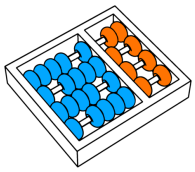
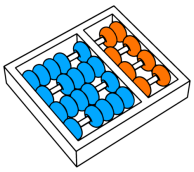


Izumi Oniki Chiquito

"Protocolos Criptográficos de Identificação Baseados em Reticulados"

CAMPINAS
2013



Universidade Estadual de Campinas
Instituto de Computação

Izumi Oniki Chiquito

"Protocolos Criptográficos de Identificação Baseados em Reticulados"

Orientador(a): **Prof. Dr. Ricardo Dahab**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

Este exemplar corresponde à versão final da Dissertação defendida por Izumi Oniki Chiquito, sob orientação de Prof. Dr. Ricardo Dahab.

Assinatura do Orientador(a)

CAMPINAS
2013

FICHA CATALOGRÁFICA ELABORADA POR
MARIA FABIANA BEZERRA MULLER - CRB8/6162
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Oniki Chiquito, Izumi, 1985-
On4p Protocolos criptográficos de identificação baseados em reticulados / Izumi Oniki Chiquito. – Campinas, SP : [s.n.], 2012.

Orientador: Ricardo Dahab.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

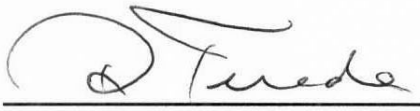
1. Criptografia. 2. Autenticação. 3. Criptografia pós-quântica. I. Dahab, Ricardo,1957-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

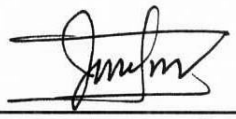
Título em inglês: Lattice-based identification schemes
Palavras-chave em inglês:
Cryptography
Authentication
Post-quantum cryptography
Área de concentração: Ciência da Computação
Titulação: Mestra em Ciência da Computação
Banca examinadora:
Ricardo Dahab [Orientador]
Routo Terada
Julio César López Hernández
Data de defesa: 26-09-2012
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 26 de Setembro de 2012, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Routo Terada
IME / USP



Prof. Dr. Julio César López Hernández
IC / UNICAMP

Prof. Dr. Ricardo Dahab
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Protocolos Criptográficos de Identificação Baseados em Reticulados

Izumi Oniki Chiquito[1]

Setembro de 2012

Banca Examinadora:

- Ricardo Dahab (Orientador)
- Routo Terada
  Instituto de Matemática e Estatística - USP
- Julio Cesar López Hernández
  Instituto de Computação - UNICAMP
- Jeroen van de Graaf (Suplente)
  Universidade Federal de Minas Gerais
- Marco Aurélio Amaral Henriques (Suplente)
  Faculdade de Engenharia Elétrica e de Computação - UNICAMP

[1]Suporte financeiro de: Bolsa de Mestrado CNPq (processo 135912/2009-1), vigência de 01/08/2009 a 31/07/2011.

Resumo

Na área de Segurança da Informação, controle de acesso diz respeito à habilidade de permitir ou negar a utilização de determinados recursos, sejam eles informações, dispositivos, serviços etc, por parte de um indivíduo. Protocolos de identificação correspondem a algoritmos criptográficos que permitem verificar, com certo grau de confiança, se a alegação de um indivíduo a respeito de sua identidade é verdadeira. Dessa forma, pode-se prover acesso controlado e conceder privilégios de utilização de recursos somente a entidades ou indivíduos cuja identidade tenha sido comprovada.

Algoritmos baseados em reticulados, de uma forma geral, têm despertado particular interesse em aplicações criptográficas, devido à sua provável resistência a ataques empregando computadores quânticos, ao contrário dos criptossistemas baseados em problemas da Teoria dos Números. Por esse motivo, nos últimos anos, tem-se buscado desenvolver protocolos de identificação cuja segurança esteja relacionada a problemas envolvendo reticulados.

Neste trabalho, foram abordadas as principais propostas recentes de protocolos de identificação baseados em reticulados. Além da apresentação dos algoritmos, é feita uma análise comparativa entre protocolos selecionados, incorporando dados experimentais de execução. A etapa de implementação aqui apresentada tem também como finalidade suprir a ausência de resultados experimentais para essa categoria de protocolos, no sentido de iniciar um processo de validação para uso dos algoritmos em aplicações práticas. Questões como possibilidades de otimização e expectativas para o futuro da área também são discutidas.

Abstract

One of the main concerns of the field of Information Security is access control, which refers to the restriction of access to several kinds of resources, such as data, places, devices, services and others. Identification schemes are cryptographic algorithms that allow verifying with some level of certainty if an identity claim is legitimate. Therefore, such schemes make possible to provide access control and grant privileges only to authorized individuals whose identities have been previously verified.

Lattice-based algorithms are particularly interesting as the cryptography community believes them to remain secure even to quantum computers attacks, as opposite to some cryptosystems used today based on Number Theory problems. For this reason, identification schemes based on lattices have received growing attention lately.

In this work, we address the main recent developments of lattice-based identification schemes. After introducing the algorithms, we make a comparative analysis of the selected schemes, using experimental data collected from our own implementation of the algorithms. The implementation phase also aims to help validating these schemes for practical use, since to this date there were practically no experimental results available. Other issues, like optimization possibilities and the future of the area, are also addressed in this work.

Agradecimentos

À minha família, sem a qual a realização deste trabalho não seria possível.

Ao meu orientador, Ricardo Dahab, que sempre demonstrou acreditar em meu potencial, e cujos comentários e observações auxiliaram não apenas a condução deste trabalho, mas em diversos aspectos da minha vida acadêmica e pessoal.

Ao Daniel, pela paciência, compreensão e apoio nos momentos mais importantes.

Ao colega Rosemberg André da Silva, por compartilhar sua experiência acerca do tema abordado, através de sugestões, dicas e esclarecimentos.

A Christiane Neme Campos, pelo apoio, especialmente durante minha participação no Programa de Estágio Docente e no período de Iniciação Científica que antecedeu este trabalho e que, de certa forma, me conduziu ao mestrado.

Aos membros da banca, pelos comentários e sugestões que permitiram o aprimoramento desta dissertação.

Ao CNPq, pelo suporte financeiro durante parte do projeto.

E a todos os demais que de alguma forma contribuíram para a concretização deste trabalho.

Sumário

Capítulo 1

Introdução

Atualmente, diversas de nossas atividades corriqueiras, especialmente as que envolvem sistemas computacionais, exigem que as ações realizadas sejam executadas apenas por determinados indivíduos previamente autorizados. Exemplos incluem o fato de somente um usuário devidamente cadastrado poder acessar um determinado computador pessoal, apenas moradores de uma certa residência poderem utilizar uma rede sem fio específica, a contratação de serviços para uma determinada linha telefônica ser restrita à solicitação do próprio assinante da linha, o acesso restrito ao conteúdo de páginas na *web*, realização de transações bancárias pela *internet*, entre outros.

Para impedir o acesso indevido a informações ou a realização de operações por pessoas não autorizadas, frequentemente é empregada uma etapa de verificação de identidade. Nestes casos, a partir da identificação provida, procura-se comprovar se a mesma é válida, isto é, se o indivíduo realmente é quem alega ser. Este procedimento é usualmente chamado de *identificação* ou *autenticação*.

As diversas maneiras a partir das quais pode ser realizada a identificação empregam o que se chama de *fatores de autenticação*, que correspondem aos elementos utilizados para verificar a identidade de um determinado indivíduo. Estes fatores de autenticação subdividem-se em três categorias:

- **Fatores de inerência**: são os fatores que correspondem àquilo que o indivíduo é ou faz. Geralmente compreendem identificadores biométricos, isto é, características físicas intrínsecas ou comportamentos particulares do indivíduo. Alguns exemplos incluem a autenticação por impressão digital, voz, padrão de retina e reconhecimento

de faces.

- **Fatores de posse:** são os fatores que correspondem àquilo que o indivíduo possui. Como exemplos, podemos citar documentos como a cédula de identidade e passaporte, *tokens* de segurança oferecidos por bancos a seus clientes, pulseiras de identificação, entre outros.

- **Fatores de conhecimento:** são os fatores que correspondem àquilo que o indivíduo sabe ou conhece. Exemplos incluem senhas, dados pessoais sigilosos e frases de segurança.

Para aumentar a segurança, isto é, a dificuldade de um indivíduo não autêntico conseguir burlar o processo de identificação, frequentemente se emprega uma combinação de dois ou mais métodos ou fatores. Por exemplo, para realizar uma compra em um estabelecimento com débito em conta bancária, pode ser necessário apresentar um cartão do banco (fator de posse), e também fornecer uma senha (fator de conhecimento). Nestes casos, diz-se que está sendo feita uma *autenticação de dois fatores*.

Uma das principais dificuldades no processo de identificação consiste em garantir que os elementos de autenticação não possam ser facilmente reproduzidos, como no caso dos fatores de inerência e posse, ou deduzidos, no caso de fatores de conhecimento. Os métodos abordados neste trabalho correspondem a diversos algoritmos criptográficos para identificação em sistemas computacionais que buscam justamente contornar essas dificuldades.

1.1 Motivação

A Criptografia pode ser definida como o estudo de técnicas matemáticas relacionadas a aspectos de segurança de informações, como confidencialidade, integridade de dados e autenticação [46]. Para atingir seus objetivos, muitos dos algoritmos criptográficos atualmente empregados toma como base problemas relacionados à Teoria dos Números, tais como o de fatoração de números inteiros, como o algoritmo RSA [57], e o de cálculo de logaritmos discretos, como a criptografia baseada em curvas elípticas [29]. Nestes casos, a segurança dos métodos fundamenta-se na dificuldade de resolver tais problemas em tempo hábil, utilizando os recursos computacionais existentes.

Com o advento da computação quântica, no entanto, será possível solucionar diversos problemas matemáticos muito rapidamente através do uso de computadores quânticos. Em particular, a fatoração de números inteiros e o cálculo de logaritmos discretos poderão ser resolvidos em tempo polinomial pelo algoritmo de Shor [61], tornando vulneráveis todos os sistemas que neles se embasam.

Acredita-se, porém, que diversas classes de sistemas criptográficos resistam tanto a computadores clássicos, quanto a computadores quânticos. Exemplos incluem sistemas baseados em códigos, como o McEliece [45], sistemas baseados em *hash*, como a assinatura de chave pública de Merkle [47], e sistemas baseados em reticulados, como o NTRU [31]. O estudo de soluções que possam ser aplicadas mesmo após a construção de computadores quânticos compreende a área que se convencionou chamar de *criptografia pós-quântica*.

Em termos históricos, os reticulados, em particular, foram estudados desde o século 18, por matemáticos como Lagrange [40] e Gauss [20]. Mais recentemente, vêm sendo empregados em diversas aplicações, não apenas criptográficas ou puramente matemáticas, mas também da Física, na forma de modelos e estruturas das ciências dos materiais e da física do estado sólido.

No que diz respeito às construções criptográficas, embora certos algoritmos baseados em reticulados apresentem provas de segurança, estes nem sempre são suficientemente eficientes para serem utilizados em aplicações reais. Outros, ainda que apresentem bom desempenho na prática, carecem do apoio de uma demonstração formal de segurança. Dessa forma, tanto o desenvolvimento de algoritmos seguros quanto de implementações eficientes correspondem a interessantes tópicos de pesquisa.

1.2 Objetivos

Neste trabalho, procurou-se contribuir no esforço de ampliação dos horizontes da criptografia pós-quântica, ao explorar esquemas de identificação baseados em reticulados. Em especial, buscou-se analisar as últimas propostas de protocolos desta categoria e, a partir deste conjunto, selecionou-se para implementação os três candidatos identificados como mais promissores, no sentido de proverem resultados experimentais mais interessantes ou proveitosos.

Atualmente, resultados sobre eventuais execuções de tais protocolos se resumem às

previsões teóricas apresentadas pelos respectivos autores. Desta forma, um dos objetivos foi coletar dados reais de execução, procurando averiguar até que ponto essas previsões se confirmam na prática. Verificações como essa são importantes no sentido de validarem as previsões teóricas, aumentando a confiança no algoritmo, ou encontrarem pequenas discrepâncias, sugerindo a existência de pontos ou suposições que devem ser revisados.

Outro objetivo foi identificar estratégias que possam prover melhorias consideráveis de desempenho dos protocolos. De forma geral, pode-se dizer que este trabalho se concentrou naquelas relativamente independentes de plataforma[1], de forma a poderem futuramente ser aplicadas de modo mais abrangente. Particularmente, analisou-se o impacto do uso de reticulados ideais na eficiência dos protocolos, bem como de técnicas de paralelização de código.

Ao final, procurou-se fomentar as discussões acerca do estado atual do desenvolvimento de protocolos de identificação, além de contribuir na tentativa de delinear possibilidades futuras e direções que oferecem boas perspectivas de resultados.

1.3 Convenções e Nomenclatura

Ao longo deste texto, procurou-se utilizar a maioria dos termos e expressões em português. Infelizmente, para certos vocábulos originários de idiomas estrangeiros, ainda não foi consolidada uma tradução para o português. Dessa forma, sempre que possível, adotou-se a tradução que se considerou mais adequada, seja por ser a mais disseminada nos trabalhos relacionados analisados, ou porque se julgou representar melhor o conceito a que se refere.

No geral, tais termos são acompanhados de sua definição, o que se supõe ser suficiente para acompanhar o texto. Ainda assim, ao final deste documento, foi incluído um glossário onde são apresentados em conjunto a tradução para o português adotada e a forma em inglês mais conhecida.

Uma minoria de termos, aqueles cuja forma em inglês já se encontra amplamente difundida e não há bons substitutos em português (por exemplo, *thread* de execução), foram mantidos no idioma original, porém grafados em itálico.

---

[1]Por estratégias de otimização dependente de plataforma, entende-se aquelas específicas para um determinado processador, sistema operacional, etc.

Ao final do documento, encontra-se também uma lista de notações com os principais símbolos empregados em alguns trechos do texto. Assim como no caso das traduções, em geral, a primeira ocorrência de cada símbolo é acompanhada da descrição de seu significado, de forma que a tabela de notações foi incluída para ser utilizada com fins de consulta.

1.4 Organização do Texto

Esta dissertação encontra-se estruturada em quatro capítulos principais, além desta introdução. O Capítulo 2 introduz os conceitos fundamentais de reticulados, incluindo suas definições básicas e resultados elementares. Além disso, são enunciados os principais problemas empregados nas construções criptográficas atuais, incluindo os protocolos abordados nesta dissertação. Ao final do capítulo, comenta-se sobre a classe de reticulados ideais, suas características e seu uso em algoritmos criptográficos.

O Capítulo 3 trata do objeto de estudo deste trabalho, isto é, protocolos de identificação. Expõe-se uma visão mais formal do processo de autenticação até agora mencionado, incluindo propriedades que devem ser respeitadas e os modelos de segurança utilizados para classificá-los. Brevemente, é traçado um paralelo entre protocolos de identificação e esquemas de assinatura digital, apresentando-se maneiras de converter tais protocolos entre si. Grande parte do capítulo é dedicada a apresentar os protocolos de identificação estudados, isto é, aqueles cuja segurança se baseia em problemas envolvendo reticulados.

No Capítulo 4, é apresentado o experimento prático realizado neste trabalho, a partir da implementação de alguns protocolos selecionados do conjunto listado no Capítulo 3. Para cada protocolo, foram elaboradas mais de uma versão de implementação, de forma que o quarto capítulo lista as diferenças entre elas, juntamente com as plataformas utilizadas para desenvolvimento e execução. Além disso, são apresentados os dados experimentais coletados a partir da execução dos protocolos.

O Capítulo 5 aprofunda a discussão acerca das principais diferenças e semelhanças entre os protocolos implementados. Dessa forma, buscou-se também averiguar as vantagens do uso de cada um desses algoritmos, seja a partir dos dados experimentais, ou das características teóricas apresentadas no Capítulo 3. Além disso, neste capítulo, são examinadas algumas possibilidades de otimização dos protocolos. Não apenas são apontadas técnicas

para melhoria de desempenho, mas também discute-se sobre os efeitos dessas modificações na tentativa de comparação dos algoritmos. São apresentadas, ainda, outras formas de se estender o trabalho apresentado neste documento, que não apenas através de otimizações.

A discussão é encerrada no Capítulo 6, com perspectivas acerca do futuro da area. É apresentada uma visão final sobre os trabalhos já desenvolvidos e aqui analisados, bem como o que acreditamos ser o possível rumo natural de evolução e desenvolvimento de tópicos relacionados. Além disso, expressamos nossas expectativas no que diz respeito à obtenção de resultados distintos aos vistos até o momento.

Capítulo 2

Reticulados

Conforme mencionado anteriormente, o estudo e desenvolvimento de sistemas criptográficos baseados em reticulados constitui um dos principais focos da criptografia pós-quântica. Neste capítulo, serão introduzidos os conceitos e definições fundamentais que dizem respeito a reticulados, bem como alguns dos problemas computacionalmente difíceis neles baseados. Particularmente, são introduzidos os reticulados ideais e comentado como estes podem ser aproveitados para desenvolver implementações eficientes da classe de protocolos considerada neste trabalho.

2.1 Conceitos Básicos

2.1.1 Definições

Considere um conjunto de vetores $\mathbf{v_1}, \mathbf{v_2}, ..., \mathbf{v_n} \in \mathbb{R}^m$ linearmente independentes. O *reticulado* L gerado por esses vetores é definido como o conjunto de combinações lineares de $\mathbf{v_1}, \mathbf{v_2}, ...\mathbf{v_n}$ com coeficientes em $\mathbb{Z}$:

$$L = \{a_1\mathbf{v_1} + a_2\mathbf{v_2} + ... + a_n\mathbf{v_n} : a_1, a_2, ..., a_n \in \mathbb{Z}\} \tag{2.1}$$

Em geral, as definições envolvendo reticulados são bastante semelhantes àquelas que concernem espaços vetoriais. Por exemplo, assim como uma base de um espaço vetorial V corresponde a qualquer conjunto de vetores linearmente independentes que geram V, um

conjunto de vetores linearmente independentes que geram o reticulado L é dito uma *base* de L. Duas bases de L têm o mesmo número de elementos, o qual chamamos de *dimensão* de L. Além disso, se $n = m$, então dizemos que o reticulado é de *dimensão completa*.

