação de $|+\rangle$, isto pode ser feito no circuito clássico adicionando um bit aleatório à entrada. □

O significado deste teorema é que podemos utilizar estas operações para implementar uma porta Toffoli confiável. A porta Toffoli é universal para computação clássica. As componentes quânticas utilizadas na implementação são cobertas por este teorema. Elas também são mostradas na Figura 5.3.

A implementação da porta Toffoli segue a construção desenvolvida na Seção 4.2.2, que funciona no $[[7,1,3]]$-código de Steane. Seguindo a discussão da Seção 4.2.5, esta implementação é confiável contra o modelo de ruído independente. Consequentemente, ela é confiável contra *ruído de fase independente*, onde supomos que apenas erros de fase são possíveis. Isto implica que ela é resistente contra vazamento independente (como definido na Seção 3.4), pelo Lema 2. Exploraremos esta conexão de maneira mais aprofundada na Seção 5.2, o que nos permitirá fazer algumas simplificações no circuito.

5.2 *Gadgets* resistentes a vazamentos

Nossa construção tolerante a falhas segue [AGP05], que funciona no modelo de ruído de fase independente, onde cada fio do circuito está sujeito a um erro de fase com probabilidade p. Lembramos da Seção 3.3 que um compilador resistente a vazamentos toma um circuito arbitrário e dá como saída um circuito com a mesma funcionalidade,

e a propriedade adicional de resistência a vazamentos. Para tornar o problema tratável, podemos reescrever o circuito em um circuito equivalente que utiliza apenas portas de um conjunto universal, isto é, um conjunto de portas que podem ser utilizadas para fazer computação arbitrária. Como observado na Seção 2.1.4, a porta Toffoli forma tal conjunto. Isto é, se pudermos implementer uma porta Toffoli resistente a vazamentos, podemos utilizá-la para obter circuitos resistentes a vazamento para computar qualquer função.

A construção da porta Toffoli apresentada na Seção 4.2.2 somente utiliza componentes especificadas no Teorema 4. Por este teorema, para cada componente existe uma tradução clássica com os mesmos fios que a componente quântica original, e portanto podemos aplicara relação entre ruído de fase independente e vazamento independente aplicando o Lema 2. Combinando o Teorema 3 com a implementação da porta Toffoli da Seção 4.2.2 e o Teorema 4, obtemos o seguinte resultado. (Lembramos que A é o número de pares de posições no maior 1-exRec.)

Corolário 1. *Seja L o modelo de vazamento independente e suponha que a probabilidade de vazamento p satisfaça $p < 1/A$. Para todo $\varepsilon > 0$, existe um ε-compilador resistente a vazamentos π tal que, dado como entradaum circuito clássico reversível arbitrário $\mathcal{C}$ com l posições e profundidade d, dá como saída um circuito clássico $\overline{\mathcal{C}}$ com $l' = O(l \operatorname{polylog}(l/\varepsilon))$ posições e profundidade $d' = O(d \operatorname{polylog}(l/\varepsilon))$.*

Demonstração. O Teorema 3 dá circuitos confiáveis contra ruído independente. Podemos relacionar esse modelo de ruído a vazamento pelo Lema 2. Aqui, apresentamos um argumento mais direto. Este argumento na verdade funciona para qualquer modelo de "vazamento em fios", isto é, um modelo onde o valor contido em fios vaza diretamente. Esta relação segue do fato de que o vazamento do estado de um fio é equivalente à introdução de um erro de fase, como mostrado na Figura 5.4.

Consequentemente, dado um circuito clássico reversível arbitrário $\mathcal{C}$, existe (pelo Teorema 3) um circuito quântico equivalente $\mathcal{C}'$ que é confiável contra ruído independente, e portanto (pela relação apresentada acima) também é resistente a vazamento independente, com l' posições e profundidade d'. Aplicando o Teorema 4 à implementação da porta Toffoli dada em 4.2.2, temos que $\mathcal{C}'$ pode ser tornado clássico. □

Dada a construção da porta Toffoli, temos um método para compilar um circuito arbitrário em um circuito em um circuito resistente a vazamentos que funciona da seguinte maneira. Primeiro, convertemos o circuito em um circuito que utiliza apenas portas Toffoli. Como vimos, nossa implementação tolerante a falhas da porta Toffoli (Seção 4.2.2) envolve apenas portas X, Z e CNOT, preparação de estados, medição nas bases Z e X, e correção de erros. Portanto, ela tem uma tradução clássica, devido ao

Teorema 4. Ademais, pelo Teorema 2, o circuito resultante é resistente a vazamentos do modelo de vazamento independente.

No restante dessa seção, apresentaremos a tradução clássica dos *gadgets* utilizados na construção da porta Toffoli. Eles podem ser obtidos mediante a aplicação do Teorema 4 aos *gadgets* introduzidos na Seção 4.2. Como veremos, no entanto, devido ao fato de que somente precisamos implementar computação clássica, além de que só temos que proteger contra erros de fase, podemos executar certas simplificações que reduzem o tamanho do circuito.