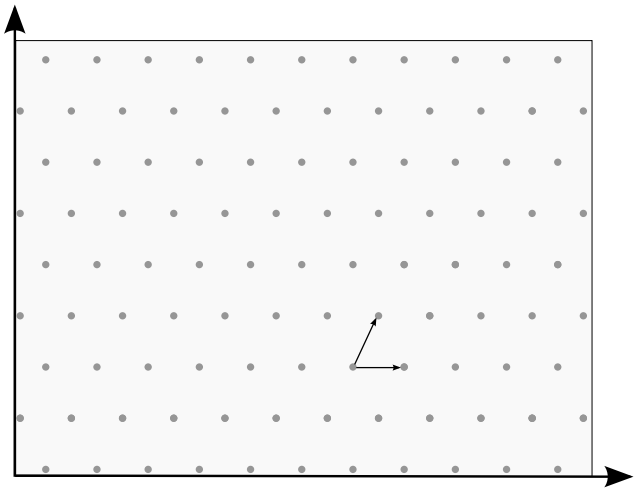


Figura 2.1: Um reticulado de dimensão 2 (pontos cinza no plano) e uma de suas bases (vetores em preto).

2.1.2 Formas de Representação de Reticulados

A equação 2.1 corresponde à forma mais usual de se introduzir o conceito de reticulados, a partir de vetores linearmente independentes que compõem uma de suas bases. Pode, inclusive, ser representada de um modo mais compacto: se $B = \{\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_n\}$ é o conjunto de vetores que compõem a base, podemos denotar o reticulado gerado por B simplesmente por $L(B)$.

Além disso, existem outras formas de definir um reticulado unicamente, que não a da equação 2.1, conforme será descrito a seguir.

Representação a partir de base em forma matricial

Seja $B = \{\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_n\}$ um conjunto de vetores linearmente independentes, com $\mathbf{v}_i = [\ b_{1i} \quad b_{2i} \quad \dots \quad b_{mi}\ ]^T$, $b_{ji} \in \mathbb{R}$, $i = 1, 2, ..., n$ e $j = 1, 2, ..., m$. Frequentemente, a base é representada por uma matriz $\mathbf{B} \in \mathbb{R}^{m \times n}$ onde as colunas correspondem aos vetores de B:

$$\mathbf{B} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

Neste caso, L pode ser definido como:

$$L(\mathbf{B}) = \{\mathbf{v} \in \mathbb{R}^m : \mathbf{v} = \mathbf{B}\mathbf{u}, \mathbf{u} \in \mathbb{Z}^n\} \tag{2.2}$$

Naturalmente, $L(B)$ e $L(\mathbf{B})$ correspondem ao mesmo reticulado, pois a definição é realizada a partir da mesma base. Porém, em muitos casos é mais conveniente representar a base em sua forma matricial, seja pela notação mais compacta, ou pela semelhança com possíveis estruturas computacionais que poderão ser utilizadas em implementações práticas, ou ainda para facilitar a aplicação de certas operações sobre os elementos do reticulado.

Além disso, duas bases $\mathbf{B}$ e $\mathbf{B}'$ de um mesmo reticulado (isto é, tais que $L(\mathbf{B}) = L(\mathbf{B}')$) estão sempre relacionadas por uma *matriz unimodular*, ou seja, uma matriz quadrada de coeficientes inteiros com determinante $+1$ ou -1.

Lema 1. *Sejam* $\mathbf{B}$ *e* $\mathbf{B}'$ *duas bases do mesmo reticulado de dimensão* n*. Então existe uma matriz* $\mathbf{M}$ *de inteiros tal que* $\mathbf{B} = \mathbf{B}'\mathbf{M}$*.*

Demonstração. Sejam $\mathbf{v}_j$ os vetores que compõem as colunas da matriz $\mathbf{B}$ e $\mathbf{u}_i$ as colunas da base $\mathbf{B}'$. Então, da definição de reticulado, todo $\mathbf{v}_j$ pode ser escrito como uma combinação linear dos $\mathbf{u}_i$ com coeficientes inteiros: $\mathbf{v}_j = \sum_{i=1}^{n} a_{ij}\mathbf{u}_i$, $a_{ij} \in \mathbb{Z}$. Dessa forma, para termos $\mathbf{B} = \mathbf{B}'\mathbf{M}$, com $\mathbf{M}$ sendo uma matriz de inteiros, basta tomar os a_{ij} como elementos de $\mathbf{M}$. □

Teorema 1. *Sejam* $\mathbf{B}$ *e* $\mathbf{B}'$ *duas bases de reticulados. Então* $L(\mathbf{B}) = L(\mathbf{B}')$ *se e somente se existe uma matriz unimodular* $\mathbf{U}$ *tal que* $\mathbf{B}' = \mathbf{B}\mathbf{U}$*.*

Demonstração. Sejam $\mathbf{B}$ e $\mathbf{B}'$ duas bases de reticulados e $\mathbf{U}$ uma matriz unimodular tal que $\mathbf{B}' = \mathbf{B}\mathbf{U}$. Tomemos um elemento $\mathbf{v} \in L(\mathbf{B}')$. Então, a partir da definição

apresentada na equação 2.2, podemos escrever $\mathbf{v} = \mathbf{B}'\mathbf{w} = (\mathbf{BU})\mathbf{w} = \mathbf{B}(\mathbf{Uw}) = \mathbf{Bw}'$, onde $\mathbf{w}' = \mathbf{Uw}$. Como tanto $\mathbf{U}$ quanto $\mathbf{w}$ possuem coeficientes inteiros, então $\mathbf{w}' \in \mathbb{Z}^n$. Dessa forma, como $\mathbf{v} = \mathbf{Bw}'$, então $\mathbf{v} \in L(\mathbf{B})$. Portanto, $L(\mathbf{B}') \subseteq L(\mathbf{B})$. Analogamente, se existir uma matriz unimodular $\mathbf{U}'$, tal que $\mathbf{B} = \mathbf{B}'\mathbf{U}'$, teremos que $L(\mathbf{B}) \subseteq L(\mathbf{B}')$. No caso, temos que $\mathbf{BUU}^{-1} = \mathbf{B}'\mathbf{U}^{-1} \Rightarrow \mathbf{B} = \mathbf{B}'\mathbf{U}^{-1}$. Além disso, como $\mathbf{U}^{-1}$ também é unimodular, temos que $L(\mathbf{B}) = L(\mathbf{B}')$.

Considere agora $\mathbf{B}$ e $\mathbf{B}'$ como sendo bases do mesmo reticulado. Então, pelo Lema 1, existem duas matrizes $\mathbf{M}$ e $\mathbf{M}'$ de inteiros, tais que $\mathbf{B} = \mathbf{B}'\mathbf{M}$ e $\mathbf{B}' = \mathbf{BM}'$. Dessa forma, $\mathbf{B} = \mathbf{B}'\mathbf{M} = \mathbf{BM}'\mathbf{M} \Rightarrow \mathbf{B} - \mathbf{BM}'\mathbf{M} = \mathbf{0} \Rightarrow \mathbf{B}(\mathbf{I} - \mathbf{M}'\mathbf{M}) = \mathbf{0}$. Como os vetores de $\mathbf{B}$ são linearmente independentes, precisamos ter $\mathbf{I} - \mathbf{M}'\mathbf{M} = \mathbf{0}$, ou seja, $\mathbf{M}'\mathbf{M} = \mathbf{I}$. Assim, $det(\mathbf{M}')\,.\,det(\mathbf{M}) = det(\mathbf{I}) = 1$. Como as matrizes $\mathbf{M}$ e $\mathbf{M}'$ possuem coeficientes inteiros, $det(\mathbf{M}), det(\mathbf{M}') \in \mathbb{Z}$ e, portanto, $det(\mathbf{M}) = det(\mathbf{M}') = \pm 1$, de forma que ambas as matrizes são unimodulares. □

Representação a partir de vetores ortogonais

Outra forma de construir reticulados a partir de matrizes é, dada uma matriz $\mathbf{A}^{n\times m}$ e um número primo q, considerar o reticulado $L_q^{\perp}(\mathbf{A})$ definido como:

$$L_q^{\perp}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Ax} = 0 \bmod q\} \tag{2.3}$$

Neste caso, podemos também ver $\mathbf{A}$ como um conjunto de n vetores:

$$\mathbf{A} = \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$

Entretanto, desta vez os vetores $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$ correspondem às linhas de $\mathbf{A}$, e um elemento $\mathbf{x} = [\ x_1 \quad x_2 \quad \ldots \quad x_m\ ]^T$ do reticulado é sempre ortogonal módulo q a esses vetores:

$$\mathbf{Ax} = 0 \bmod q \Rightarrow \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = 0 \bmod q \Rightarrow$$

$$\Rightarrow \begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m = 0 \bmod q \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m = 0 \bmod q \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nm}x_m = 0 \bmod q \end{cases} \Rightarrow \begin{cases} \mathbf{v}_1\mathbf{x} = 0 \bmod q \\ \mathbf{v}_2\mathbf{x} = 0 \bmod q \\ \vdots \\ \mathbf{v}_n\mathbf{x} = 0 \bmod q \end{cases}$$

A forma de determinar um reticulado dada pela equação 2.3 é utilizada na definição do Problema da Solução Inteira Curta (SIS), o qual constitui a base de alguns dos protocolos de identificação abordados neste trabalho. Mais detalhes sobre este problema, incluindo a dificuldade de resolvê-lo, serão apresentados na Seção 2.2.

Representação por matriz de Gram

Para uma base $\mathbf{B}$ de um reticulado L, sua matriz de Gram é definida por $\mathbf{G} = \mathbf{B}\mathbf{B}^T$. Ou seja, $\mathbf{G}$ é a matriz formada por todos os possíveis produtos escalares de dois vetores da base $\mathbf{B}$. Uma matriz de Gram também pode ser utilizada para definir o reticulado unicamente, o qual é denotado neste caso por $L_{\mathbf{G}}$. Dessa forma, um reticulado pode admitir diversas representações por matriz de Gram, em função de suas inúmeras possíveis bases.

Ainda não se conhece uma forma eficiente de obter uma base $\mathbf{B}$ que satisfaça a equação $\mathbf{G} = \mathbf{B}\mathbf{B}^T$, onde $\mathbf{G}$ é uma matriz de Gram do reticulado. Contudo, atualmente também não há nenhuma demonstração formal de que este seja um problema computacionalmente difícil, embora exista esta conjectura, conforme veremos na próxima seção.

Representação pela forma normal de Hermite

Podemos ainda representar um reticulado a partir de sua forma normal de Hermite, a qual é dada por uma matriz $\mathbf{H}$ onde as linhas são formadas por vetores que compõe uma base do reticulado e que obedecem aos seguintes critérios:

- a matriz resultante é triangular inferior[1],
- os elementos da diagonal principal são positivos,
- em uma linha, todos os elementos à esquerda da diagonal principal são negativos e com módulo superior ao valor da diagonal naquela linha.

Ao contrário da representação por matriz de Gram ou por apresentação de uma base, a forma normal de Hermite é única para um dado reticulado [13]. Além disso, pode ser obtida a partir de uma base $\mathbf{B}$ do reticulado através de uma série de operações elementares sobre suas colunas, de forma a termos $\mathbf{H} = \mathbf{BU}$, onde $\mathbf{U}$ é uma matriz unimodular [9, 30].

Por ser uma representação única, dois reticulados são ditos *isomorfos* se e somente se possuem a mesma forma normal de Hermite, e essa relação é denotada pelo símbolo $\cong$. Por exemplo, se $L_{\mathbf{G}}$ e $L_{\mathbf{G}'}$ são isomorfos, escrevemos $L_{\mathbf{G}} \cong L_{\mathbf{G}'}$.

2.2 Problemas Difíceis Envolvendo Reticulados

A dificuldade de se resolver certos problemas computacionais relacionados a reticulados pode ser aproveitada na construção de sistemas criptográficos. Nestes casos, a segurança é sustentada pelo fato de que atacar com sucesso o algoritmo empregado pelo sistema em questão implica em resolver de forma eficiente o problema difícil no qual o algoritmo se baseia. Dois problemas fundamentais em reticulados são o Problema do Vetor Mínimo (SVP) e o Problema do Vetor Mais Próximo (CVP):

- **Problema do Vetor Mínimo (SVP):** encontrar um vetor não nulo de menor comprimento em L, ou seja, encontrar um vetor $\mathbf{v} \in L$ que minimize a norma Euclidiana $||\mathbf{v}||$.
- **Problema do Vetor Mais Próximo (CVP)**: dado um vetor $\mathbf{u} \in \mathbb{R}^m - L$, encontrar o vetor $\mathbf{v} \in L$ mais próximo de $\mathbf{u}$, isto é, encontrar $\mathbf{v}$ que minimize a norma Euclidiana $||\mathbf{u} - \mathbf{v}||$.

Existem também variações do SVP e do CVP que costumam despertar interesse, como as versões de aproximação e decisão do SVP, ambas descritas a seguir:

[1]Existem definições nas quais a matriz é triangular superior.

- **Problema do Vetor Mínimo Aproximado (apprSVP):** seja $\mathbf{v}$ o menor vetor não nulo de L. Dado um fator de aproximação $\gamma > 1$, encontrar um vetor não nulo $\mathbf{u} \in L$ cujo comprimento não exceda $\gamma||\mathbf{v}||$.

- **Problema de Decisão do Vetor Mínimo (GapSVP):** dado um inteiro d, um fator $\gamma > 1$ e um reticulado L, fornecer SIM como saída se existe um vetor $\mathbf{v}$ não nulo em L tal que $||\mathbf{v}|| \leq d$, ou NÃO se para todo $\mathbf{v}$ não nulo em L, $||\mathbf{v}|| > \gamma d$.

Sabe-se que o problema do vetor mínimo é NP-difícil e NP-difícil de ser aproximado por um fator constante, conforme demonstrado, respectivamente, por Ajtai em 1998 [4] e por Khot em 2004 [39]. Conjectura-se que resolver o SVP com fatores de aproximação polinomiais em função da dimensão do reticulado também constitua um problema difícil. O LLL [41], um dos algoritmos mais conhecidos para problemas envolvendo reticulados, obtém uma solução em tempo polinomial para o SVP considerando um fator de aproximação de $(2/\sqrt{3})^n$, onde n representa a dimensão do reticulado [35]. Já o melhor algoritmo para se obter uma solução exata do SVP requer tempo $2^{O(n)}$ [3]. Em relação ao CVP, demonstrou-se que qualquer dificuldade para ser resolver o SVP implica na mesma dificuldade para o CVP [25].

O caráter NP-difícil de um problema por si só nem sempre é suficiente para garantir a segurança de um sistema nele baseado. Trata-se apenas de um indicativo de que no pior caso o problema provavelmente não pode ser resolvido eficientemente por um algoritmo polinomial. Entretanto, na maioria dos casos é preciso lidar com elementos selecionados aleatoriamente, como chaves e mensagens, e a NP-dificuldade não garante que o caso médio será igualmente difícil.

Em 1996, Ajtai [1] demonstrou que a complexidade no caso médio de alguns problemas envolvendo reticulados é equivalente à dificuldade do pior caso. A partir deste resultado, construiu em conjunto com Dwork [2] o primeiro sistema criptográfico cuja segurança podia ser provada utilizando exclusivamente a complexidade do pior caso de uma certa versão do SVP.

Exemplo 1. Um exemplo de aplicação de problemas em reticulados no desenvolvimento de algoritmos criptográficos é o sistema GGH [23]. Sua ideia geral consiste em utilizar como chave privada uma base "boa" de um reticulado L e como chave pública uma base "ruim" de L. Considera-se como "boa", uma base cujos vetores sejam razoavelmente

ortogonais dois a dois, enquanto que uma base "ruim" é aquela na qual o ângulo entre os vetores é relativamente pequeno.

No sistema GGH, a mensagem a ser enviada é tratada como um vetor $\mathbf{m}$, cujos membros são utilizados como coeficientes de uma combinação linear dos elementos da base "ruim", resultando em um vetor $\mathbf{v} \in L$. Somando-se a $\mathbf{v}$ uma pequena perturbação aleatória, obtém-se um segundo vetor $\mathbf{w} \notin L$, levemente diferente de $\mathbf{v}$. Dessa forma, para recuperar $\mathbf{v}$ a partir de $\mathbf{w}$, é preciso resolver o CVP, o que constitui uma tarefa computacionalmente difícil. Por outro lado, tal problema pode ser resolvido a partir do algoritmo de Babai [6] sob a condição especial de se conhecer uma base "boa" de L. Dessa forma, somente o detentor da chave privada consegue reconstruir $\mathbf{m}$.

Outros dois problemas envolvendo reticulados são o Problema dos Menores Vetores Linearmente Independentes (SIVP) e o Problema da Solução Inteira Curta (SIS).

- **Problema dos Menores Vetores Linearmente Independentes (SIVP):** dada uma base $\mathbf{B} \in \mathbb{Z}^{n \times n}$ de um reticulado, encontrar n vetores linearmente independentes $\mathbf{S} = [\mathbf{s}_1, ..., \mathbf{s}_n]$ (onde $\mathbf{s}_i \in L(\mathbf{B})$ para todo i) que minimizem a quantidade $||\mathbf{S}|| = \max_i ||\mathbf{s}_i||$.

- **Problema da Solução Inteira Curta (SIS):** dados um limite $\varepsilon > 0$ e um reticulado $L_q^{\perp}(\mathbf{A})$, onde $\mathbf{A} \in \mathbb{Z}^{n \times m}$ e q é um número primo, encontrar um vetor $\mathbf{v} \in L_q^{\perp}(\mathbf{A})$, tal que $||\mathbf{v}|| \leq \varepsilon$.