5.2.1 Medições

Para um código CSS geral, medir os operadores lógicos pode ser realizado por uma medição transversal, seguida por pós-processamento clássico. Para $\overline{Z}$, as medições são feitas na base computacional, que, como visto no Teorema 4, possuem uma tradução clássica trivial. Em particular, para o código de Steane temos $\overline{Z} = Z_1Z_2Z_3$, de forma que o autovalor pode ser obtido medindo os 3 primeiros qubits na base computacional.

Em geral, também é necessário realizar correção de erros para tornar o procedimento tolerante a falhas. No entanto, para o código de Steane, se temos apenas erros de fase, é possível $\overline{Z}$ de uma maneira bastante simples. Como explicado na Seção 4.1.4, o autovalor de $\overline{Z}$ somente pode ser afetado por erros de bit, que podem ser diagnosticados por uma medição transversal na base computacional seguida por verificações de paridade. Mas como estamos supondo que apenas erros de fase são possíveis, estas verificações não precisam ser feitas.

Como $\overline{X} = X_1X_2X_3$, podemos fazer uma medição na base X medindo os 3 primeiros qubits. No entanto, em nosso caso podemos também aplicar a operação $X_1X_2X_3$ controlada por $|+\rangle$, como mostrado na Figura 5.5, seguida por uma medição na base computacional.

Note que este procedimento não é transversal, mas erros de fase somente propagam do bloco de dados para o estado $|+\rangle$ auxiliar. Como argumentado na demonstração do Teorema 4, podemos supor que este estado foi preparado perfeitamente, uma vez que, na tradução clássica, ele corresponde à geração de um bit aleatório. Isto garante que o procedimento é tolerante a falhas.

5.2.2 Correção de erros

A correção de erros consiste em extração de síndrome, em que os erros são diagnosticados, e um passo de recuperação, executado para transformar o estado de volta ao estado correto. Para extração de síndrome, utilizamos o método de correção de erros

de Steane, introduzido na Seção 4.1.2. Pela discussão feita naquela seção, lembre-se que extração de síndrome para erros de bit e erros de fase são feitas separadamente. Uma vez que, por hipótese, apenas erros de fase são possíveis, podemos descartar a parte em que a síndrome de erros de bit é medida. Para erros de fase, precisamos preparar um estado auxiliar $|\overline{0}\rangle$, o que é feito utilizando o método de decodificação de estados auxiliares descrito na Seção 4.1.3. A tradução clássica do procedimento será discutida na Seção 5.2.3.

Para o passo de recuperação, note que como somente executamos computação em estados da base computacional, e supomos que apenas erros de fase são possíveis, o passo de recuperação consiste em aplicar operadores Z, os quais, pelo Teorema 4 (e Figura 5.3) nunca podem alterar o estado da computação, Portanto, o passo de recuperação pode ser removido com segurança da computação.

Isto não deveria ser surpreendente, uma vez que supomos que a computação clássica não tem erros. Note, no entanto, que não removemos o passo de extração de síndrome. Embora este passo não afete o estado lógico, ele pode ser interpretado como a injeção de aleatoriedade no sistema, o que intuitivamente ajuda a prevenir que o estado vaze completamente ao longo do tempo. Este método é comparável à técnica de mascaramento descrita na Seção 3.5.4.

5.2.3 Preparação de estados

Temos vários estados a preparar. precisamos do estado auxiliar de Toffoli $|\overline{\Theta}\rangle$, cujo procedimento de preparação requer os estados $|\overline{+}\rangle$ e o "estado de Shor" $|\text{even}_7\rangle$. Também precisamos preparar $|\overline{0}\rangle$ para o procedimento de correção de erros.

As preparações dos estados auxiliares são feitas de acordo com o método de decodificação de estados auxiliares descritos na Seção 4.2.4. Os circuitos de codificação e decodificação para $|\overline{0}\rangle$ são mostrados na Figura 4.9, enquanto os circuitos para $|\overline{+}\rangle$ são mostrados na Figura 4.10.

O método utilizado para preparar $|\overline{+}\rangle$ também pode ser usado para preparar o "estado de Shor" $|\text{even}_7\rangle$, dado pela superposição uniforme de todas as palavras de peso par em $\{0,1\}^7$, e que é usado na construção do estado auxiliar de Toffoli. Nosso método para preparar e verificar $|\text{even}_7\rangle$ é o mesmo apresentado em [DA07] para o *cat state*, com a diferença de que ele é feito na base complementar—$|\text{even}_7\rangle$ é obtido a partir do *cat state* pela aplicação da porta Hadamard transversalmente. O método é descrito em detalhe na Seção 4.2.4.

Notamos que, como observado na Seção 4.1.2, uma vez que a porta Toffoli não é uma porta Clifford, ela não propagada erros de Pauli a erros de Pauli. Em particular, um erro de fase no terceiro bloco é propagado a uma superposição de erros de fase.

Isto pode tornar uma simulação clássica do circuito de nível k (na construção concatenada) ineficiente, devido ao fato de que os padrões de erro crescem exponencialmente em k. Mas, novamente, isto não é um problema para nossa construção, porque não precisamos fazer recuperação de erros.