Acerca do SIVP, sabe-se que este problema é NP-difícil de ser aproximado por qualquer fator constante [7] e que o melhor algoritmo para obter uma solução exata leva tempo em torno de $n!$, onde n é a dimensão do reticulado [48]. Ademais, conjectura-se que o SIVP seja difícil de ser aproximado por fatores polinomiais. Em 2007, foi demonstrado que resolver o SIS para uma matriz $\mathbf{A}$ arbitrária é pelo menos tão difícil quanto aproximar o SIVP por um fator polinomial [49]. Além disso, provou-se que se existe um algoritmo polinomial que resolva o SIS no caso médio, então existe um algoritmo polinomial que resolva o GapSVP no pior caso, para um fator de aproximação $\tilde{O}(\gamma n)$.

Por fim, podemos mencionar problemas relacionados a representações por matriz de Gram e forma normal de Hermite:

- **Problema da Fatoração da Matriz de Gram (GMFP):** dada uma representação por matriz de Gram $\mathbf{G}$ de um reticulado L, encontrar uma base $\mathbf{B}$ de L que

satisfaça $\mathbf{G} = \mathbf{B}\mathbf{B}^T$.

- **Problema de Isomorfismo de Reticulados (LIP):** dadas duas matrizes de Gram $\mathbf{G}$ e $\mathbf{G}'$ de dois reticulados $L_{\mathbf{G}}$ e $L_{\mathbf{G}'}$, determinar se $L_{\mathbf{G}}$ e $L_{\mathbf{G}'}$ são isomorfos.

A forma normal de Hermite pode ser facilmente calculada quando uma base do reticulado é conhecida. Dessa forma, encontrar uma solução para o GMFP implica em resolver o LIP, pois basta comparar as formas normais de Hermite para determinar se dois reticulados são isomorfos. Para a classe específica de reticulados hiperbólicos, isto é, aqueles cujas matrizes de Gram são da forma $\mathbf{G} = \mathbf{U}\mathbf{U}^T$, onde $\mathbf{U}$ é uma matriz unimodular, Szydlo [63] provou que o GMFP é equivalente ao SVP e que existe uma redução polinomial do LIP para o GMFP. Ainda não se sabe se esses resultados valem para outras classes de reticulados, ou qual seria a dificuldade de resolver tais problemas no caso geral. Szydlo demonstrou também que se o LIP pudesse ser resolvido, então os esquemas de assinatura GGH [23] e NtruSign [32] seriam inseguros, o que a princípio reforçaria a ideia de que trata-se de um problema difícil. Posteriormente, provou-se que ambos os esquemas GGH e NtruSign apresentavam falhas de segurança [53], embora este fato não implique em uma maior facilidade para resolver o LIP.

2.3 Reticulados Ideais

Uma classe de reticulados que tem despertado particular interesse no desenvolvimento de sistemas e protocolos criptográficos é a dos *reticulados ideais*. Nesta seção, iremos definir o que são reticulados ideais, apresentar os principais conceitos a eles relacionados, e comentar como os mesmos podem ser aproveitados em construções criptográficas.

2.3.1 Conceitos Básicos de Álgebra

A definição de reticulados ideais, embora simples, está atrelada a alguns conceitos algébricos fundamentais, como os de grupos e anéis. Por esse motivo, a seguir serão apresentadas as definições básicas de álgebra que antecedem a de reticulados ideais. Existem diversas propriedades e resultados interessantes relacionadas a esses conceitos. Porém, aqui nos limitaremos apenas a apresentar os pontos necessários para a compreensão deste trabalho.

Grupo e Subgrupo

Seja G um conjunto e $*$ uma operação binária sobre os elementos de G. O par $(G, *)$ é um *grupo* se satisfaz as seguintes propriedades:

1. A operação $*$ é associativa: para quaisquer elementos a, b e c de G, temos que $a * (b * c) = (a * b) * c$.

2. Existe um elemento neutro e, tal que $a * e = e * a = a$, para todo elemento a de G.

3. Para todo elemento a de G, existe um elemento simétrico, ou inverso, a', tal que $a * a' = a' * a = e$.

Frequentemente, a operação $*$ é omitida, e denotamos o grupo simplesmente a partir do conjunto de elementos (por exemplo, grupo G).

Um grupo é dito *abeliano* (ou *comutativo*) se a operação $*$ é comutativa. Ou seja, para quaisquer elementos a e b de um grupo abeliano G, temos que $a * b = b * a$.

Usualmente, quando a operação do grupo é representada graficamente pelo símbolo $+$, dizemos que trata-se de um *grupo aditivo*. Além disso, denotamos o elemento neutro por 0 e o simétrico por $-a$, de forma que $a + 0 = 0 + a = a$ e $a + (-a) = -a + a = 0$. Similarmente, nos casos em que a operação é denotada pelo símbolo $*$, dizemos se tratar de um *grupo multiplicativo*. O elemento neutro é denotado então por 1 e muitas vezes chamado de *identidade*: $a * 1 = 1 * a = a$. Já o elemento simétrico, denominado *inverso*, costuma ser denotado por a^{-1}, de modo que $a * a^{-1} = a^{-1} * a = 1$.

Além disso, se G' é um subconjunto de G e o par $(G', *)$ constitui um grupo, então dizemos de G' é um *subgrupo* do grupo G. Ademais, G' é dito um *subgrupo próprio* de G se $G' \neq G$.

Anel e Corpo

Dado um conjunto R e duas operações binárias $+$ e $*$ sobre os elementos de R, dizemos que $(R, +, *)$ é um *anel*, se:

1. $(R, +)$ é um grupo abeliano com elemento neutro denotado por 0.

2. A operação $*$ é associativa: para quaisquer elementos a, b e c de R, temos que $a * (b * c) = (a * b) * c$.

3. Existe uma identidade multiplicativa 1, diferente de 0, tal que $1 * a = a * 1 = a$.

4. A operação $*$ é distributiva sobre $+$: para quaisquer elementos a, b e c de R, temos que $a * (b + c) = (a * b) + (a * c)$ e $(b + c) * a = (b * a) + (c * a)$.

Um anel é dito *comutativo* se a operação $*$ é comutativa, isto é, se $a * b = b * a$ para quaisquer elementos a e b do anel. Assim como no caso dos grupos, usualmente se omitem as operações $+$ e $*$, denotando o anel $(R, +, *)$ simplesmente por R.

Um anel comutativo é chamado de *corpo* se todos os seus elementos diferentes de zero possuem um inverso multiplicativo. Isto é, se para todo $x \neq 0$, existe um y tal que $x * y = 1$.

Polinômio e Operações Modulares

Dado um anel comutativo R, um *polinômio* sobre o anel R pode ser descrito como uma expressão da seguinte forma:

$$f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n,$$

onde $a_i \in R$, para $i = 0, 1, ..., n$, e $n \geq 0$. Os elementos a_i são ditos os coeficientes de x^i e o maior m para o qual $a_m \neq 0$ é denominado *grau* do polinômio $f(x)$.

A realização de operações módulo $f(x)$ entre polinômios é análoga às operações módulo n entre números inteiros. Na aritmética modular convencional, parte-se do princípio que qualquer número inteiro pode ser expresso na forma de um dividendo, tal que:

$$dividendo = quociente \times divisor + resto,$$

de modo que o resultado da operação módulo n corresponde sempre ao resto da divisão por n. Em se tratando de polinômios, também leva-se em conta tal expressão, pois qualquer polinômio $g(x)$ pode ser escrito como:

$$g(x) = q(x)\, f(x) + r(x),$$

com o grau de $r(x)$ sendo sempre menor que o grau de $f(x)$. Por exemplo, considerando os polinômios com coeficientes inteiros, podemos exprimir $g(x) = x^3 - 2x^2 + 3x - 6$ como $g(x) = (x - 2)(x^2 + 1) + (2x - 4)$. Assim, o resultado da multiplicação do polinômios $h(x) = x - 2$ por $i(x) = x^2 + 3$, convencionalmente dado por $g(x)$, passa a ser $2x - 4$ quando a operação é realizada módulo $f(x) = x^2 + 1$.

Representação Vetorial de Polinômios

Dado um polinômio $f(x) = a_0 + a_1x + a_2x^2 + ... + a_{n-1}x^{n-1}$, podemos representá-lo como um vetor $\mathbf{f}$ de dimensão n, onde cada coordenada de $\mathbf{f}$ equivale a um coeficiente de $f(x)$. Isto é, $\mathbf{f} = (a_0, a_1, ..., a_{n-1})$. Neste texto, frequentemente iremos nos referir a vetores como sendo polinômios e vice-versa, ficando subentendida a forma de representação previamente mencionada. Desta forma, como reticulados são conjuntos de vetores, um reticulado de dimensão n pode também ser tratado como um conjunto de polinômios de grau $n - 1$.

Anel de Polinômios

Dizemos que o *anel de polinômios* $R[x]$ é o anel formado por todos os polinômios com coeficientes em R, com as duas operações do anel sendo dadas pela adição e multiplicação padrão de polinômios, porém com a aritmética sobre os coeficientes sendo realizadas em R. Por exemplo, $\mathbb{N}[x]$ é o anel de polinômios cujos coeficientes são números naturais.

Além disso, dado um anel de polinômios $F[x]$, onde F é um corpo, e um polinômio $f(x)$ de grau n, denota-se por $F[x]/\langle f(x)\rangle$ o conjunto de polinômios em $F[x]$ com grau menor do que n. Neste caso, as operações de adição e multiplicação entre os elementos são realizadas módulo $f(x)$. Sabe-se que $F[x]/\langle f(x)\rangle$ também constitui um anel comutativo [46].

Ideal de um Anel

Para um anel arbitrário $(R, +, *)$, um subconjunto I de R é chamado *ideal à direita*, se:

1. $(I, +)$ é um subgrupo de $(R, +)$
2. $x * r$ está em I para todo x em I e r em R

e *ideal à esquerda* se:

1. $(I, +)$ é um subgrupo de $(R, +)$
2. $r * x$ está em I para todo x em I e r em R

Se o anel é comutativo, então as definições de ideal à direita e à esquerda coincidem, e denominamos I simplesmente como *ideal de R.*

2.3.2 Definição e Obtenção de Reticulados Ideais

Considere o anel $R = \mathbb{Z}[x]/\langle f \rangle$ de polinômios com coeficientes inteiros, módulo alguma função polinomial mônica[2] f de grau n. Ideais em R são subgrupos correspondentes a reticulados denominados *reticulados ideais*.

Retomando a forma de se definir um reticulado dada pela equação 2.3, podemos obter reticulados ideais como $L_q^{\perp}(\mathbf{A})$, ao considerar que a matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ assume uma certa forma particular [50], dada por:

$$\mathbf{A} = [\mathbf{A}^{(1)}|...|\mathbf{A}^{(m/n)}], \tag{2.4}$$

onde $\mathbf{A}^{(i)} = [\mathbf{a}^{(i)}, \mathbf{F}\mathbf{a}^{(i)}, ..., \mathbf{F}^{n-1}\mathbf{a}^{(i)}]$, $\mathbf{a}^{(i)} \in \mathbb{Z}^n$ e

$$\mathbf{F} = \left[\begin{array}{ccc|c} & \mathbf{0}^T & & \\ \hline \ddots & & & \\ & \mathbf{I} & & -\mathbf{f} \\ & & \ddots & \end{array}\right] \tag{2.5}$$

No caso, $\mathbf{f}$ é o vetor obtido a partir das coordenadas do polinômio $f(x) = f_0 + f_1 x + ... + f_{n-1}x^{n-1} + x^n$, de tal forma que $\mathbf{f} = (f_0, f_1, ..., f_{n-1})$, e o reticulado resultante equivale a um ideal em $\mathbb{Z}[x]/\langle f(x) \rangle$.

Diversos dos protocolos abordados neste trabalho utilizam reticulados da forma $L_q^{\perp}(\mathbf{A})$, de modo que modificá-los para que passem a utilizar reticulados ideais, ao invés de genéricos, pode ser feito de maneira simples apenas substituindo a matriz $\mathbf{A}$ utilizada.

2.3.3 Reticulados Cíclicos e Anticíclicos

Duas classes particulares de reticulados ideais podem ser obtidas da maneira apresentada na subseção anterior: os reticulados cíclicos e anticíclicos, os quais serão introduzidos a seguir e utilizados para exemplificar como reticulados ideais podem ser empregados na construção de algoritmos mais eficientes.

---

[2]Isto é, uma função polinomial cujo coeficiente do termo de maior grau seja 1.

Reticulados Cíclicos

Seja L um reticulado de dimensão n. Dizemos que L é um *reticulado cíclico* se para todo vetor $\mathbf{v} = (a_1, ..., a_n)$ de L, o vetor $\mathbf{u} = (a_n, a_1, ..., a_{n-1})$ também está em L. Além disso, dizemos que o vetor $\mathbf{u}$ é obtido a partir de um *deslocamento cíclico para a direita* das coordenadas de $\mathbf{v}$. Similarmente, $\mathbf{v}$ pode ser obtido a partir de um *deslocamento cíclico para a esquerda* das coordenadas de $\mathbf{u}$. Nestes casos, o deslocamento foi de um elemento, apenas. O vetor $\mathbf{w} = (a_{n-2}, a_{n-1}, a_n, a_1, ..., a_{n-3})$, por outro lado, corresponde ao deslocamento cíclico de $\mathbf{v}$ em três coordenadas (ou elementos) para a direita, e ao deslocamento das coordenadas de $\mathbf{u}$ em dois elementos, também para a direita.

Uma *matriz circulante* é uma matriz na qual cada linha, vista como um vetor, corresponde à linha acima com as coordenadas rotacionadas de um elemento para a direita (Figura 2.2a). Tais matrizes correspondem a um caso especial das matrizes de Toeplitz, nas quais diagonais da esquerda para a direita são constantes (Figura 2.2b). De forma geral, podemos dizer que matrizes circulantes são matrizes de Toeplitz com a propriedade adicional de que $a_i = a_{i+n+1}$, para $-n \leq i < 0$.

$$\begin{bmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \\ a_n & a_0 & a_1 & \cdots & a_{n-1} \\ a_{n-1} & a_n & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & a_1 \\ a_1 & a_2 & \cdots & a_n & a_0 \end{bmatrix}$$

(a) Uma matriz circulante

$$\begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} & a_n \\ a_{-1} & a_0 & a_1 & a_2 & & a_{n-1} \\ a_{-2} & a_{-1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & a_{-2} & \ddots & \ddots & \ddots & a_2 \\ \vdots & & \ddots & \ddots & \ddots & a_1 \\ a_{-n} & \cdots & \cdots & a_{-2} & a_{-1} & a_0 \end{bmatrix}$$

(b) Uma matriz de Toeplitz genérica

Figura 2.2: Matrizes de Toeplitz

Um exemplo de matriz circulante é a matriz $\mathbf{T}$ exibida abaixo. Facilmente percebe-se que esta matriz corresponde à matriz $\mathbf{F}$ apresentada anteriormente na equação 2.5, para o caso particular em que $\mathbf{f} = (-1, 0, ..., 0)$.

$$\mathbf{T} = \left[\begin{array}{ccc|c} & \mathbf{0}^T & & 1 \\ \hline \ddots & & & 0 \\ & \mathbf{I} & & \vdots \\ & & \ddots & 0 \end{array}\right]$$

Por ser composta apenas por zeros e uns em posições específicas, ao ser multiplicada por qualquer vetor, a matriz $\mathbf{T}$ tem como resultado uma rotação cíclica em um elemento das coordenadas do vetor original.

$$\mathbf{Ta}^{(i)} = \left[\begin{array}{ccccc|c} 0 & \cdots & & \cdots & 0 & 1 \\ \hline 1 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 1 & 0 & & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \\ 0 & & 0 & 1 & 0 & \vdots \\ 0 & \cdots & \cdots & \ddots & 0 & 1 \end{array}\right] \left[\begin{array}{c} 0 \\ \\ \\ \\ \\ \\ \end{array}\right] \left[\begin{array}{c} a_1^{(i)} \\ a_2^{(i)} \\ \vdots \\ \\ \vdots \\ a_n^{(i)} \end{array}\right] = \left[\begin{array}{c} a_n^{(i)} \\ a_1^{(i)} \\ a_2^{(i)} \\ \vdots \\ \vdots \\ a_{n-1}^{(i)} \end{array}\right]$$

Desta forma, qualquer matriz $\mathbf{A}^{(i)}$ definida a partir de um vetor $\mathbf{a}^{(i)}$ e $\mathbf{T}$ como $\mathbf{A}^{(i)} = [\mathbf{a}^{(i)}, \mathbf{Ta}^{(i)}, ..., \mathbf{T}^{n-1}\mathbf{a}^{(i)}]$ é uma matriz circulante, e a matriz $\mathbf{A} = [\mathbf{A}^{(i)}|...|\mathbf{A}^{(m/n)}]$ (como na equação 2.4) é dita uma *matriz circulante por blocos* (pois cada bloco $\mathbf{A}^{(i)}$ é uma matriz circulante). Além disso, o reticulado $L_q^{\perp}(\mathbf{A})$ corresponde a um ideal em $R = \mathbb{Z}[x]/\langle x^n - 1\rangle$, sendo portanto um reticulado ideal.