5.2.4 Portas lógicas

Como observado na Seção 5.1, para computacção clássica universal, somente é necessário implementar a porta Toffoli, descrita na Seção 4.2.2. A implementação apresentada ali utiliza outras portas lógicas: X, Z, CNOT e CZ.

Todas estas portas são transversais, como observado na Seção 4.2.1. Adicionalmente, pela aplicação do Teorema 4, podemos ver que a tradução clássica de $\overline{Z}$ consiste em aplicar a porta identidade transversalmente—uma vez que a implementação somente possui portas Z. Este é também o caso para $\overline{CZ}$.

Para a porta Toffoli, utilizamos a tradução clássica do *gadget* para a porta Toffoli, como mostrada na Figura 4.6. A corretude do circuito pode ser facilmente verificada. No entanto, não mostramos como executar o subcircuito no retângulo tracejado; de fato, este subcircuito utiliza uma porta Toffoli, que é exatamente a porta que estamos tentando implementar. Em vez disso, utilizamos um circuito alternativo para preparar o estado $|\overline{\Theta}\rangle = |\overline{000}\rangle + |\overline{100}\rangle + |\overline{010}\rangle + |\overline{111}\rangle$, que é a saída do subcircuito no retângulo tracejado. O circuito alternativo é mostrado na Figura 4.12.

Assim, temos uma maneira de preparar o estado auxiliar $|\overline{\Theta}\rangle$ de maneira tolerante a falhas. Ademais, a tradução clássica é fácil: a única componente nova aqui é a medição da paridade do estado de Shor, que pode ser feita medindo o estado na base computacional e realizando um XOR de todos os resultados. Agora, toda componente na Figura 4.6 tem uma tradução clássica.

5.3 Discussão

Em [AGP05], é obtida uma estimativa para o *accuracy threshold* ε_0; temos, $\varepsilon_0 \geq 2.73 \times 10^{-5}$. Como a nossa construção é derivada da construção no artigo citado, a mesma cota é válida aqui. Devido ao fato de que a nossa construção é mais simples, é de se esperar que uma estimativa melhor pode ser encontrada. A estimativa em [AGP05] é dominada pelo tamanho do maior 1-exRec para o CNOT, e o mesmo é válido para a nossa construção. No artigo citado, o 1-exRec para o CNOT tem 575 posições, enquanto o nosso tem 188; o número menor é devido ao fato de que não temos que executar a

correção de erros de bit. Uma estimativa rápida é a recíproca do número de pares de posições, que no nosso caso dá $\varepsilon_0 \approx 5.69 \times 10^{-5}$.

Isto é uma melhora modesta sobre a estimativa original. Podemos fazer um cálculo mais refinado, como descrito em [AGP05, Section 8.3], mas de qualquer forma isto não faria nossa construção melhor do que as já apresentadas na literatura, que funcionam para um nível de ruído arbitrário.

Podemos fazer uma comparação ao resultado de [DDF14]; ali, um protocolo para vazamento ruidoso (como definido na Seção 3.4) é apresentada[2] Eles obtêm um δ-compilador resistente a vazamento com *overhead* $O(d)$ onde $d := \log(1/\delta)$ que funciona para níveis de ruído $p > 1/2 - O(1/d)$. Nosso protocolo também funciona para vazamento ruidoso, aplicando o Lema 1. Como vimos (Teorema 3), o *overhead* $O(\text{polylog}(L/\delta))$ para um circuito com L posições, e funciona para níveis de ruído $p > 1/2 - \varepsilon_0/2$, onde ε_0 é o *accuracy threshold*. Assim, nossa construção funciona melhor para níveis altos de ruído e δ baixo; no entanto, é difícil fazer uma comparação direta porque os resultados em [DDF14] são assintóticos.

Deve ser possível obter uma construção melhorada, mas isto parece exigir uma abordagem diferente a tolerância a falhas. Em particular, o esquema de concatenação desenvolvido na Seção 4.2.5 não pode produzir um *overhead* constante (veja também [Got13]).

Uma opção promissora é utilizar *color codes* [BMD06, BMD07, LAR11]. *Gauge color codes* [Bom13] parecem ser adequados ao nosso cenário. *Color codes* são códigos CSS, então espera-se que uma tradução clássica seja possível; em particular, a maior parte da construção apresentada no Capítulo 4 funciona automaticamente—uma vez que ela é válida para códigos CSS. A exceção é a construção do estado auxiliar de Toffoli, que apenas funciona para códigos CSS baseados em códigos clássicos auto-duais (isto é, onde o código dual é igual ao código original); a construção de [Bom13] parece funcionar com esta restrição.

A principal atração de *color codes* é que eles parecem aceitar *thresholds* relativamente altos. Estimativas numéricas indicam que eles podem ir a até 10% [LAR11]. No entanto, ainda faltam estimativas rigorosas para um *threshold*, no espírito de [AGP05].