Seja $v(x) = a_0 + a_1x + ... + a_{n-1}x^{n-1}$ um elemento de $L_q^{\perp}(\mathbf{A})$. Sabemos que $u(x) = x$ é um elemento de $\mathbb{Z}[x]/\langle x^n - 1\rangle$ e que, portanto, $v'(x) = v(x).u(x)$ pertence a $L_q^{\perp}(\mathbf{A})$, dada a definição de ideal de um anel. Normalmente, teríamos:

$$v'(x) = v(x).u(x) = a_0x + a_1x^2 + ... + a_{n-1}x^n,$$

de forma que ao realizar a operação módulo $x^n - 1$:

$$v'(x) = a_{n-1} + a_0x + a_1x^2 + ... + a_{n-2}x^{n-1}$$

Ou seja, para todo $\mathbf{v} = (a_0, a_1, ...a_{n-1})$ pertencente a $L_q^{\perp}(\mathbf{A})$, sabemos que o vetor $\mathbf{v'} = (a_{n-1}, a_0, a_1, ...a_{n-2})$ também está em $L_q^{\perp}(\mathbf{A})$. Dado que $\mathbf{v'}$ corresponde a um deslocamento cíclico para a direita das coordenadas de $\mathbf{v}$, temos que reticulados correspondentes a ideais em $R = \mathbb{Z}[x]/\langle x^n - 1\rangle$ são cíclicos, às vezes chamados *reticulados ideais cíclicos*.

Reticulados Anticíclicos

Seja L um reticulado de dimensão n. Dizemos que L é um *reticulado anticíclico* se para todo vetor $\mathbf{v} = (a_1, ..., a_n)$ de L, o vetor $\mathbf{u} = (-a_n, a_1, ..., a_{n-1})$ também está em L. Esta

definição é bastante semelhante à de reticulados cíclicos, com a exceção de que a última coordenada do vetor original passa a ter, no vetor deslocado, o sinal alterado.

Uma forma de obter o vetor $\mathbf{u}$ a partir de $\mathbf{v}$ corresponde à multiplicação da matriz $\mathbf{F}$ abaixo por $\mathbf{v}$, onde $\mathbf{F}$ se difere de uma matriz circulante também apenas pelo sinal de um de seus elementos:

$$\mathbf{F} = \left[\begin{array}{cccc|c} 0 & \cdots & \cdots & 0 & -1 \\ \hline 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \vdots & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & & 1 & 0 \end{array}\right]$$

Como $\mathbf{F}$ é da forma da equação 2.5, com $\mathbf{f} = (1, 0, ..., 0)$, então $L_q^{\perp}(\mathbf{A})$ (com $\mathbf{A}$ definido como na equação 2.4) é um reticulado ideal de $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$.

Assim como no caso de reticulados cíclicos, temos que para todo $v(x) = a_0 + a_1 x + ... + a_{n-1}x^{n-1}$ pertencente a $L_q^{\perp}(\mathbf{A})$, dado $u(x) = x \in \mathbb{Z}[x]/\langle x^n + 1\rangle$, temos que $v'(x) = v(x).u(x)$ é um elemento de $L_q^{\perp}(\mathbf{A})$. Realizando a operação módulo $x^n + 1$:

$$v'(x) = -a_{n-1} + a_0 x + a_1 x^2 + ... + a_{n-2}x^n$$

Portanto, reticulados correspondentes a ideais em $\mathbb{Z}[x]/\langle x^n + 1\rangle$ são reticulados anti-cíclicos.

2.3.4 Aplicação de Reticulados Ideais

Dadas as definições que concernem reticulados ideais e as duas categorias de exemplos de tais reticulados, podemos comentar brevemente no que o uso dessas estruturas impacta nos protocolos e algoritmos criptográficos que os empregam, em relação a versões que utilizam reticulados genéricos[3].

Representação Sucinta

Em primeiro lugar, se observarmos a forma de se obter um reticulado ideal a partir da matriz $\mathbf{A}$ dada pela Equação 2.4, notamos que é suficiente especificar apenas os vetores

[3]Isto é, não necessariamente reticulados ideais.

$a^{(i)}$ para se definir a matriz completa[4]. Isso significa que a partir de m/n vetores de dimensão n, isto é, de m coeficientes, podemos definir completamente uma matriz de dimensão $n \times m$.

Utilizando um anel que resulte em um reticulado cíclico ou anticíclico, como os mencionados anteriormente, obtemos então uma representação sucinta da matriz, visto que todas as colunas de um bloco podem ser facilmente obtidas a partir de deslocamentos cíclicos da primeira coluna do bloco, com eventuais inversões de sinal, no caso dos reticulados anticíclicos. Para os demais casos, se f for tal que exista uma relação entre $a^{(i)}$ e $\mathbf{F}a^{(i)}$ simples de ser determinada (como os deslocamentos cíclicos), também é possível tirar proveito de uma representação mais sucinta da matriz.

Multiplicação Eficiente

A maior vantagem, possivelmente, do uso de reticulados ideais, se dá na eficiência obtida nas operações de multiplicação. Conforme será apresentado, muitos dos protocolos estudados envolvem a multiplicação da matriz $\mathbf{A}$ que define o reticulado (Equação 2.4) por um vetor de inteiros $\mathbf{x}$ (algumas vezes binário). Nestes casos, a estrutura especial de $\mathbf{A}$ pode ser aproveitada para que esta multiplicação seja realizada mais rapidamente.

Assim como separamos a matriz $\mathbf{A}$ em blocos $\mathbf{A}^{(i)}$, podemos dividir este vetor $\mathbf{x}$ em m/n vetores menores de dimensão n, os quais chamaremos de $\mathbf{x}^{(i)}$. Desse modo, a multiplicação $\mathbf{Ax}$ pode ser facilmente obtida como a soma das multiplicações $\mathbf{A}^{(i)}\mathbf{x}^{(i)}$.

Tomando como exemplo os reticulados cíclicos, dado que $\mathbf{A}^{(i)}$ é uma matriz circulante com primeira coluna dada por $a^{(i)}$, então a multiplicação $\mathbf{A}^{(i)}\mathbf{x}^{(i)}$ é obtida pela convolução circular de $a^{(i)}$ e $\mathbf{x}^{(i)}$:

$$\mathbf{y} = \mathbf{A}^{(i)}\mathbf{x}^{(i)} = \mathbf{a}^{(i)} * \mathbf{x}^{(i)}$$

Exemplo 2. Considere o caso em que $n = 3$ e $a^{(i)} = [\ a_0 \quad a_1 \quad a_2\ ]^T$, $\mathbf{x}^{(i)} = [\ b_0 \quad b_1 \quad b_2\ ]^T$ e $\mathbf{y} = [\ y_0 \quad y_1 \quad y_2\ ]^T$. Realizando diretamente a multiplicação da matriz $\mathbf{A}^{(i)}$ por $\mathbf{x}^{(i)}$, temos que o valor de $\mathbf{y}$ é:

[4]Além do polinômio f, que define o anel com o qual se está trabalhando, naturalmente.

$$\mathbf{x} = \underbrace{\left[\begin{array}{ccc} x_1 & \ldots & x_n \end{array}\right.}_{\mathbf{x}^{(1)}} \Big| \quad \ldots \quad \Big| \underbrace{\left.\begin{array}{ccc} x_{m-n+1} & \ldots & x_m \end{array}\right]^T}_{\mathbf{x}^{(m/n)}}$$

$$\mathbf{Ax} = \left[\begin{array}{ccc} \mathbf{A}^{(1)} & \cdots & \mathbf{A}^{(m/n)} \end{array}\right] \left[\begin{array}{c} \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(m/n)} \end{array}\right] = \mathbf{A}^{(1)}\mathbf{x}^{(1)} + \ldots + \mathbf{A}^{(m/n)}\mathbf{x}^{(m/n)}$$

Figura 2.3: Divisão do vetor $\mathbf{x}$ em vetores menores e multiplicação de $\mathbf{A}$ por $\mathbf{x}$.

$$\mathbf{y} = \left[\begin{array}{c} y_0 \\ y_1 \\ y_2 \end{array}\right] = \left[\begin{array}{ccc} a_0 & a_2 & a_1 \\ a_1 & a_0 & a_2 \\ a_2 & a_1 & a_0 \end{array}\right] \left[\begin{array}{c} b_0 \\ b_1 \\ b_2 \end{array}\right] = \left[\begin{array}{c} a_0b_0 + a_1b_2 + a_2b_1 \\ a_0b_1 + a_1b_0 + a_2b_2 \\ a_0b_2 + a_1b_1 + a_2b_0 \end{array}\right]$$

Para calcular a convolução circular $a^{(i)} * \mathbf{x}^{(i)}$, estendemos $\mathbf{x}^{(i)}$ como uma função discreta periódica:

$$\mathbf{x}^{(i)} = (..., b_0, b_1, b_2, b_0, b_1, b_2, ...)$$

Neste caso, quando escrevermos b_i, com $i \notin \{0, 1, 2\}$, queremos dizer a i-ésima entrada de $\mathbf{x}^{(i)}$ após a expansão, de forma que $b_i = b_{i \bmod n}$ (por exemplo, $b_4 = b_1$ e $b_{-1} = b_2$). Para $a^{(i)}$, que será vista como uma função não periódica, consideraremos $a_i = 0$, se $i \notin \{0, 1, 2\}$.

Da definição de convolução, os y_i são dados por $y_i = \sum_{j=-\infty}^{\infty} a_j b_{i-j}$. Assim:

$$\mathbf{y} = \left[\begin{array}{c} y_0 \\ y_1 \\ y_2 \end{array}\right] = \left[\begin{array}{c} a_0b_0 + a_1b_{-1} + a_2b_{-2} \\ a_0b_1 + a_1b_0 + a_2b_{-1} \\ a_0b_2 + a_1b_1 + a_2b_0 \end{array}\right] = \left[\begin{array}{c} a_0b_0 + a_1b_2 + a_2b_1 \\ a_0b_1 + a_1b_0 + a_2b_2 \\ a_0b_2 + a_1b_1 + a_2b_0 \end{array}\right]$$

Dessa forma, o resultado da convolução circular é o mesmo obtido através da multiplicação direta da matriz $\mathbf{A}^{(i)}$ por $\mathbf{x}^{(i)}$.

Ainda considerando reticulados cíclicos, outra maneira de se calcular $\mathbf{A}^{(i)}\mathbf{x}^{(i)}$ a partir de $a^{(i)}$ consiste em considerar $a^{(i)}$ e $\mathbf{x}^{(i)}$ como polinômios e realizar a multiplicação desses dois elementos. Dado que os dois polinômios possuem grau n, sua multiplicação resultaria em um polinômio de grau $2n$. Porém, lembrando que reticulados cíclicos equivalem a ideais no anel $\mathbb{Z}[x]/\langle x^n - 1\rangle$, então os termos x^{n+k}, $k = 0, ..., n$ equivalem a x^k.

Exemplo 3. Considerando ainda o caso em que $n = 3$, podemos escrever $a^{(i)} = a_0 + a_1x + a_2x^2$ e $\mathbf{x} = b_0 + b_1x + b_2x^2$, de modo que:

$$\begin{aligned} y(x) = (a_0 + a_1x + a_2x^2)(b_0 + b_1x + b_2x^2) &= a_ob_0 + \\ &(a_0b_1 + a_1b_0)x + \\ &(a_0b_2 + a_1b_1 + a_2b_0)x^2 + \\ &(a_1b_2 + a_2b_1)x^3 + \\ &a_2b_2x^4 \end{aligned}$$

o que no anel $\mathbb{Z}[x]/\langle x^3 - 1\rangle$ equivale a:

$$\begin{aligned} y(x) &= a_ob_0 + a_1b_2 + a_2b_1 + \\ &(a_0b_1 + a_1b_0 + a_2b_2)x + \\ &(a_0b_2 + a_1b_1 + a_2b_0)x^2 \end{aligned}$$

Assim, a representação vetorial de $y(x)$ equivale ao vetor $\mathbf{y} = \mathbf{A}^{(i)}\mathbf{x}^{(i)}$.

Considerando a multiplicação de dois polinômios $a(x)$ e $b(x)$ de grau n sem nos restringirmos a nenhum anel, sabemos que para $c(x) = a(x).b(x)$, o vetor de coeficientes de $c(x)$ pode ser obtido a partir da *convolução* dos vetores correspondentes a $a(x)$ e $b(x)$. Isto é:

$$c_i = \sum_{j=0}^{i} a_j b_{i-j}, \tag{2.6}$$

para $i = 0, ..., 2n$, onde a_k, b_k, c_k correspondem aos coeficientes de x^k em $a(x)$, $b(x)$ e $c(x)$ respectivamente[5].

Dessa forma, no anel $\mathbb{Z}[x]/\langle x^n - 1\rangle$ também podemos utilizar esta relação, porém lembrando que os termos x^{n+k} , com $k = 0, ..., n$, equivalem a x^k. Ou seja, tanto a definição de convolução cíclica quanto da convolução tradicional podem ser aplicadas à multiplicação em reticulados cíclicos.

No caso de reticulados anticíclicos correspondentes a ideais no anel $\mathbb{Z}[x]/\langle x^n + 1\rangle$, devemos considerar que os termos x^{n+k} , $k = 0, ..., n$ equivalem a $-x^k$, e então também podemos utilizar a convolução de $a^{(i)}$ e $\mathbf{x}^{(i)}$ para calcular $\mathbf{A}^{(i)}\mathbf{x}^{(i)}$:

$$\mathbf{y} = \mathbf{A}^{(i)}\mathbf{x}^{(i)} = a^{(i)} * \mathbf{x}^{(i)}.$$

Exemplo 4. Tomando novamente o caso em que $n = 3$, porém considerando agora o anel $\mathbb{Z}[x]/\langle x^3 + 1\rangle$, iremos calcular a convolução dos coeficientes de $a^{(i)} = a_0 + a_1x + a_2x^2$ e $\mathbf{x} = b_0 + b_1x + b_2x^2$, a partir da fórmula apresentada na Equação 2.6:

$$\begin{aligned}
c_0 &= \sum_{j=0}^{0} a_j b_{-j} = a_0b_0 \\
c_1 &= \sum_{j=0}^{1} a_j b_{1-j} = a_0b_1 + a_1b_0 \\
c_2 &= \sum_{j=0}^{2} a_j b_{2-j} = a_0b_2 + a_1b_1 + a_2b_0 \\
c_3 &= \sum_{j=0}^{3} a_j b_{3-j} = a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 = a_1b_2 + a_2b_1 \\
c_4 &= \sum_{j=0}^{4} a_j b_{4-j} = a_0b_4 + a_1b_3 + a_2b_2 + a_3b_1 + a_4b_0 = a_2b_2
\end{aligned}$$

onde utilizamos o fato de que $a_3 = a_4 = b_3 = b_4 = 0$, pois nenhum dos dois polinômios possui coeficientes não nulos para x^3 e x^4.

[5]E portanto $a_k = b_k = 0$, para $k > n$.

Como x^4 no anel $\mathbb{Z}[x]/\langle x^3+1\rangle$ equivale a $-x^1$, o coeficiente c_4 deve ser subtraído de c_1. Similarmente, x^3 equivale a $-x^0 = -1$, de forma que c_3 deve ser subtraído de c_0. Dessa forma, o resultado da convolução no anel $\mathbb{Z}[x]/\langle x^3+1\rangle$ é:

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_0 - c_3 \\ c_1 - c_4 \\ c_2 \end{bmatrix} = \begin{bmatrix} a_0b_0 - a_1b_2 - a_2b_1 \\ a_0b_1 + a_1b_0 - a_2b_2 \\ a_0b_2 + a_1b_1 + a_2b_0 \end{bmatrix}$$

Realizando a multiplicação da matriz $\mathbf{A}^{(i)}$ por $\mathbf{x}^{(i)}$, teríamos:

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_0 & -a_2 & -a_1 \\ a_1 & a_0 & -a_2 \\ a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_0b_0 - a_1b_2 - a_2b_1 \\ a_0b_1 + a_1b_0 - a_2b_2 \\ a_0b_2 + a_1b_1 + a_2b_0 \end{bmatrix},$$

que é justamente o resultado obtido através da convolução.

Pelo Teorema da Convolução [8], sabemos que a transformada de Fourier da convolução de duas funções é igual ao produto ponto a ponto das transformadas de Fourier de cada função:

$$\mathcal{F}\{p * q\} = \mathcal{F}\{p\} \,.\, \mathcal{F}\{q\}$$

Dessa forma, para determinar a multiplicação de dois polinômios, ou seja, a convolução da Equação 2.6, utilizamos a sequência de passos da Figura 2.4. Primeiramente calculamos as transformadas de Fourier dos dois polinômios (operação 1), realizamos a multiplicação ponto a ponto (operação 2) e depois calculamos a transformação inversa do resultado (operação 3).

$$p * q = \mathcal{F}^{-1}\big\{\mathcal{F}\{p\} \,.\, \mathcal{F}\{q\}\big\}$$

Sabe-se que o cálculo das operações 1 e 3 pode ser realizado em tempo $O(n \log n)$, a partir da aplicação do algoritmo da Transformada Rápida de Fourier (FFT) [10], e a multiplicação ponto a ponto da operação 2 em tempo $O(n)$. Assim, o cálculo do produto da matriz $\mathbf{A}^{(i)}$ pelo vetor $\mathbf{x}^{(i)}$ como a multiplicação dos polinômios $a^{(i)}$ e $\mathbf{x}^{(i)}$, quando possível, proporciona uma melhoria assintótica considerável no tempo necessário para se realizar a operação.

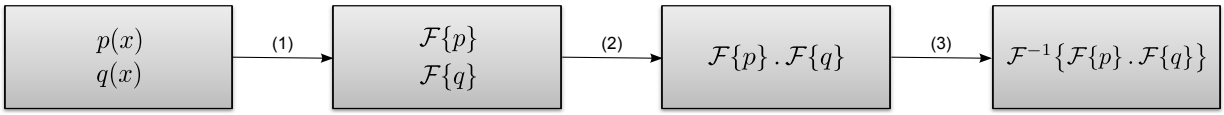


Figura 2.4: Cálculo da convolução dos coeficientes de dois polinômios

Além disso, quando a FFT é aplicada a números inteiros módulo algum primo p, podemos utilizar a sua variação que não emprega valores complexos, a NTT (do inglês, *Number-Theoretic Transform*) [12].

Problemas Envolvendo Reticulados Ideais

Apesar das vantagens práticas que o uso de reticulados ideais pode apresentar, a maior incerteza que circunda a sua aplicação é o impacto na segurança dos algoritmos. Lembrando que a segurança se baseia na dificuldade de se resolver certos problemas, essa incerteza se traduz no questionamento de se tal dificuldade é mantida quando se impõem restrições sobre os elementos utilizados. Mais especificamente, questiona-se se a estrutura particular de tais reticulados, a qual pode ser aproveitada para melhorar a demanda espaço e tempo de operações, também não pode facilitar a resolução dos problemas envolvidos.

Atualmente, embora se conheça a complexidade de se resolver certos problemas envolvendo reticulados genéricos, pouco se sabe se essa dificuldade se mantém em um domínio restrito a reticulados ideais. A princípio, encontrar soluções para alguns problemas considerando apenas reticulados ideais poderia ser potencialmente mais fácil, especialmente para determinados ideais e classes, como os reticulados cíclicos. Por exemplo, não se sabe ao certo se o SVP e o SIVP são NP-difíceis também para reticulados ideais. Entretanto, é razoável conjecturar que ambos são difíceis de serem aproximados pelo menos por fatores polinomiais, no pior caso.

Hoje, os melhores algoritmos para resolver o SVP em reticulados ideais têm tempo exponencial [59] e o algoritmo polinomial mais eficiente (LLL) obtém soluções com fatores de aproximação quase exponenciais. Isso ocorre porque o algoritmo LLL não consegue aproveitar a estrutura algébrica dos reticulados ideais, de forma que não há razão para crer que este funcione melhor mesmo quando se considera apenas reticulados cíclicos. Sob

certas condições, sabe-se que resolver o SIS em reticulados ideais é pelo menos tão difícil quanto o SVP, também em reticulados ideais. De qualquer maneira, é possível construir esquemas criptográficos mais eficientes utilizando reticulados ideais, sob a hipótese de que determinados problemas são difíceis.

Capítulo 3

Protocolos de Identificação

Neste capítulo, serão apresentados de forma resumida os principais conceitos envolvendo protocolos de identificação, incluindo sua estrutura básica, modelos de segurança e classes de ataque. Além disso, comentaremos brevemente acerca de esquemas de assinatura digital e sua relação com protocolos de identificação. A maior parte do capítulo, entretanto, será dedicada a apresentar os objetos de estudo desse trabalho, isto é, protocolos construídos com base em problemas envolvendo reticulados.

3.1 Conceitos Básicos

3.1.1 Definições

Protocolos de identificação são mecanismos que permitem que uma entidade A (o demonstrador) forneça garantias de sua identidade a uma segunda entidade B (o verificador). Adicionalmente, mesmo após um número arbitrário de execuções do protocolo, uma terceira entidade, ou até mesmo B, não pode ser capaz de utilizar a identidade de A, isto é, não deve ser capaz de fazer-se passar por A. Aplicações práticas desses protocolos incluem controle de acesso e outras aplicações que necessitem de evidência de identidade de usuários. Em alguns casos, é necessário algum dispositivo de hardware adicional e em outros somente alguma informação privada do indivíduo que se identifica (o demonstrador), como no caso de senhas de computadores.

Frequentemente, usa-se a nomenclatura *autenticação de entidade* como sinônimo da definição de identificação apresentada, muitas vezes omitindo o termo "entidade". Ao

longo deste texto, contudo, iremos sempre nos referir a protocolos de identificação.

De qualquer forma, tais protocolos usualmente empregam algum mecanismo do tipo desafio-resposta para obter provas de uma determinada identidade. Assim, o verificador B envia um desafio à entidade A a ser identificada. A resposta a esse desafio toma como base um conhecimento secreto que A possui, mas não revela o segredo em si, embora alguma informação parcial possa ser exposta. Este mecanismo encontra-se esquematizado na Figura 3.1, na qual a entidade A é detentora do segredo s e responde ao desafio c do verificador B em função dos valores de s e c. Protocolos de identificação que envolvem mecanismos de desafio-resposta costumam ser chamados de esquemas de *autenticação forte*.

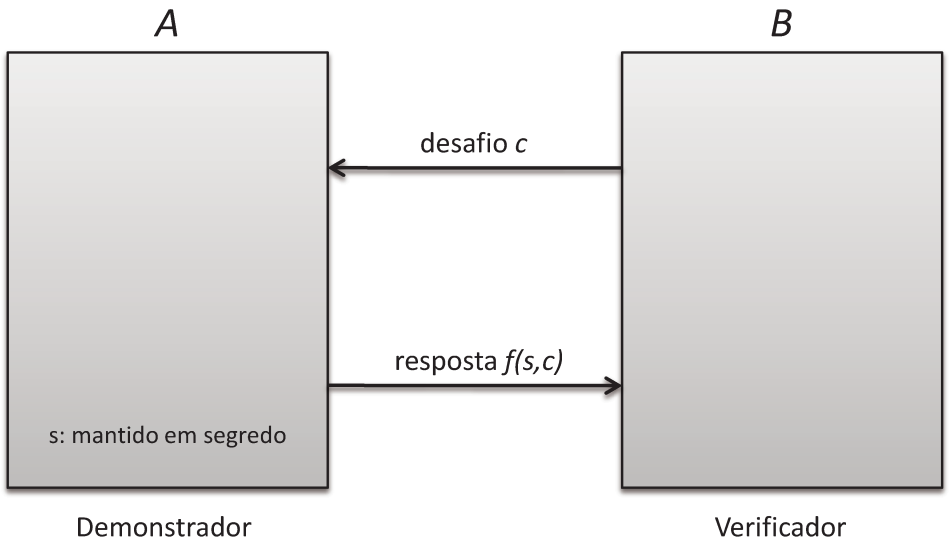


Figura 3.1: Mecanismo de desafio-resposta empregado pelos protocolos de identificação.

Nos protocolos que serão apresentados neste trabalho, para que tal mecanismo possa ser aplicado, pressupõe-se a existência de dois algoritmos. O primeiro deles consiste em um algoritmo de *geração de chaves*, no qual seleciona-se o valor secreto mantido pelo demonstrador A e um valor público correspondente, que permitirá que B faça a verificação de identidade. No Algoritmo 1 abaixo, tais valores estão representados respectivamente pela chave privada s_A e pela chave pública p_A.

Algoritmo 1: Geração de Chaves
1. Chave privada: s_A
2. Chave pública: p_A

O segundo algoritmo corresponde ao processo de identificação propriamente dito, que ocorre na forma de um protocolo interativo entre o demonstrador e o verificador. É neste momento que são trocadas as mensagens de desafio e resposta mencionadas anteriormente. Além disso, antes desse passo, costuma-se trocar um conjunto de valores, chamados de *valores de compromisso*, selecionados aleatoriamente a cada interação. No caso, tais valores são utilizados em conjunto com a chave secreta para compor a resposta ao desafio. Dessa forma, ainda que o verificador envie o mesmo desafio repetidas vezes, a resposta do demonstrador será sempre distinta, em função dos diferentes valores de compromisso.

Em muitos casos, para que se possa aceitar a prova de identidade do demonstrador com uma certa segurança, é necessário que o procedimento interativo seja repetido no mínimo um determinado número de vezes. Neste caso, dizemos que o protocolo é composto de várias *rodadas*, onde uma rodada corresponde a uma repetição do processo de interação. Ademais, para especificar o protocolo completo, geralmente é suficiente descrever somente os passos de uma rodada, conforme o esquema simplificado no Algoritmo 2 abaixo.

Algoritmo 2: Uma Rodada do Protocolo de Identificação		
Demonstrador		Verificador
1. Calcula valores de compromisso $\mathbf{x} = \{x_1, x_2..., x_n\}$	$\xrightarrow{\mathbf{x}}$	
2.	$\xleftarrow{c}$	Seleciona desafio c
3. Calcula resposta $y = f(s_A, c, x_1, ..., x_n)$	$\xrightarrow{y}$	
4.		Verifica se a resposta é válida, utilizando p_A

3.1.2 Propriedades Desejáveis

Por se tratarem de sistemas de demonstração interativos, protocolos de identificação procuram satisfazer duas propriedades: *completude* e *coerência*. A primeira delas significa que um demonstrador honesto sempre terá sua prova de identidade aceita pelo verificador. Conforme será visto adiante, alguns protocolos obtém certos tipos de ganho em detrimento dessa propriedade. Nestes casos, diz-se que o protocolo apresenta um *erro de completude*, que representa a probabilidade de um demonstrador legítimo ser rejeitado. Quando não há erro de completude[1], dizemos que o protocolo apresenta *completude perfeita.* A propriedade de *coerência*, por sua vez, requer que o verificador não aceite a prova de identidade de um impostor, exceto por uma pequena probabilidade, cujo limite superior é denominado *erro de coerência*.

Conforme mencionado anteriormente, ao responder o desafio do verificador, embora o demonstrador não revele sua chave secreta, a mesma é utilizada para compor a resposta, de forma que alguma informação parcial pode ser exposta indesejadamente. Dizemos que os métodos que não permitem transferência sequer parcial de informações que possam resultar na dedução do segredo apresentam a propriedade de *conhecimento zero*. Nesses casos, a entidade A demonstra conhecimento do valor secreto, sem no entanto revelar nenhuma informação adicional além deste conhecimento.

Formalmente, dizemos que um protocolo apresenta a propriedade de conhecimento zero se existe um algoritmo que, sem nenhuma interação com o demonstrador legítimo, é capaz de produzir saídas indistinguíveis das resultantes da interação com este demonstrador. Ou seja, qualquer conhecimento que poderia ser obtido ao interagir com o demonstrador verdadeiro, também poderia ter sido simulado sem nenhum acesso a este.

Neste sentido, como nem sempre é possível atingir esta situação ideal, definem-se os seguintes tipos de conhecimento zero:

- **Conhecimento zero perfeito:** quando as distribuições das saídas produzidas pelo simulador e pelo protocolo real são idênticas.
- **Conhecimento zero estatístico:** quando as distribuições não são idênticas, mas estatisticamente próximas, isto é, sua diferença estatística está limitada por uma

[1]Isto é, quando o erro de completude é igual a zero.

função cujo valor é assintoticamente zero.

- **Conhecimento zero computacional:** quando as distribuições não podem ser diferenciadas por nenhum algoritmo eficiente, isto é, com tempo de execução polinomial.

Protocolos de conhecimento zero satisfazem as três propriedades já mencionadas, ou seja, completude, coerência e conhecimento zero. Por este motivo, a prova oferecida pelo demonstrador não é determinística, mas sim probabilística, já que existe uma pequena possibilidade de um demonstrador fraudulento convencer o verificador.

Uma alternativa à propriedade de conhecimento zero é a de *indistinguibilidade de evidência*, introduzida por Feige e Shamir [17]. Em se tratando de protocolos de identificação, uma *evidência* corresponde à chave secreta, para qual sempre existe uma chave pública correspondente que é utilizada por B para averiguar se a prova de identidade apresentada por A é válida. No caso de protocolos de conhecimento zero, portanto, ao final da interação, a única informação que o verificador obtém é a de que a evidência utilizada pelo demonstrador é válida. Já para protocolos que provêm indistinguibilidade de evidência, a única garantia necessária é a de que, havendo duas chaves privadas válidas para uma mesma chave pública, o verificador não consiga determinar qual das duas foi utilizada pelo demonstrador. Entretanto, outras informações podem ser expostas, tais como o conjunto de evidências possíveis. Além da condição mais fraca, uma diferença fundamental de indistinguibilidade de evidência em relação a conhecimento zero é a de que, se presente, a primeira propriedade é preservada quando o protocolo interativo é executado diversas vezes em paralelo, mas a segunda não necessariamente.

3.1.3 Classes de Ataques e Modelos de Segurança

Existem diversas *classes de ataques* a um protocolo de identificação, isto é, de tentativas de burlar o procedimento para que um impostor seja aceito como um demonstrador legítimo. O *modelo de segurança* adotado define as hipóteses admitidas sobre a capacidade do adversário de atacar o protocolo.

Em um *ataque passivo*, o adversário apenas monitora as informações trocadas entre demonstrador e verificador, na tentativa de posteriormente conseguir validar uma prova

de identidade falsa. Ou seja, no *modelo de ataque passivo*, supomos que a capacidade do adversário se limita a monitorar o canal de comunicação.

Já em um *ataque ativo* existe uma etapa inicial em que o adversário interage uma vez com o demonstrador autêntico que deseja personificar, agindo como um suposto verificador. Em seguida, utilizando as informações coletadas nessa primeira fase, o impostor tenta ludibriar o verificador real.

Um *ataque ativo concorrente* é mais forte que um ataque passivo ou um ataque ativo convencional. Neste cenário, o adversário também assume o papel de um verificador desonesto, mas pode interagir com diversos "clones" do demonstrador de forma concorrente, os quais utilizam a mesma chave secreta, mas mantêm estados independentes. Logo após, ocorre a tentativa de se passar por um demonstrador legítimo.

3.1.4 Protocolo de Fiat-Shamir

O protocolo de Fiat-Shamir [19] é um dos mais antigos esquemas de identificação de conhecimento zero propostos. De forma similar ao algoritmo RSA, este protocolo faz uso de um inteiro n formado a partir do produto de dois números primos. A fatoração de n é conhecida apenas por um centro confiável, que encerra suas atividades após publicar o valor de n e distribuir cartões de identificação às entidades. Cada cartão armazena como informação secreta um conjunto de números inteiros $s_1, s_2, ..., s_k$, tais que $1 \leq s_i < n$, para $i = 1, 2, ..., k$. Além disso, ao distribuir os cartões, o centro confiável publica k valores v_i, tais que $v_i^{-1} = s_i^2 \bmod n$. O processo de seleção e divulgação dessas informações corresponde à etapa de geração de chaves, conforme esquematizado abaixo.

FIAT-SHAMIR: Geração de Chaves

---

1. Centro confiável seleciona e publica $n = pq$, mantendo os primos p e q em segredo.
2. Chave privada: $s_1, s_2, ..., s_k \in \mathbb{Z}^m$
3. Chave pública: $v_1, v_2, ..., v_k$, tais que
   $v_i^{-1} = s_i^2 \bmod n$, para $i = 1, 2, ..., k$

A verificação da identidade de A propriamente dita se dá conforme a sequência de passos apresentada na página seguinte. Inicialmente, o demonstrador seleciona aleatori-

amente um inteiro $r \in \{1, 2, ..., n-1\}$, calcula $x = r^2 \bmod n$ e informa o valor de x a B (passos 1 e 2). Em seguida, o verificador envia como desafio um vetor binário aleatório de dimensão k (passo 3), cujos bits são utilizados em conjunto com os valores s_i para compor a resposta de A, dada por $y \leftarrow r \prod_{e_i=1} s_i \mod n$ (passo 4). Caso o demonstrador seja legítimo, o verificador pode atestar no passo 5 que:

$$y^2 \prod_{e_i=1} v_i = r^2 \prod_{e_j=1} s_j^2 \prod_{e_i=1} v_i = r^2 \prod_{e_i=1} s_i^2 \; v_i = r^2 \equiv x \pmod{n}$$

Pode-se notar que ao repetir o procedimento um certo número t de vezes, a probabilidade de se produzir respostas válidas sem conhecer as raízes dos valores v_i^{-1} torna-se muito baixa. Como a dificuldade de extrair estas raízes quadradas, sem conhecer a fatoração de n, equivale à de fatorar n, a segurança do protocolo é assegurada. Isto é, um impostor seria incapaz de responder de forma apropriada os desafios do verificador e se passar por A. Ainda, os autores provam que para um dado k e um t arbitrário, este protocolo é de conhecimento zero.

PROTOCOLO DE IDENTIFICAÇÃO DE FIAT-SHAMIR

Demonstrador A		Verificador B
Repetir t vezes os passos de 1 a 5:		
1. $r \xleftarrow{\$} \{0, 1, ..., n-1\}$		
2. $x \leftarrow r^2 \bmod n$	$\xrightarrow{x}$	
3.	$\xleftarrow[desafio]{\mathbf{e}}$	$\mathbf{e} = (e_1, e_2, ..., e_k) \xleftarrow{\$} \{0,1\}^k$
4. $y \leftarrow r \prod_{e_i=1} s_i \mod n$	$\xrightarrow{y}$	
5.		Verificar se $x \equiv y^2 \prod_{e_i=1} v_i \pmod{n}$

3.2 Esquemas de Assinatura

Assinaturas digitais se assemelham a protocolos de identificação no sentido de também envolverem algo produzido por uma entidade A (desta vez chamada de assinante) sendo examinado por um verificador B. Contudo, no caso de assinaturas, o que se deseja averiguar é se uma dada mensagem foi de fato produzida por A e manteve-se intacta durante a transmissão, e não a identidade de A em si.

Dessa forma, uma das principais diferenças entre protocolos de identificação e de assinatura é que no primeiro, A geralmente fornece as informações a B imediatamente após sua requisição, e estas são prontamente examinadas. Este cenário pode ser observado na prática nos casos de pessoas identificando-se em caixas bancários ou fornecendo senhas para acesso a computadores. Já assinaturas devem ser passíveis de serem verificadas por um tempo indefinidamente longo após sua geração, e até mesmo sem a presença da entidade que as criou.

Além disso, assinaturas são compostas utilizando exclusivamente informações privadas do assinante e dados derivados da mensagem a ser assinada. Isto é, não é incorporado nenhum elemento fornecido por B, de forma que uma mesma assinatura pode ser examinada por mais de um verificador. Em uma analogia com assinaturas em papel, um documento assinado pode ser verificado anos após sua criação, à revelia da pessoa que o assinou, e avaliado por diversos indivíduos.

3.2.1 Conversão entre Protocolos de Identificação e Assinatura

Um protocolo de assinatura pode ser convertido em um protocolo de identificação a partir de uma redução simples, bastando considerar o desafio do verificador como uma mensagem a ser assinada. Dado que o esquema de assinatura assegura que a mensagem (desafio) só poderá ser corretamente assinada por um demonstrador legítimo, se o passo de verificação indicar uma assinatura válida, aceita-se a identidade do demonstrador.