[2]Notamos que o tratamento no artigo citado inclui uma definição mais geral de vazamento ruidoso que a discutida aqui.

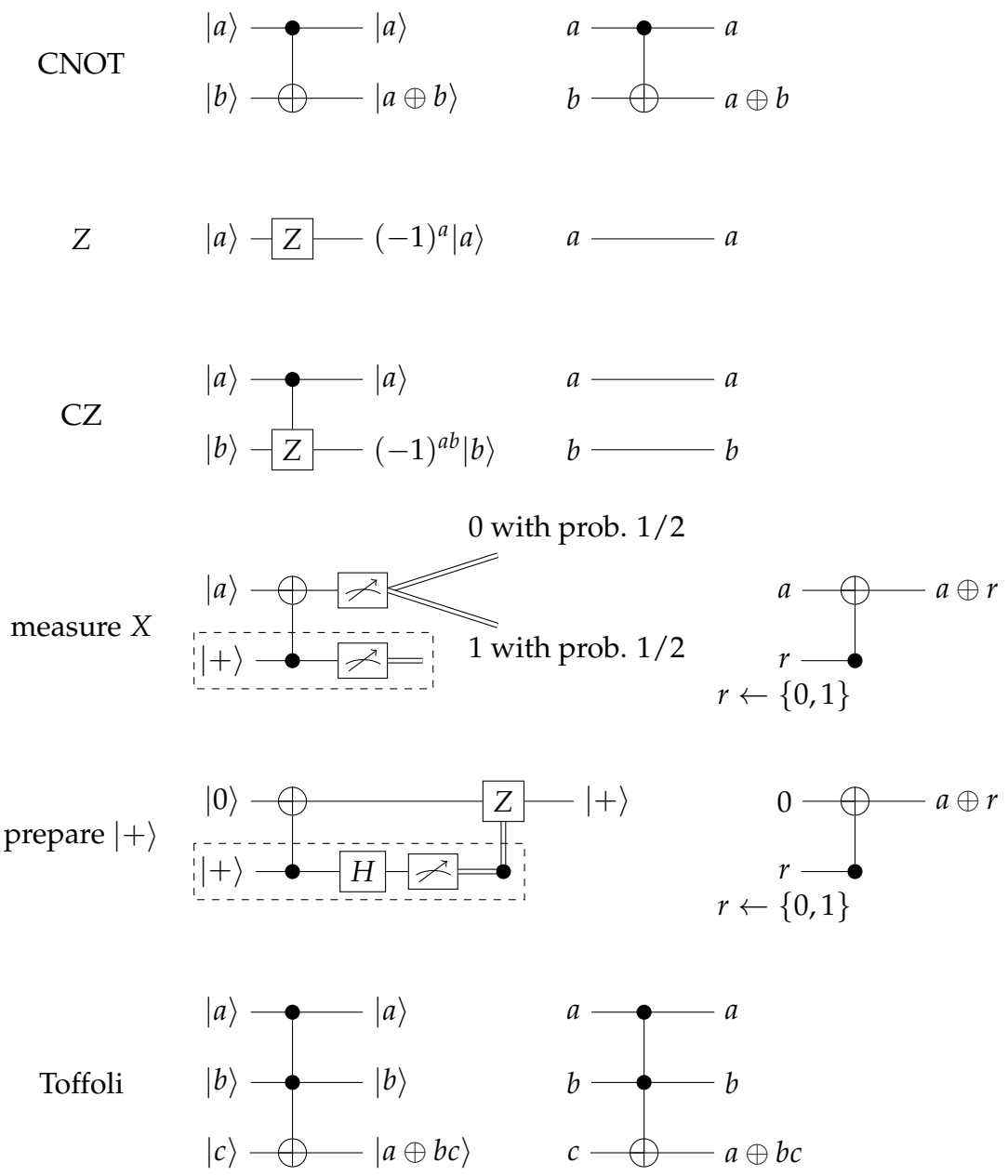


Figura 5.3: Componentes quânticas utilizadas na implementação resistente a vazamentos (esquerda) e sua tradução clássica (direita).

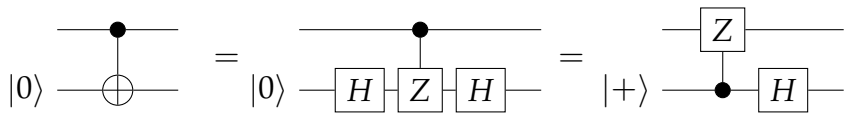


Figura 5.4: Vazar um bit (codificado no fio acima) é equivalente a introduzir um erro de fase.