Infelizmente, a conversão contrária não pode ser realizada de maneira tão trivial. Porém, Fiat e Shamir [19] apontam uma forma de converter seu protocolo de identificação no esquema de assinatura digital exibido no quadro a seguir. Este método é particularmente interessante, pois pode ser adaptado para outros protocolos que sigam uma estrutura semelhante de interação entre as duas entidades. Por esse motivo, é conhecido como *Transformação de Fiat-Shamir* ou *Heurística de Fiat-Shamir*.

3.2.2 Transformação de Fiat-Shamir

No esquema de assinatura proposto por Fiat e Shamir, as chaves privada e pública de cada assinante são geradas de maneira idêntica à adotada pelo protocolo de identificação.

FIAT-SHAMIR: Geração de Chaves	
1.	Centro confiável seleciona e publica $n = pq$, mantendo os primos p e q em segredo.
2. Chave privada:	$s_1, s_2, ..., s_k \in \mathbb{Z}^m$
3. Chave pública:	$v_1, v_2, ..., v_k$, tais que $v_i^{-1} = s_i^2 \bmod n$, para $i = 1, 2, ..., k$

Em seguida, conforme esquematizado a seguir, os passos A1 a A4 realizados por A para assinar uma mensagem m são bastante similares aos quatro primeiros passos do protocolo original, quando consideramos as t rodadas. A maior modificação ocorre no passo A3, no qual o vetor enviado por B no desafio do protocolo de identificação é substituído pelo resultado da aplicação de uma função pseudo-aleatória f sobre um vetor formado a partir dos bits da mensagem em questão e dos valores x_i selecionados por A.

Durante a fase de verificação, o cálculo realizado em B1 é o mesmo efetuado no passo 5 do protocolo de identificação, quando consideramos todas as t iterações. Contudo, desta vez o verificador não tem conhecimento suficiente para apurar de forma direta se os resultados z_i correspondem aos valores x_i. Ao invés disso, B aplica a função f sobre $(m, z_1, z_2..., z_t)$ e confronta o valor obtido com a matriz de e_{ij} fornecida por A (passo B2).

Conforme mencionado anteriormente, em geral a sequência de passos que compõe o protocolo de identificação é executada sem grandes intervalos entre um passo e outro. No esquema de Fiat-Shamir apresentado, isso pode ser evidenciado pela predominância de ações intercaladas entre demonstrador e verificador. Já no caso do protocolo de assinatura, observam-se duas fases distintas, a de assinatura propriamente dita, e a de verificação. Do mesmo modo, A não recebe de B nenhuma informação a ser incorporada na assinatura da mensagem m, de forma que a comprovação de sua validade não é restrita a um único verificador.

PROTOCOLO DE ASSINATURA DE FIAT-SHAMIR

Assinante A

Para assinar a mensagem m:

A1. $\mathbf{r} = (r_1, r_2, ..., r_t) \xleftarrow{\$} \{0, 1, ..., n-1\}^t$

A2. Para $i = 1, 2, ..., t$

$$x_i \leftarrow r_i^2 \mod n$$

A3. Calcular $f(m, x_1, x_2..., x_t)$ e usar os primeiros kt bits como valores e_{ij} $(1 \leq i \leq t,\ 1 \leq j \leq k)$

A4. Para $i = 1, 2, ..., t$

$$y_i \leftarrow r_i \prod_{e_{ij}=1} s_j \mod n$$

A5. Enviar m, a matriz de valores e_{ij} e todos os y_i para B

Verificador B

Para verificar a assinatura de A da mensagem m:

B1: Para $i = 1, 2, ..., t$

$$z_i \leftarrow y_i^2 \prod_{e_{ij}=1} v_j \mod n$$

B2: Calcular $f(m, z_1, z_2..., z_t)$ e verificar se os primeiros kt bits correspondem aos valores e_{ij}

3.3 Protocolos de Identificação Baseados em Reticulados

Como mencionado, neste trabalho procurou-se abordar aqueles protocolos de identificação cuja construção tenha sido realizada com base em problemas envolvendo reticulados. A seguir, serão apresentados os principais representantes desta classe.

Os protocolos estão representados pelos nomes de seus respectivos autores, por não terem sido explicitamente nomeados de outra forma nos artigos em que foram apresentados. Para os protocolos propostos por múltiplos autores, utilizou-se o nome do primeiro autor acompanhado da expressão *et alii* abreviada.

3.3.1 Nguyen

Em 2003, Nguyen [52] apresentou um protocolo de identificação que toma por hipótese que não existe um algoritmo eficiente para resolver o GMFP. Dessa forma, podemos tomar como chave privada uma matriz $\mathbf{M} \in \mathbb{Z}^{n \times n}$ tal que $det(\mathbf{M}) \neq 0$, e a matriz de Gram $\mathbf{G} = \mathbf{M}\mathbf{M}^T$ como chave pública. Assim, apenas um demonstrador legítimo conseguiria executar o protocolo para provar sua identidade, visto que de acordo com a hipótese adotada, seria computacionalmente inviável computar $\mathbf{M}$ a partir de $\mathbf{G}$.

NGUYEN: Geração de Chaves	
1. Chave privada:	$\mathbf{M} \xleftarrow{\$} \mathbb{Z}^{n \times n}$, $det(\mathbf{M}) \neq 0$
2. Chave pública:	$\mathbf{G} \leftarrow \mathbf{M}\mathbf{M}^T$

A etapa interativa é realizada em t rodadas. No início de cada uma delas, o demonstrador seleciona aleatoriamente uma matriz $\mathbf{U} \in \mathbb{Z}^{n \times n}$ tal que $det(\mathbf{U}) \neq 0$ (passo 1) e envia a matriz $\mathbf{W} \leftarrow \mathbf{U}\mathbf{G}\mathbf{U}^T$ ao verificador (passo 2). No passo seguinte, o verificador seleciona de maneira aleatória o desafio c e o transmite ao demonstrador. A resposta correspondente, dada por $\mathbf{V} = \mathbf{U}\mathbf{M}^c$, é enviada no passo 4. No último passo, o verificador confirma se $det(\mathbf{V}) \neq 0$ e $\mathbf{V}\mathbf{G}^{1-c}\mathbf{V}^T = \mathbf{W}$. No caso, se $c = 0$, a resposta enviada pelo demonstrador seria $\mathbf{V} = \mathbf{U}$, e portanto $det(\mathbf{V}) = det(\mathbf{U}) \neq 0$ e $\mathbf{V}\mathbf{G}^{1-c}\mathbf{V}^T = \mathbf{U}\mathbf{G}\mathbf{U}^T = \mathbf{W}$.

Por outro lado, se $c = 1$, então $\mathbf{V} = \mathbf{UM}$. Logo, $det(\mathbf{V}) = det(\mathbf{U})\,det(\mathbf{M}) \neq 0$ e $\mathbf{VG}^{1-c}\mathbf{V}^T = \mathbf{VV}^T = \mathbf{UM}(\mathbf{UM})^T = \mathbf{UMM}^T\mathbf{U}^T = \mathbf{UGU}^T = \mathbf{W}$.

NGUYEN: Uma Rodada do Protocolo de Identificação

	Demonstrador		Verificador
1.	$\mathbf{U} \xleftarrow{\$} \mathbb{Z}^{n\times n}$ tal que $det(\mathbf{U}) \neq 0$		
2.	$\mathbf{W} \leftarrow \mathbf{UGU}^T$	$\xrightarrow{\mathbf{W}}$	
3.		$\xleftarrow{c}$	$c \xleftarrow{\$} \{0,1\}$
4.	$\mathbf{V} = \mathbf{UM}^c$	$\xrightarrow{\mathbf{V}}$	
5.			Aceita se $det(\mathbf{V}) \neq 0$ e $\mathbf{VG}^{1-c}\mathbf{V}^T = \mathbf{W}$

O verificador aceita a identidade alegada pelo demostrador se aceitar todas as t rodadas do protocolo, o que sempre acontece no caso de um demonstrador legítimo. Já a probabilidade de um impostor ser aceito é menor que $1/2^t + \epsilon$ e demonstra-se ainda que este protocolo é de conhecimento zero [52].

3.3.2 Lyubashevsky I

Em 2008, Lyubashevsky [42] propôs um esquema de identificação cuja segurança se baseia na dificuldade de se resolver o SIS. Dado um trio de parâmetros n, m e p, onde $m = \lceil 4n \log n \rceil$ e p é um inteiro de ordem $\tilde{\Theta}(n^3)$, o demonstrador seleciona aleatoriamente um vetor binário $\hat{w}$ de dimensão m como chave privada. A chave pública correspondente é composta por uma matriz $\mathbf{A} \xleftarrow{\$} \mathbb{Z}^{n\times m}$, também selecionada aleatoriamente, e por um vetor $\mathbf{w} \leftarrow \mathbf{A\hat{w}} \bmod p$. No caso, a matriz $\mathbf{A}$ pode tanto ser criada pelo demonstrador quanto por uma terceira entidade confiável e, nesta última situação, tem-se a vantagem de que todos os usuários do protocolo podem compartilhar a mesma matriz.

No primeiro passo da fase de interação, o demonstrador seleciona aleatoriamente um vetor $\hat{\mathbf{y}}$ do conjunto $\{0, 1, ..., 5m-1\}^m$ e, no passo 2, transmite ao verificador o vetor $\mathbf{y} \leftarrow \mathbf{A\hat{y}} \bmod p$. O verificador então envia um desafio $c \xleftarrow{\$} \{0,1\}$ (passo 3), que é respondido pelo demonstrador conforme esquematizado nos passos de 4 a 8. Se $c = 0$, a resposta

Lyubashevsky I: Geração de Chaves	
1. Chave privada:	$\hat{\mathbf{w}} \xleftarrow{\$} \{0,1\}^m$
2. Chave pública:	$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_p^{n\times m}$
	$\mathbf{w} \leftarrow \mathbf{A}\hat{\mathbf{w}} \bmod p$

ao desafio é simplesmente $\mathbf{z} = \hat{\mathbf{y}} + c\,\hat{\mathbf{w}} = \hat{\mathbf{y}}$. Senão, é verificado se a quantidade $\hat{\mathbf{y}} + \hat{\mathbf{w}}$ se encontra no conjunto SAFE $= \{1, 2, ..., 5m-1\}^m$ e, em caso positivo, o valor $\mathbf{z} = \hat{\mathbf{y}} + c\,\hat{\mathbf{w}} = \hat{\mathbf{y}} + \hat{\mathbf{w}}$ é enviado como resposta. Em caso negativo, o demonstrador envia $\mathbf{z} \leftarrow \perp$ indicando que se recusa a responder ao desafio. Nesta última situação, a interação é rejeitada imediatamente pelo verificador. Caso contrário, resta ao verificador conferir se a resposta recebida é válida (passo 9). Independentemente do desafio c, temos que o maior valor que um elemento do vetor $\mathbf{z}$ pode assumir é $5m-1$, e portanto, considerando as m coordenadas, $||\mathbf{z}|| \leq \sqrt{m(5m-1)^2} = m^{0.5}(5m-1) = 5m^{1.5} - m^{0.5} \leq 5m^{1.5}$. Além disso, $\mathbf{Az} = \mathbf{A}(\hat{\mathbf{y}} + c\,\hat{\mathbf{w}}) = \mathbf{A}\hat{\mathbf{y}} + \mathbf{A}c\,\hat{\mathbf{w}} = \mathbf{y} + c\,\mathbf{w} \pmod p$. O valor de d indica se a interação foi aceita ($d = 1$) ou recusada ($d = 0$).

A verificação para examinar se $\hat{\mathbf{y}} + \hat{\mathbf{w}}$ encontra-se dentro do conjunto SAFE visa impedir que sejam expostas informações acerca da chave secreta $\hat{\mathbf{w}}$. No caso, como $\hat{\mathbf{w}}$ é binário e as coordenadas de $\hat{\mathbf{y}}$ assumem valores entre 0 e $5m-1$, uma coordenada 0 em $\hat{\mathbf{y}} + \hat{\mathbf{w}}$ só ocorre caso esta coordenada em $\hat{\mathbf{w}}$ também seja 0. Similarmente, uma coordenada $5m$ em $\hat{\mathbf{y}} + \hat{\mathbf{w}}$ revela que a coordenada correspondente em $\hat{\mathbf{w}}$ é 1. Portanto, vetores com essas coordenadas são deixados de fora do conjunto SAFE. Uma alternativa seria o demonstrador sempre selecionar $\hat{\mathbf{y}}$ de modo que não seja possível inferir esse tipo de informação a partir de $\hat{\mathbf{y}} + \hat{\mathbf{w}}$. Contudo, como para $c = 0$ o valor $\hat{\mathbf{y}}$ é revelado, a distribuição dos $\hat{\mathbf{y}}$ utilizados em diversas rodadas do protocolo podem acabar expondo o segredo $\hat{\mathbf{w}}$. Outra possibilidade seria fazer com que m fosse da ordem de $n^{\omega(1)}$, de tal forma que a possibilidade de uma coordenada de $\hat{\mathbf{y}}$ ser 0 ou $5m-1$ fosse muito baixa. Entretanto, um m assim tão grande enfraqueceria o resultado da prova de segurança consideravelmente.

Essa particularidade do protocolo conduz a situações em que mesmo um demonstrador honesto possa ser rejeitado pelo verificador, caso se recuse a responder o desafio. Dessa

LYUBASHEVSKY I: Uma Rodada do Protocolo de Identificação

	Demonstrador		Verificador
1.	$\hat{\mathbf{y}} \xleftarrow{\$} \{0,1,...,5m-1\}^m$		
2.	$\mathbf{y} \leftarrow \mathbf{A}\hat{\mathbf{y}} \bmod p$	$\xrightarrow{\mathbf{y}}$	
3.		$\xleftarrow{c}$	$c \xleftarrow{\$} \{0,1\}$
4.	Se $c = 1$ e $\hat{\mathbf{y}} + \hat{\mathbf{w}} \notin$ SAFE		
5.	$\mathbf{z} \leftarrow \perp$		
6.	Senão		
7.	$\mathbf{z} \leftarrow \hat{\mathbf{y}} + c\,\hat{\mathbf{w}}$		
8.		$\xrightarrow{\mathbf{z}}$	
9.			Se $\|\|\mathbf{z}\|\| \leq 5m^{1.5}$ e $\mathbf{Az} \bmod p = c\,\mathbf{w} + \mathbf{y}$
10.			$d \leftarrow 1$
11.			Senão
12.			$d \leftarrow 0$

forma, é preciso garantir que tal circunstância não ocorra com frequência. Ademais, o protocolo deve ser repetido uma quantidade suficiente de vezes para que seja possível diferenciar este caso da interação com um demonstrador desonesto. No caso, considerando t rodadas, temos o protocolo completo apresentado a seguir.

Dessa forma, temos que a probabilidade de que o verificador aceite uma interação do protocolo com um demonstrador honesto, isto é, que um dado d_i seja 1, é maior ou igual à probabilidade de que $\hat{\mathbf{y}}_i + \hat{\mathbf{w}} \in$ SAFE, pois nesse caso o desafio sempre é respondido, independentemente do valor de c. Para $m \geq 10$, essa probabilidade é de pelo menos 81% e, no total, o protocolo tem erro de completude de $2^{-t/14}$ [42].

PROTOCOLO DE IDENTIFICAÇÃO DE LYUBASHEVSKY I

	Demonstrador		Verificador
1.	Para $i = 1$ até t:		
2.	$\hat{\mathbf{y}}_i \xleftarrow{\$} \{0, 1, ..., 5m-1\}^m$		
3.	$\mathbf{y}_i \leftarrow \mathbf{A}\hat{\mathbf{y}}_i \bmod p$	$\xrightarrow{\mathbf{y}_1, ..., \mathbf{y}_t}$	
4.			Para i=1 até t:
5.		$\xleftarrow{c_1, ..., c_t}$	$c_i \xleftarrow{\$} \{0, 1\}$
6.	Para $i = 1$ até t:		
7.	Se $c_i = 1$ e $\hat{\mathbf{y}}_i + \hat{\mathbf{w}} \notin$ SAFE		
8.	$\mathbf{z}_i \leftarrow \perp$		
9.	Senão		
10.	$\mathbf{z}_i \leftarrow \hat{\mathbf{y}}_i + c_i\hat{\mathbf{w}}$		
11.		$\xrightarrow{\mathbf{z}_1, ..., \mathbf{z}_t}$	
12.			Para $i = 1$ até t:
13.			Se $\|\|\mathbf{z}_i\|\| \leq 5m^{1.5}$ e $\mathbf{A}\mathbf{z}_i \bmod p = c_i\mathbf{w} + \mathbf{y}_i$
14.			$d_i \leftarrow 1$
15.			Senão
16.			$d_i \leftarrow 0$
17.			$soma = d_1 + ... + d_t$
18.			Se $soma \geq 0.65t$ então ACEITA
19.			Senão REJEITA

Conforme mencionado, a segurança do protocolo se baseia na dificuldade de se resolver o SIS, no sentido de que pode-se provar que um adversário que consegue atacar com sucesso o protocolo, consegue resolver o SIS para uma matriz $\mathbf{A}$ qualquer, considerando o modelo de ataques ativos.

O autor discute brevemente a possibilidade de se trabalhar com reticulados ideais, restringindo-se àqueles correspondentes a ideais no anel $\mathbb{Z}[x]/\langle x^n+1\rangle$. Neste caso, a única modificação necessária seria não mais escolher a matriz $\mathbf{A}$ aleatoriamente do conjunto $\mathbb{Z}_p^{n\times m}$, mas selecionar uma matriz na forma da Equação 2.4, com $\mathbf{f} = (1, 0, ..., 0)$. Para essa situação, é fornecido apenas o rascunho da demonstração de segurança, já que esta é muito similar à apresentada para reticulados genéricos.

Por fim, é apontado que infelizmente este protocolo não pode ser utilizado na prática, pois é inseguro para parâmetros aceitáveis para o uso em aplicações reais. Em 2009, Lyubashevsky propôs um novo protocolo [43], aproveitando algumas ideias presentes nesse trabalho anterior, tais como a do demonstrador se recusar a responder ao desafio em algumas interações. Essa proposta mais recente será apresentada na Seção 3.3.4.

3.3.3 Kawachi et al.

Kawachi et al. [38] propõem uma variação do protocolo de identificação de Stern [62], baseado em problemas de teoria de códigos, que seja resistente a ataques concorrentes, assumindo o pior caso de problemas envolvendo reticulados. Além disso, a partir do protocolo proposto, é possível construir um *esquema de identificação ad hoc anônimo*. Este tipo de mecanismo envolve também duas entidades, o demonstrador e o verificador, mas também supõe a existência de um grupo ad hoc. Desse modo, dadas as chaves públicas de todos os membros do grupo, o objetivo é convencer o verificador que o demonstrador pertence ao grupo, sem especificar a sua identidade.

O algoritmo de geração de chaves do protocolo seleciona como chave privada um vetor binário $\mathbf{x}$, cujo peso de Hamming é metade de sua dimensão m. A chave pública é composta por uma matriz $\mathbf{A}$, selecionada aleatoriamente de $\mathbb{Z}_q^{n\times m}$, e de um vetor $\mathbf{y}$, calculado a partir de $\mathbf{x}$ e $\mathbf{A}$.

O processo no qual é realizada a identificação propriamente dita se inicia a partir da seleção aleatória de uma permutação σ (no caso, S_m é o conjunto de todas as permutações de m elementos) e de um vetor $\mathbf{r}$, nos passos 1 e 2, respectivamente. A partir de σ, $\mathbf{r}$, $\mathbf{A}$

KAWACHI ET AL.: Geração de Chaves	
1. Chave privada:	$\mathbf{x} \xleftarrow{\$} \{0,1\}^m$ tal que o peso de Hamming de $\mathbf{x}$ é $m/2$
2. Chave pública:	$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$
	$\mathbf{y} \leftarrow \mathbf{A}\mathbf{x} \bmod q$

e $\mathbf{x}$, são calculados e transmitidos ao verificador três valores de compromisso nos passos de 3 a 5.

Como função de compromisso, os autores sugerem $\text{COM}(s, \rho) = h_{\mathbf{C}}(s) + f_{\mathbf{B}}(\rho)$, onde s possui dimensão m, ρ possui dimensão n, e o resultado da função também corresponde a um vetor de tamanho n. Se a função de compromisso é apenas $\text{COM}(s)$, então utiliza-se simplesmente $\text{COM}(s) = h_{\mathbf{C}}(s)$. Para determinar $h_{\mathbf{C}}(s)$ e $f_{\mathbf{B}}(\rho)$, parte-se da definição de função $f_{\mathbf{Y}}(\mathbf{x}) = \mathbf{Y}\mathbf{x} \bmod q$, de forma que para $f_{\mathbf{B}}(\rho)$, temos simplesmente $f_{\mathbf{B}}(\rho)$=$\mathbf{B}\rho \bmod q$.

Para calcular $h_{\mathbf{C}}(s)$, utiliza-se a construção de Merkle–Damgård a partir de $f_{\mathbf{C}}(.)$:

1. $k = \lceil m/n \rceil$
2. Se necessário, estende-se s com zeros até que sua dimensão corresponda a kn
3. Divide-se s em k vetores s_0, s_1,..., s_k de dimensão n
4. $H_0 \leftarrow 0$
5. Para $i = 0$ até $k + 1$:

   $$H_i \leftarrow f_{\mathbf{C}}(H_{i-1} || s_{i-1})$$

6. $h_{\mathbf{C}}(s) = H_{k+1}$

Dessa forma, embora tome como entrada um vetor de dimensão $m > n$, o resultado de $h_{\mathbf{C}}(s)$ também possui dimensão n.

Após receber os valores de compromisso, o verificador escolhe aleatoriamente o valor do desafio Ch a ser enviado ao demonstrador. Em função deste desafio, o demonstrador revela valores que permitam verificar dois dos três compromissos. Assim, se $Ch = 0$, são

enviados $\sigma(\mathbf{x})$ e $\sigma(\mathbf{r})$ e examina-se c_1 e c_2 (passos 7 e 8). No caso de c_1, a verificação é direta e para c_2, devemos considerar que $\sigma(\mathbf{x}+\mathbf{r}) = \sigma(\mathbf{x}) + \sigma(\mathbf{r})$. Já se $Ch = 1$, são enviados σ e $\mathbf{x}+\mathbf{r}$, e analisados c_0 e c_2 (passos 9 e 10). Para c_0, é preciso levar em conta que $\mathbf{A}(\mathbf{x}+\mathbf{r}) - \mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{A}\mathbf{r} - \mathbf{y} = \mathbf{y} + \mathbf{A}\mathbf{r} - \mathbf{y} = \mathbf{A}\mathbf{r}$. Por fim, se $Ch = 2$, são enviados σ e $\mathbf{r}$, de forma a permitir que c_0 e c_1 sejam verificados.

KAWACHI ET AL.: Protocolo de Identificação

	Demonstrador		Verificador
1.	$\sigma \xleftarrow{\$} S_m$		
2.	$\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^m$		
3.	$c_0 \leftarrow \textsc{Com}(\sigma, \mathbf{A}\mathbf{r})$	$\xrightarrow{c_0}$	
4.	$c_1 \leftarrow \textsc{Com}(\sigma(\mathbf{r}))$	$\xrightarrow{c_1}$	
5.	$c_2 \leftarrow \textsc{Com}(\sigma(\mathbf{x}+\mathbf{r}))$	$\xrightarrow{c_2}$	
6.		$\xleftarrow{Ch}$	$Ch \xleftarrow{\$} \{0,1,2\}$
7.	Se $Ch = 0$, então:		
8.		$\xrightarrow{\sigma(\mathbf{x}), \sigma(\mathbf{r})}$	Verifica se $c_1 = \textsc{Com}(\sigma(\mathbf{r}))$, $c_2 = \textsc{Com}(\sigma(\mathbf{x}) + \sigma(\mathbf{r}))$ e se o peso de Hamming de $\sigma(\mathbf{x})$ é igual a $m/2$
9.	Se $Ch = 1$, então:		
10.		$\xrightarrow{\sigma, \mathbf{x}+\mathbf{r}}$	Verifica se $c_0 = \textsc{Com}(\sigma, \mathbf{A}(\mathbf{x}+\mathbf{r}) - \mathbf{y})$, $c_2 = \textsc{Com}(\sigma(\mathbf{x}+\mathbf{r}))$
11.	Se $Ch = 2$, então:		
12.		$\xrightarrow{\sigma, \mathbf{r}}$	Verifica se $c_0 = \textsc{Com}(\sigma, \mathbf{A}\mathbf{r})$, $c_1 = \textsc{Com}(\sigma(\mathbf{r}))$

Foi demonstrado que este protocolo é de conhecimento zero e que a completude é perfeita. Já o erro de coerência é de no máximo 2/3 para cada execução [38]. Assim, paralelizando cada passo um número $t = \omega(\log n)$ de vezes, este erro se torna desprezível.

3.3.4 Lyubashevsky II

Uma desvantagem dos esquemas de identificação baseados em reticulados anteriores [38, 42, 51] era a de que estes exigiam a transferência de um número muito grande de bits, visto que para cada bit de desafio enviado, a resposta consistia de um certo número $\tilde{O}(n)$ bits. Como nestes casos a segurança está diretamente ligada ao número de bits de desafio enviados pelo verificador, na prática milhões de bits precisavam ser transmitidos. Nos protocolos baseados em problemas de Teoria dos Números, por outro lado, a resposta é maior que o desafio apenas por um pequeno fator. Da mesma forma, os protocolos de assinatura baseados em reticulados anteriores ou eram muito ineficientes ou produziam assinaturas também da ordem de milhões de bits.

O segundo protocolo de identificação proposto por Lyubashevsky [43] tem complexidade de aproximadamente 65.000 bits e produz assinaturas em torno de 50.000 bits, quando convertido em um protocolo de assinatura a partir de uma transformação como a de Fiat-Shamir [19]. Este resultado é atingido explorando a estrutura algébrica limitada de reticulados ideais, de forma que os bits do desafio possam ser tratados coletivamente. Em protocolos anteriores que não envolviam reticulados, isso era feito interpretando a sequência completa do desafio como um inteiro em um determinado domínio, e não simplesmente como um conjunto de zeros e uns. No esquema proposto, a cadeia correspondente ao desafio é vista como um polinômio no anel $\mathbb{Z}[x]/\langle x^n + 1\rangle$, e a segurança do protocolo se baseia na dificuldade em se encontrar o vetor mais curto aproximado por um fator de $\tilde{O}(n^2)$ em reticulados correspondentes a ideais nesse anel. O uso de tais reticulados também permite que o protocolo seja bastante eficiente, com as operações requerendo tempo $\tilde{O}(n)$.

É aproveitada também a ideia de que em algumas situações, o demonstrador pode não responder ao desafio, de forma que alguns passos precisem ser repetidos um certo número de vezes. Nesses casos, o desafio pode ser sempre o mesmo, alterando-se apenas as informações selecionadas aleatoriamente pelo demonstrador e os valores correspondentes enviados ao verificador. Como esses últimos podem ter tamanho bastante reduzido, o número de bits transmitidos é determinado basicamente pelo tamanho da resposta ao desafio.

Converter o protocolo de identificação em um protocolo de assinatura permite ainda mais otimizações, já que não há necessidade de transmitir os casos de falha, os quais correspondem às vezes em que o demonstrador não consegue responder ao desafio no

protocolo de identificação. Embora as falhas custem tempo, o tamanho da assinatura final é o mesmo ocorrendo ou não falha. No caso, a probabilidade de falha é pequena (2/3), de modo que espera-se que o protocolo seja repetido três vezes até se obter uma assinatura com sucesso.

O protocolo trabalha com cinco inteiros: n, m, σ, κ e p, onde n é obrigatoriamente uma potência de 2, κ é tal que $2^{\kappa}\binom{n}{\kappa} > 2^{160}$ e p deve ser um primo aproximadamente igual a $(2\sigma+1)^m.2^{-\frac{128}{n}}$. Além disso, são utilizados os seguintes domínios:

	Definição
R	anel $\mathbb{Z}_p[x]/\langle x^n+1\rangle$
D	$\{g \in R : \|\|g\|\|_\infty \leq mn\sigma\kappa\}$
D_s	$\{g \in R : \|\|g\|\|_\infty \leq \sigma\}$
D_c	$\{g \in R : \|\|g\|\|_1 \leq \kappa\}$
D_y	$\{g \in R : \|\|g\|\|_\infty \leq mn\sigma\kappa\}$
G	$\{g \in R : \|\|g\|\|_\infty \leq mn\sigma\kappa - \sigma\kappa\}$

Assim, a chave privada $\hat{\mathbf{s}}$ é composta de m polinômios (vetores) selecionados aleatoriamente de D_s, e a chave pública é construída da função de hash h e do polinômio $S \leftarrow h(\hat{s})$. A função h é escolhida dentre a família $\mathcal{H}(R, D, m)$, cujas funções mapeiam valores de D^m a R (no caso, $D_s^m \subset D^m$). Uma função em $\mathcal{H}(R, D, m)$ é definida a partir de um dado $\hat{\mathbf{a}} \in R^m$ como sendo $h_{\hat{\mathbf{a}}}(\hat{\mathbf{z}}) = \hat{\mathbf{a}} \,.\, \hat{\mathbf{z}}$. Todos os demonstradores que utilizem o protocolo podem compartilhar a mesma função h, de forma que o passo de seleção desta função pode ser realizado uma única vez.

Lyubashevsky II: Geração de Chaves	
1. Chave privada:	$\hat{\mathbf{s}} \xleftarrow{\$} D_s^m$
2. Chave pública:	$h \xleftarrow{\$} \mathcal{H}(R, D, m)$
	$\mathbf{S} \leftarrow h(\hat{\mathbf{s}})$

Com exceção dos domínios, a estrutura geral deste protocolo é muito semelhante à da primeira proposta de Lyubashevsky. No primeiro passo, é selecionado aleatoriamente um vetor $\hat{\mathbf{y}}$, a partir do qual calcula-se $\mathbf{Y}$, aplicando-se a função h, e cujo valor é enviado

ao verificador (passo 2). No passo 3, é determinado e transmitido o desafio $\mathbf{c}$, o qual, conforme dito anteriormente, não é mais considerado um conjunto de valores binários, mas sim visto como um polinômio no anel R. Dessa forma, utiliza-se $\mathbf{c}$ para calcular $\hat{\mathbf{z}} \leftarrow \hat{\mathbf{s}}\mathbf{c} + \hat{\mathbf{y}}$ (passo 4), cujo resultado é verificado na etapa seguinte para determinar se o demonstrador irá abortar o protocolo ($\hat{\mathbf{z}} \leftarrow \perp$, no passo 6) ou não. O valor de $\hat{\mathbf{z}}$ é enviado no passo 7 e examinado no passo 8. Se o demonstrador é honesto e não abortou o protocolo, pode-se verificar que $\hat{\mathbf{z}} \in G^m$ e $h(\hat{\mathbf{z}}) = h(\hat{\mathbf{s}}\mathbf{c} + \hat{\mathbf{y}}) = \mathbf{S}c + \mathbf{Y}$.

LYUBASHEVSKY II: Protocolo de Identificação

	Demonstrador		Verificador
1.	$\hat{\mathbf{y}} \xleftarrow{\$} D_y^m$		
2.	$\mathbf{Y} \leftarrow h(\hat{\mathbf{y}})$	$\xrightarrow{\mathbf{Y}}$	
3.		$\xleftarrow{\mathbf{c}}$	$\mathbf{c} \xleftarrow{\$} D_c$
4.	$\hat{\mathbf{z}} \leftarrow \hat{\mathbf{s}}\mathbf{c} + \hat{\mathbf{y}}$		
5.	Se $\hat{\mathbf{z}} \notin G^m$então		
6.	$\hat{\mathbf{z}} \leftarrow \perp$		
7.		$\xrightarrow{\hat{\mathbf{z}}}$	
8.			Aceita se, e somente se, $\hat{\mathbf{z}} \in G^m$ e $h(\hat{\mathbf{z}}) = \mathbf{S}c + \mathbf{Y}$

Devido à possibilidade do demonstrador abortar o protocolo, existe um erro de completude de $1-1/e$. Além disso, demonstrou-se que este protocolo provê indistinguibilidade de evidência, e os valores apontados pelo autor para os parâmetros utilizados garantem que o erro de coerência será no máximo 2^{-80} [43].

3.3.5 Cayrel et al.

Cayrel et al. [11] também propõem um esquema de identificação utilizando reticulados baseado no SIS. Em sua fase de geração de chaves, dado um conjunto de parâmetros (m, n, q) previamente selecionados, é utilizado como chave privada um vetor binário $\mathbf{x}$ de dimensão m. A partir de $\mathbf{x}$ e de uma matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ escolhida de forma aleatória, gera-

se o vetor $\mathbf{y} = \mathbf{Ax} \bmod q$, que compõe a chave pública juntamente com $\mathbf{A}$ e uma função de compromisso COM, equivalente a uma função de hash, escolhida de uma família $\mathcal{F}$ de funções apropriadas. Como sugestão, os autores apontam funções como as empregadas no protocolo de Kawachi et al. (Seção 3.3.3).

CAYREL ET AL.: Geração de Chaves		
1.	Chave privada:	$\mathbf{x} \xleftarrow{\$} \{0,1\}^m$ tal que o peso de Hamming de $\mathbf{x}$ é $m/2$
2.	Chave pública:	$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$
		$\mathbf{y} \leftarrow \mathbf{Ax} \bmod q$
		$\text{COM} \xleftarrow{\$} \mathcal{F}$

O processo de identificação inicia-se com a seleção aleatória de diversos elementos por parte do demonstrador (passos 1 e 2). Assim como no protocolo de Kawachi et al., no passo 1, S_m é o conjunto de todas as permutações de m elementos, e define-se P_σ como a matriz de permutação de dimensão $m \times m$ que representa uma permutação $\sigma \in S_m$. Nos passos 3 e 4, dois valores de compromisso, c_0 e c_1, são calculados e enviados ao verificador. Os cálculos tomam como base os valores selecionados nos passos anteriores e, no caso de c_1, ocorre também incorporação de informações sobre a chave secreta $\mathbf{x}$.

No passo 5, o verificador transmite um valor α selecionado aleatoriamente em $\mathbb{Z}_q^*$, que é utilizado pelo demonstrador para determinar β (passo 6). Após receber β, o verificador envia como desafio um único bit b (passo 7).