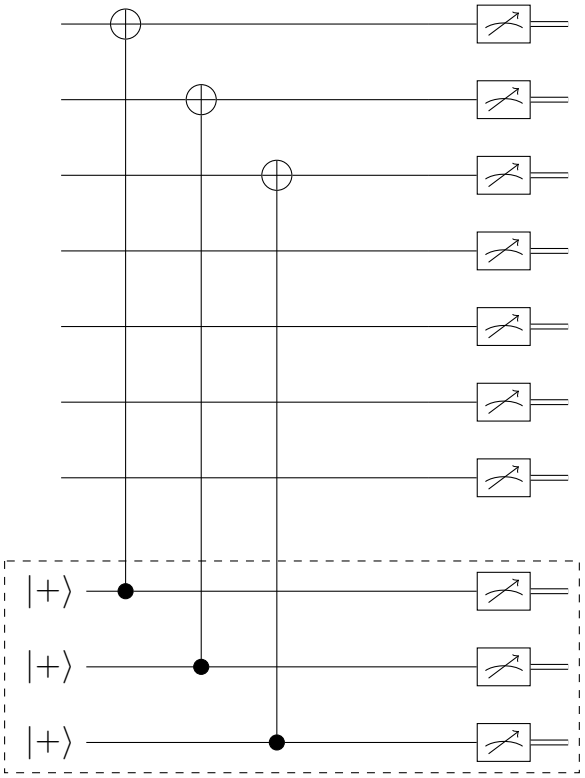


Figura 5.5: Medição na base X.

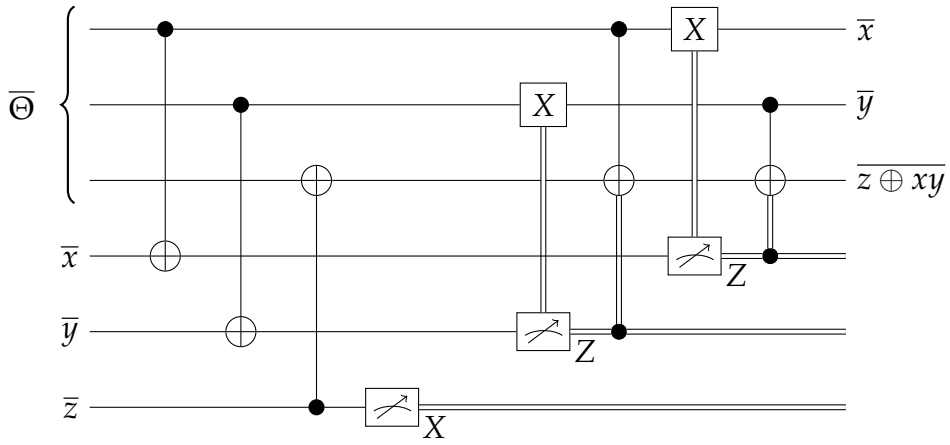


Figura 5.6: Porta Toffoli resistente a vazamentos, baseada na implementação da Figura 4.6. Todas as operações são executas em entradas codificadas. As portas de fase são substituídas pela identidade, e os *gadgets* de medição são realizados como explicado na Seção 5.2.1. A entrada $\overline{\Theta}$ é a saída da tradução clássica do circuito da Figura 4.12.

Capítulo 6

Conclusão

O Teorema 2 estabelece uma relação entre computação quântica tolerante a falhas e resistência a vazamentos. Combinando este resultado com o Teorema 4, podemos fazer uma conexão específica a resistência a vazamentos *clássica* (Corolário 1). Embora isto relacione modelos de vazamento e ruído (pelo Lema 2), não é claro, em geral, como as propriedades do modelo de ruído se relacionam às propriedades do modelo de vazamento. No entanto, em alguns casos o modelo de vazamento resultando de um dado modelo de ruído tem uma interpretação simples. Neste trabalho, analisamos o modelo de vazamento independente, que corresponde ao modelo de ruído de fase independente. Adicionalmente, pelo Lema 1, nossa construção também é resistente a vazamento ruidoso para nívels de ruído no intervalo $((1-p)/2, 1/2)$, onde p é dado pelo Corolário 1. Uma direção futura possível é tomar um modelo de vazamento que é utilizado em outras propostas de resistência a vazamento e tentar entender o modelo de ruído correspondente. Reciprocamente, poder-se-ia tomar, por exemplo, o modelo de "ruído local" (*local noise*) de [AGP05] e analisar como se parece o modelo de vazamento correspondente.

Desenvolvemos uma implementação concreta de computação universal resistente a vazamentos baseada na construção de tolerância a falhas de [DA07]. Esta construção funciona no modelo de ruído independente utilizando o código de Steane concatenado. Tolerância a falhas é alcançada supondo probabilidade de erro $p < \varepsilon_0$, onde ε_0 é o *accuracy threshold*. Juntando isto aos nossos resultados, temos uma construção resistente a vazamentos para computação clássica universai supondo vazamento independente com probabilidade de vazamento por fio $p < \varepsilon_0$, como expresso no Corolário 1. Pelo argumento dado em 5.3, podemos estimar $\varepsilon_0 \geq 5.69 \times 10^{-5}$.

Observando a tradução clássica de nosso esquema de codificação, nota-se que ele é um tipo de *mascaramento booleano* (*boolean masking*), uma técnica comumente usada para alcançar resistência a vazamentos [CJRR99, PR13, DDF14]. Esta conexão talvez

não seja tão surpreendente, se levarmos dois fatos em conta. Primeiro, códigos corretores de erro quânticos são relacionados a esquemas clássicos de *secret sharing* [Got00]. Segundo, esquemas de mascaramento booleano existentes são frequentemente baseados em *secret sharing*, que é uma ferramenta útil em provas de resistência a vazamentos. A principal diferença é que provamos a resistência a vazamentos de nosso esquema explorando resultados existentes em tolerância a falhas, em vez de usar a propriedade de *secret sharing* diretamente. Parece haver outros paralelos: operações que preservam a propriedade de *secret sharing* são feitas executando portas transversalmente, e a medição de síndromes é semelhante a uma atualização da codificação. Deixamos aberta a questão de fazer uma conexão formal entre as duas abordagens.

Apesar de que não esperamos que resultados existentes em tolerância a falha deem construções diretas para os modelos de vazamento comumente estudados na literatura—especialmente devido ao fato de que, até o momento, só se provou tolerância a falhas para poucos modelos de ruído—, notamos que tolerância a falhas quânticas é uma exigência mais forte que resistência a vazamento clássico: como vimos neste trabalho, traduzir de tolerância a falhas quânticas a resistência a vazamento clássico nos permite fazer várias simplificações. Adicionalmente, as técnicas utilizadas na área de tolerância a falhas são diferentes—embora relacionadas, como observado no parágrafo anterior. Assim, esperamos que nosso resultado clarifique, a partir de um novo ponto de vista, a área de resistência a vazamentos.

Uma outra direção possível é explorar o fato de que a conexão do Lema 2 funciona em ambas as direções: resultados na área de ataques de canal lateral e resistência a vazamentos podem nos dizer algo sobre tolerância a falhas. Trabalhos em tolerância a falhas frequentemente se preocupam com vazamento *adversarial*, no sentido de que o adversário tem uma certa escolha em que informação é vazada do circuito. Como esta é uma exigência mais forte que as dadas em modelos de vazamento probabilístico, é de se esperar que protocolos deveriam ser mais difíceis de obter. Não obstante, descobriu-se vários resultados positivos, desde o resultado inicial de [ISW03]. Em contraste, sabe-se muito pouco sobre os modelos correspondentes de "ruído adversarial" em tolerância a falhas. De fato, esta tarefa parece ser muito mais difícil, e já foram apresentados vários argumentos contra a possibilidade de tolerância a falhas em tal cenário [Kal09].

Referências

[AA04] Dmitri Asonov and Rakesh Agrawal, *Keyboard acoustic emanations*, 2012 IEEE Symposium on Security and Privacy, IEEE Computer Society, 2004, pp. 3–3. 42

[AGP05] Panos Aliferis, Daniel Gottesman, and John Preskill, *Quantum accuracy threshold for concatenated distance-3 codes*, arXiv preprint quant-ph/0504218 (2005). 5, 6, 49, 52, 58, 60, 73, 76, 78, 82, 86, 87, 90

[AK96] Ross Anderson and Markus Kuhn, *Tamper resistance—a cautionary note*, Proceedings of the second Usenix workshop on electronic commerce, vol. 2, 1996, pp. 1–11. 37, 43

[And08] Ross Anderson, *Security engineering*, John Wiley & Sons, 2008. 43

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson, *Lucky 13: Breaking the TLS and DTLS record protocols*, IEEE Symposium on Security and Privacy, 2013. 1

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael T. Kalai, and Guy N. Rothblum, *Program obfuscation with leaky hardware*, Advances in Cryptology–ASIACRYPT 2011, Springer, 2011, pp. 722–739. 3

[Bel98] Mihir Bellare, *Practice-oriented provable-security*, Information Security, Springer, 1998, pp. 221–231. 1

[Ben89] Charles H. Bennett, *Time/space trade-offs for reversible computation*, SIAM Journal on Computing **18** (1989), no. 4, 766–776. 14

[BFGV12] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede, *Theory and practice of a leakage resilient masking scheme*, Advances in Cryptology–ASIACRYPT 2012, Springer, 2012, pp. 758–775. 2

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang, *On the (im)possibility of obfuscating programs*, Advances in Cryptology–CRYPTO 2001, Springer, 2001, pp. 1–18. 3

[BMD06] Hector Bombin and Miguel Angel Martin-Delgado, *Topological quantum distillation*, Physical review letters **97** (2006), no. 18, 180501. 87

[BMD07] _____, *Exact topological quantum order in d=3 and beyond: Branyons and brane-net condensates*, Physical Review B **75** (2007), no. 7, 075103. 87

[Bom13] Hector Bombin, *Gauge color codes*, arXiv preprint arXiv:1311.0879 (2013). 87

[Can01] Ran Canetti, *Universally composable security: A new paradigm for cryptographic protocols*, Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on, IEEE, 2001, pp. 136–145. 29

[CGS02] Claude Crépeau, Daniel Gottesman, and Adam Smith, *Secure multi-party quantum computation*, Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM, 2002, pp. 643–652. 4

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi, *Towards sound approaches to counteract power-analysis attacks*, Advances in Cryptology—CRYPTO'99, Springer, 1999, pp. 398–412. 90

[CRSS97] Arthur R. Calderbank, Eric M. Rains, Peter W. Shor, and Neil J.A. Sloane, *Quantum error correction and orthogonal geometry*, Physical Review Letters **78** (1997), no. 3, 405. 10

[DA07] David P. DiVincenzo and Panos Aliferis, *Effective fault-tolerant quantum computation with slow measurements*, Physical Review Letters **98** (2007), no. 2, 020501. 57, 67, 70, 85, 90

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust, *Unifying leakage models: From probing attacks to noisy leakage.*, Advances in Cryptology–EUROCRYPT 2014, Springer, 2014, pp. 423–440. 6, 38, 39, 87, 90

[DF12] Stefan Dziembowski and Sebastian Faust, *Leakage-resilient circuits without computational assumptions*, Theory of Cryptography, Springer, 2012, pp. 230–247. 4

[DH76] Whitfield Diffie and Martin E. Hellman, *New directions in cryptography*, Information Theory, IEEE Transactions on **22** (1976), no. 6, 644–654. 1

[EK09] Bryan Eastin and Emanuel Knill, *Restrictions on transversal encoded quantum gate sets*, Physical review letters **102** (2009), no. 11, 110502. 51

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan, *Protecting circuits from leakage: the computationally-bounded and noisy cases*, Advances in Cryptology–EUROCRYPT 2010, Springer, 2010, pp. 135–156. 4, 37, 39, 40

[Gai08] Frank Gaitan, *Quantum error correction and fault-tolerant quantum computing*, CRC Press, 2008. 62

[Got97] Daniel Gottesman, *Stabilizer codes and quantum error correction*, arXiv preprint quant-ph/9705052 (1997). 21

[Got98a] ———, *The Heisenberg representation of quantum computers*, arXiv preprint quant-ph/9807006 (1998). 11, 24, 55

[Got98b] ———, *Theory of fault-tolerant quantum computation*, Physical Review A **57** (1998), no. 1, 127. 62

[Got00] ———, *Theory of quantum secret sharing*, Physical Review A **61** (2000), no. 4, 042311. 91

[Got09] ———, *An introduction to quantum error correction and fault-tolerant quantum computation*, Quantum Information Science and Its Contributions to Mathematics, Proceedings of Symposia in Applied Mathematics, vol. 68, 2009, p. 13. 50, 52

[Got13] ———, *What is the overhead required for fault-tolerant quantum computation?*, arXiv preprint arXiv:1310.2984 (2013). 54, 87

[GR10] Shafi Goldwasser and Guy N. Rothblum, *Securing computation against continuous leakage*, Advances in Cryptology–CRYPTO 2010, Springer, 2010, pp. 59–79. 4

[GR12] ———, *How to compute in the presence of leakage*, Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on, IEEE, 2012, pp. 31–40. 2, 3

[GST13] Daniel Genkin, Adi Shamir, and Eran Tromer, *RSA key extraction via low-bandwidth acoustic cryptanalysis*, Tech. report, Cryptology ePrint Archive, Report 2013/857, 2013. `http://eprint.iacr.org/2013/857`, 2013. 2, 42

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner, *Private circuits II: Keeping secrets in tamperable circuits*, Advances in Cryptology-EUROCRYPT 2006, Springer, 2006, pp. 308–327. 2, 3, 37

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner, *Private circuits: Securing hardware against probing attacks*, Advances in Cryptology-CRYPTO 2003, Springer, 2003, pp. 463–481. 2, 3, 4, 37, 38, 91

[KAI79] Takumi Kasai, Akeo Adachi, and Shigeki Iwata, *Classes of pebble games and complete problems*, SIAM Journal on Computing **8** (1979), no. 4, 574–586. 14

[Kal09] Gil Kalai, *Quantum computers: noise propagation and adversarial noise models*, arXiv preprint arXiv:0904.3265 (2009). 91

[Ker83] Auguste Kerckhoffs, *La cryptographie militaire*, 1883. 1

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun, *Differential power analysis*, Advances in Cryptology—CRYPTO'99, Springer, 1999, pp. 388–397. 41

[KK99] Oliver Kömmerling and Markus G Kuhn, *Design principles for tamper-resistant smartcard processors*, USENIX workshop on Smartcard Technology, vol. 12, 1999, pp. 9–20. 43

[KSW08] Dennis Kretschmann, Dirk Schlingemann, and Reinhard F. Werner, *The information-disturbance tradeoff and the continuity of Stinespring's representation*, IEEE Transactions on Information Theory **54** (2008), no. 4, 1708–1717. 46, 47

[LAR11] Andrew J. Landahl, Jonas T. Anderson, and Patrick R. Rice, *Fault-tolerant quantum computing with color codes*, arXiv preprint arXiv:1108.5738 (2011). 87

[LRR14] Felipe G. Lacerda, Joseph M. Renes, and Renato Renner, *Classical leakage resilience from fault-tolerant quantum computation*, arXiv preprint arXiv:1404.7516 (2014). 4

[Mau12] Ueli Maurer, *Constructive cryptography—A new paradigm for security definitions and proofs*, Theory of Security and Applications, Springer, 2012, pp. 33–56. 29

[MR04] Silvio Micali and Leonid Reyzin, *Physically observable cryptography*, Theory of Cryptography, Springer, 2004, pp. 278–296. 3

[MR11] Ueli Maurer and Renato Renner, *Abstract cryptography*, Innovations in Computer Science, 2011, pp. 1–21. 29, 33

[Nat07] National Security Agency, *TEMPEST: A signal problem*, `http://www.nsa.gov/public_info/_files/cryptologic_spectrum/tempest.pdf`, 2007, Online; accessed June 5th, 2013. 1

[NC00] Michael A. Nielsen and Isaac L. Chuang, *Quantum computation and quantum information*, Cambridge University Press, 2000. 15, 16, 17, 18, 23

[NRS01] Gabriele Nebe, Eric M. Rains, and Neil J.A. Sloane, *The invariants of the Clifford groups*, Designs, Codes and Cryptography **24** (2001), no. 1, 99–122. 11

[NS12] Moni Naor and Gil Segev, *Public-key cryptosystems resilient to key leakage*, SIAM Journal on Computing **41** (2012), no. 4, 772–814. 2

[Pei80] Charles S. Peirce, *A boolean algebra with one constant*, Collected papers of Charles Sanders Peirce **4** (1880), 1931–35. 12

[PM05] Thomas Popp and Stefan Mangard, *Masked dual-rail pre-charge logic: DPA-resistance without routing constraints*, Cryptographic Hardware and Embedded Systems–CHES 2005, Springer, 2005, pp. 172–186. 43

[PR13] Emmanuel Prouff and Matthieu Rivain, *Masking against side-channel attacks: A formal security proof*, Advances in Cryptology–EUROCRYPT 2013, Springer, 2013, pp. 142–159. 39, 90

[PR14] Christopher Portmann and Renato Renner, *Cryptographic security of quantum key distribution*, arXiv preprint arXiv:1409.3525 (2014). 29, 32

[Pre98] John Preskill, *Lecture notes for physics 229: Quantum information and computation*, California Institute of Technology (1998). 12, 14

[RS14] Joseph M. Renes and Volkher B. Scholz, *Operationally-motivated uncertainty relations for joint measurability and the error-disturbance tradeoff*, arXiv:1402.6711 [math-ph, physics:quant-ph] (2014). 46

[Sho95] Peter W. Shor, *Scheme for reducing decoherence in quantum computer memory*, Physical review A **52** (1995), no. 4, R2493. 19

[Sho96] _____, *Fault-tolerant quantum computation*, Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, IEEE, 1996, pp. 56–65. 52, 53

[SMY06] François-Xavier Standaert, Tal G. Malkin, and Moti Yung, *A formal practice-oriented model for the analysis of side-channel attacks*, IACR e-print archive **134** (2006), 2006. 2

[SP00] Peter W. Shor and John Preskill, *Simple proof of security of the BB84 quantum key distribution protocol*, Physical Review Letters **85** (2000), no. 2, 441. 3

[SPY+10] Francois-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald, *Leakage-resilient cryptography in practice*, Towards Hardware-Intrinsic Security, Springer, 2010, pp. 99–134. 2, 4

[SPY13] François-Xavier Standaert, Olivier Pereira, and Yu Yu, *Leakage-resilient symmetric cryptography under empirically verifiable assumptions*, Advances in Cryptology–CRYPTO 2013, Springer, 2013, pp. 335–352. 2, 4

[Ste96] Andrew Steane, *Multiple-particle interference and quantum error correction*, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **452** (1996), no. 1954, 2551–2577. 26

[Ste97] Andrew M. Steane, *Active stabilization, quantum computation, and quantum state synthesis*, Physical Review Letters **78** (1997), no. 11, 2252. 54

[Sti55] W. Forrest Stinespring, *Positive functions on c*-algebras*, Proceedings of the American Mathematical Society **6** (1955), no. 2, 211–216. 16

[SWT01] Dawn Xiaodong Song, David Wagner, and Xuqing Tian, *Timing analysis of keystrokes and timing attacks on SSH*, USENIX Security Symposium, vol. 2001, 2001. 42

[TAV02] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede, *A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards*, Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European, IEEE, 2002, pp. 403–406. 43

[Tof80] Tommaso Toffoli, *Reversible computing*, Springer, 1980. 11

[WG88] Peter Wright and Paul Greengrass, *Spycatcher: The candid autobiography of a senior intelligence officer*, Dell, 1988. 42

[Wil09] Mark M. Wilde, *Logical operators of quantum codes*, Physical Review A **79** (2009), no. 6, 062322. 23, 26

[ZZT09] Li Zhuang, Feng Zhou, and Doug Tygar, *Keyboard acoustic emanations revisited*, ACM Transactions on Information and System Security (TISSEC) **13** (2009), no. 1, 3. 42