A resposta ao desafio é revelar σ e $\mathbf{r_0}$, se $b = 0$, ou $\mathbf{P}_\sigma\mathbf{x}$ e $\mathbf{r_1}$, se $b = 1$. No primeiro caso, se os valores informados pelo demonstrador são válidos, o verificador pode confirmar no passo 9 que:

$$\begin{aligned}
\text{COM}(\sigma||\mathbf{A}\mathbf{P}_\sigma^{-1}\beta - \alpha\mathbf{y}, \mathbf{r}_0) &= \text{COM}(\sigma||\mathbf{A}(\mathbf{u} + \alpha\mathbf{x}) - \alpha\mathbf{y}, \mathbf{r}_0) \\
&= \text{COM}(\sigma||\mathbf{Au} + \alpha\mathbf{Ax} - \alpha\mathbf{y}, \mathbf{r}_0) \\
&= \text{COM}(\sigma||\mathbf{Au}, \mathbf{r}_0) \\
&= c_0
\end{aligned}$$

CAYREL ET AL.: Protocolo de Identificação

	Demonstrador		Verificador
1.	$\mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m$, $\sigma \xleftarrow{\$} S_m$		
2.	$\mathbf{r}_0 \xleftarrow{\$} \{0,1\}^n, \mathbf{r}_1 \xleftarrow{\$} \{0,1\}^n$		
3.	$c_0 \leftarrow \text{COM}(\sigma \| \| \mathbf{Au}, \mathbf{r}_0)$	$\xrightarrow{c_0}$	
4.	$c_1 \leftarrow \text{COM}(\mathbf{P}_\sigma \mathbf{u} \| \| \mathbf{P}_\sigma \mathbf{x}, \mathbf{r}_1)$	$\xrightarrow{c_1}$	
5.		$\xleftarrow{\alpha}$	$\alpha \xleftarrow{\$} \mathbb{Z}_q^*$
6.	$\beta \leftarrow \mathbf{P}_\sigma(\mathbf{u} + \alpha \mathbf{x})$	$\xrightarrow{\beta}$	
7.		$\xleftarrow{b}$	$b \xleftarrow{\$} \{0,1\}$
8.	Se $b = 0$, então:		
9.		$\xrightarrow{\sigma, \mathbf{r}_0}$	Verifica se $c_0 = \text{COM}(\sigma \| \| \mathbf{AP}_\sigma^{-1} \beta - \alpha \mathbf{y}, r_0)$
10.	Senão:		
11.		$\xrightarrow{\mathbf{P}_\sigma \mathbf{x}, \mathbf{r}_1}$	Verifica se $c_1 = \text{COM}(\beta - \alpha \mathbf{P}_\sigma \mathbf{x} \| \| \mathbf{P}_\sigma \mathbf{x}, \mathbf{r_1})$

Já para $b = 1$, o verificador pode apurar no passo 11 que:

$$\begin{aligned}
\text{COM}(\beta - \alpha \mathbf{P}_\sigma \mathbf{x} \| \| \mathbf{P}_\sigma \mathbf{x}, \mathbf{r_1}) &= \text{COM}(\mathbf{P}_\sigma(\mathbf{u} + \alpha \mathbf{x}) - \alpha \mathbf{P}_\sigma \mathbf{x} \| \| \mathbf{P}_\sigma \mathbf{x}, \mathbf{r_1}) \\
&= \text{COM}(\mathbf{P}_\sigma \mathbf{u} + \mathbf{P}_\sigma \alpha \mathbf{x} - \alpha \mathbf{P}_\sigma \mathbf{x} \| \| \mathbf{P}_\sigma \mathbf{x}, \mathbf{r_1}) \\
&= \text{COM}(\mathbf{P}_\sigma \mathbf{u} + \alpha \mathbf{P}_\sigma \mathbf{x} - \alpha \mathbf{P}_\sigma \mathbf{x} \| \| \mathbf{P}_\sigma \mathbf{x}, \mathbf{r_1}) \\
&= \text{COM}(\mathbf{P}_\sigma \mathbf{u} \| \| \mathbf{P}_\sigma \mathbf{x}, \mathbf{r_1}) \\
&= c_1
\end{aligned}$$

Em uma execução, pode-se demonstrar que a probabilidade do verificador aceitar a prova de identidade de um impostor é de 1/2, de forma que o procedimento deve ser executado diversas vezes até se obter o grau de confiança desejado.

3.3.6 Rückert

Rückert [58] propõe um *esquema de identificação baseado em identidade*. Neste caso, a chave pública é substituída por um identificador único, como por exemplo, um endereço de e-mail, e uma entidade confiável obtém a chave privada correspondente a esse identificador, utilizando o que se chama de *algoritmo de extração*. O esquema apresentado é uma variação do proposto por Lyubashevsky [43], utilizando o conceito de *árvores bonsai* para efetuar a extração da chave privada.

Em arboricultura, um bonsai é uma árvore cultivada pelo homem de forma a se tornar uma réplica em miniatura de uma árvore natural, simulando sua estrutura e padrão de crescimento, porém em escala reduzida. A técnica de produzir bonsais envolve combinar um crescimento não direcionado, ou não controlado (isto é, um crescimento natural), com um crescimento direcionado, ou controlado, obtido principalmente através de podas. Porém, um observador é incapaz distinguir o crescimento direcionado do não controlado.

No contexto de reticulados, uma árvore bonsai é composta por vetores: uma raiz $\hat{\mathbf{a}}^*$ e ramos $\hat{\mathbf{b}}_i^{(b)}$, $b \in \{0,1\}$. A raiz representa porção natural, aleatória, e define um reticulado $L_R^{\perp}(\hat{\mathbf{a}}^*) = \{x \in \mathbb{Z}^m : x \,.\, \hat{\mathbf{a}}^* \equiv 0 \bmod p\}$, correspondente a um ideal no anel R. Além disso, pode-se provar que existe um algoritmo representando o crescimento direcionado, que obtém um conjunto de ramos $\langle \hat{b} \rangle = \left\{ (\hat{b}_1^{(0)}, \hat{b}_1^{(1)}), ..., (\hat{b}_\lambda^{(0)}, \hat{b}_\lambda^{(1)}) \right\}$ e uma base $\mathbf{S}^*$ de $L_R^{\perp}(\hat{\mathbf{a}}^*)$.

Assim como no protocolo de Lyubashevsky, é definida uma série de parâmetros a serem utilizados, como os inteiros n, m_1, m_2, m, p, e λ, e os domínios D, D_s, D_c, D_y e G. Neste protocolo, R se mantém como o anel $\mathbb{Z}_p[x]/\langle x^n + 1 \rangle$, com p primo.

O conjunto composto por $\hat{\mathbf{a}}^* \in R^{m_1+m_2}$, $\langle \hat{b} \rangle$ e $S \xleftarrow{\$} R$ corresponde a uma chave pública master, com a base $\mathbf{S}^*$ sendo a chave privada correspondente. Esses valores são utilizados no algoritmo de extração de chave para obter uma chave privada $\hat{s}_{ID}$ para um identificador $ID \in \{0,1\}^\lambda$. O cálculo de $\hat{s}_{ID}$ é realizado a partir do uso de uma função pertencente a uma família de funções apresentada por Gentry et al. [22] e depende da base $\mathbf{S}^*$, de forma que somente a entidade que tem conhecimento de $\mathbf{S}^*$ consegue realizar a extração da chave. Além disso, considerando uma identidade ID como sendo composta pelos bits $ID_1, ..., ID_\lambda$, então cada identidade define um caminho único na árvore, dado por $\hat{\mathbf{a}}_{ID} = \hat{\mathbf{a}}^* || \hat{b}_1^{(ID_1)} || ... || \hat{b}_\lambda^{(ID_\lambda)}$. Isto é, um caminho partindo da raiz $\hat{\mathbf{a}}^*$ e seguindo os ramos $\hat{b}_i^{(b)}$, $i = 1, ..., \lambda$ e $b \in \{0,1\}$. No caso, $\hat{\mathbf{a}}_{ID}$ é utilizado para definir a função $h_{\hat{\mathbf{a}}_{ID}}(\hat{\mathbf{z}}) = \hat{\mathbf{a}}_{ID} \,.\, \hat{\mathbf{z}}$. Dessa forma, a função $h_{\hat{\mathbf{a}}_{ID}}$ utilizada por cada demonstrador depende

de sua identidade.

A estrutura geral do protocolo é idêntica à do de Lyubashevsky, conforme exibido a seguir.

RÜCKERT: Protocolo de Identificação

	Demonstrador		Verificador
1.	$\hat{\mathbf{y}} \xleftarrow{\$} D_y^m$		
2.	$\mathbf{Y} \leftarrow h_{\hat{\mathbf{a}}_{ID}}(\hat{\mathbf{y}})$	$\xrightarrow{\mathbf{Y}}$	
3.		$\xleftarrow{\mathbf{c}}$	$\mathbf{c} \xleftarrow{\$} D_c$
4.	$\hat{\mathbf{z}} \leftarrow \hat{\mathbf{s}}_{ID}\mathbf{c} + \hat{\mathbf{y}}$		
5.	Se $\hat{\mathbf{z}} \notin G^m$ então		
6.	$\hat{\mathbf{z}} \leftarrow \perp$		
7.		$\xrightarrow{\hat{\mathbf{z}}}$	
8.			Aceita se, e somente se, $\hat{\mathbf{z}} \in G^m$ e $h_{\hat{\mathbf{a}}_{ID}}(\hat{\mathbf{z}}) = Sc + \mathbf{Y}$

Como neste protocolo existe a possibilidade do demonstrador se recusar a responder no passo 6, há um erro de completude dado por $1 - 1/e^{\phi}$, onde $\phi \geq 1$ é um inteiro utilizado na determinação de D_y [58].

3.3.7 Síntese dos Protocolos

Observando os protocolos apresentados nesta seção, notamos que o problema mais considerado é o SIS (conforme Tabela 3.2), provavelmente devido à sua maior facilidade de ser incorporado no protocolo, o que é feito simplesmente a partir da multiplicação de uma matriz por um vetor mantido em segredo. Em termos do problema no qual se baseia, o protocolo de Nguyen é o mais particular, embora tome como base um problema cuja segurança ainda permanece como uma questão em aberto.

Apenas os protocolos de Lyubashevsky e de Rückert apresentam erros de completude, devido à possibilidade do demonstrador se recusar a responder o desafio e abortar o protocolo. Contudo, este erro passa a ser desprezível quando se aumenta a quantidade de

Protocolo	Problema	Erro de Completude	Erro de Coerência	Complexidade de Tempo	Outros
Nguyen	GMFP	0	$2^{-t}+\epsilon$	$O(n^{2.376})$	Conhecimento zero
Lyubashevsky I	SIS	$2^{-t/14}$	?	$O(n^2)$	Indistinguibilidade de evidência
Kawachi et al.	GapSVP / SIS	0	$(2/3)^t$	$O(n^2)$	Conhecimento zero
Lyubashevsky II	SVP	$1-1/e$	2^{-80}	$O(n\log n)$	Indistinguibilidade de evidência
Cayrel et al.	SIS	0	2^{-t}	$O(n^2)$	Conhecimento zero
Rückert	SVP	$1-1/e^{\phi}$	?	$O(n\log n)$	Indistinguibilidade de evidência

Tabela 3.2: Síntese dos protocolos de identificação.

rodadas realizadas. Nos casos do segundo protocolo de Lyubashevsky e do protocolo de Rückert, o erro de completude apresentado na tabela diz respeito a uma única execução, já que não é especificada a fração das respostas enviadas pelo demonstrador que deveriam ser aceitas para que sua identidade seja reconhecida (como no caso do primeiro protocolo de Lyubashevsky, onde esse valor é de 65%, conforme o passo 17).

Os demais protocolos exigem um certo número de execuções para reduzir o erro de coerência. Para o protocolo de Nguyen e Cayrel et al., este erro está apresentado em função do número t de rodadas. No protocolo de Kawachi et al., o erro de coerência é de no máximo $2/3$ para cada execução. Assim, se consideramos que para o demonstrador ser aceito, nenhuma interação possa ser recusada (já que não há erro de completude), então para t rodadas, o erro de coerência seria de $(2/3)^t$.

Em geral, não é apresentada muita discussão acerca da complexidade de tempo das operações envolvidas nos protocolos. Nguyen aponta que o uso de matrizes circulantes reduz o custo da multiplicação de matrizes para n^2. Na tabela, considerou-se o caso de matrizes genéricas, porém supondo que a multiplicação de matrizes $n\times n$ pode se realizada em tempo $O(n^{2.376})$, aplicando o algoritmo de Coppersmith e Winograd [14]. Os passos mais caros dos protocolos de Lyubashevsky I, Kawachi et al. e Cayrel et al. consistem na

multiplicação de matrizes por vetores, que pode ser feita com $O(n^2)$ operações quando a matriz não possui nenhuma estrutura específica. O uso de reticulados ideais pode fazer com que esse custo seja reduzido para $O(n \log n)$, como no caso do segundo protocolo de Lyubashevsky e do protocolo de Rückert. Nesses últimos, a aplicação de reticulados ideais é explícita, enquanto que nos três primeiros protocolos mencionados, não, embora os autores comentem sobre essa possibilidade. Por esse motivo, a complexidade de tempo apresentada na Tabela 3.2 considera o caso mais geral em que os reticulados não são necessariamente ideais.

Capítulo 4

Experimento Prático de Implementação

Dada a carência de resultados experimentais acerca da classe de protocolos abordados, considerou-se que seria proveitoso realizar um experimento prático em que alguns protocolos fossem implementados e dados reais de execução pudessem ser coletados. Desta maneira, seria possível, principalmente, averiguar em que grau as previsões assintóticas de desempenho providas pelos autores são confirmadas. Além disso, visto que tais protocolos são elaborados justamente com o objetivo de em algum momento serem utilizados na prática, caso se mostrem adequados, é necessária alguma validação experimental antes de se considerar a sua aplicação de fato. Embora este trabalho não tenha o objetivo de prover este tipo de validação, o desenvolvimento e análise dessas primeiras implementações visa dar início a esse processo.

Dos protocolos analisados, procurou-se determinar aqueles cujo experimento de implementação traria os resultados mais interessantes ou proveitosos. Dessa forma, três foram selecionados: o protocolo de Kawachi et al. [38], o segundo protocolo de Lyubashevsky [43] e o protocolo de Cayrel et al [11]. No caso, o protocolo de Nguyen [52] não foi selecionado por se basear em um problema cuja dificuldade ainda se encontra em aberto, e portanto ser menos promissor que os demais. O primeiro protocolo de Lyubashevsky [42] foi descartado, pois o próprio autor aponta que os parâmetros de segurança precisariam ser grandes demais, inviabilizando seu uso na prática. Já em relação ao protocolo de Rückert [58], por este ser baseado no segundo protocolo de Lyubashevsky, considerou-se

que os resultados seriam bastante semelhantes aos daquele.

Neste capítulo, iremos apresentar o conjunto de implementações realizadas para os protocolos selecionados, juntamente com os dados experimentais coletados a partir da execução dos programas elaborados. Serão comentadas também algumas das dificuldades encontradas durante o desenvolvimento, ou questões sobre as quais foi necessária alguma reflexão adicional.

4.1 Plataformas Utilizadas

Para implementar os protocolos selecionados, foram utilizadas duas linguagens de programação, uma biblioteca externa e algumas ferramentas. A seguir, será apresentado o conjunto desses elementos empregados, bem como dos equipamentos utilizados para teste dos programas desenvolvidos.

4.1.1 Linguagens e Bibliotecas Utilizadas

As implementações realizadas podem ser divididas em duas categorias, de acordo com a linguagem e bibliotecas utilizadas. A primeira delas corresponde às implementações efetuadas em linguagem de programação C, sem auxílio de bibliotecas adicionais à biblioteca padrão do C (libc). Neste caso, todas as operações envolvendo matrizes e vetores tiveram que ser desenvolvidas separadamente.

Já a segunda classe de programas faz uso da biblioteca NTL [54], que provê uma série de estruturas e funções para manipulação de inteiros de tamanho arbitrário, vetores, matrizes, polinômios, entre outros. Dessa forma, as operações com matrizes e vetores ou polinômios (no caso de reticulados ideais) podem ser facilmente realizadas a partir da chamada dos métodos das classes correspondentes. Além disso, como se trata de uma biblioteca desenvolvida em C++, as implementações que a utilizam também foram realizadas nessa linguagem. No caso, foi utilizada a versão 5.5.2 da NTL.

4.1.2 Ferramentas Utilizadas

4.1.2.1 Compilador GCC

Para compilar os programas desenvolvidos, foram utilizados os compiladores gcc, para a linguagem C, e g++, para a linguagem C++, ambos presentes no conjunto GNU Compiler

Collection (GCC) [21]. Durante a fase de desenvolvimento, foram utilizados os parâmetros -W, para informar possíveis *warnings* identificados pelo compilador, -lm, para uso das rotinas de operações matemáticas, e -pedantic e -ansi, para verificar se havia alguma violação aos padrões ISO C e ISO C++. Já na fase de testes, foi acionada a otimização realizada pelo compilador, a partir do parâmetro -O3. Finalmente, para o uso de determinadas ferramentas, foram aplicados alguns parâmetros adicionais, conforme apresentado nas subseções a seguir. A versão do GCC utilizada, tanto para a etapa de implementação quanto de testes, foi a 4.5.3.

4.1.2.2 OpenMP

Uma das maneiras de melhorar o desempenho da execução dos protocolos selecionados é tirar proveito da possibilidade de se efetuar diversas operações em paralelo. Para tanto, foi utilizado o OpenMP (do inglês *Open Multi-Processing*) [55], uma interface de programação de aplicativo (API) que facilita a programação com multiprocessamento em sistemas de memória compartilhada. O OpenMP está disponível para diversas arquiteturas, incluindo as plataformas Linux e Windows, e aceita tanto a linguagem C quanto C++, desde que exista suporte do compilador. O GCC possui uma implementação do OpenMP desde a sua versão 4.2 [26], a qual pode ser utilizada a partir da compilação com o parâmetro -fopenmp.

A paralelização do OpenMP baseia-se no modelo de programação *fork-join*, na qual algumas regiões do código podem ser executadas de maneira paralela, enquanto que outras ficam a cargo de uma única *thread*, na qual todas as operações são realizadas sequencialmente. A determinação dessas regiões fica a cargo do programador, que possui controle total sobre a paralelização, visto que esta não se dá de maneira automática, como ocorre com algumas outras ferramentas para multiprocessamento.

A execução do programa sempre se inicia com uma única *thread*, denominada *thread master*, na qual as operações são realizadas de modo serial até que seja encontrada a primeira região assinalada como paralela. Neste ponto, ocorre o *fork*, isto é, cria-se um conjunto de *threads* entre as quais as operações da região paralelizada podem ser corretamente distribuídas e executadas de maneira simultânea. Quando é atingido o fim da região paralela, as diversas *threads* efetuam um procedimento de sincronização, transparente para o programador, e finalizam sua execução, de modo que reste apenas a *thread*

master novamente. Este conjunto de ações é denominado *join*.

O mecanismo *fork-join* pode se repetir diversas vezes, em função das inúmeras regiões paralelas que o programa pode possuir. A Figura 4.1 apresenta um exemplo desse procedimento, com quatro *threads*.

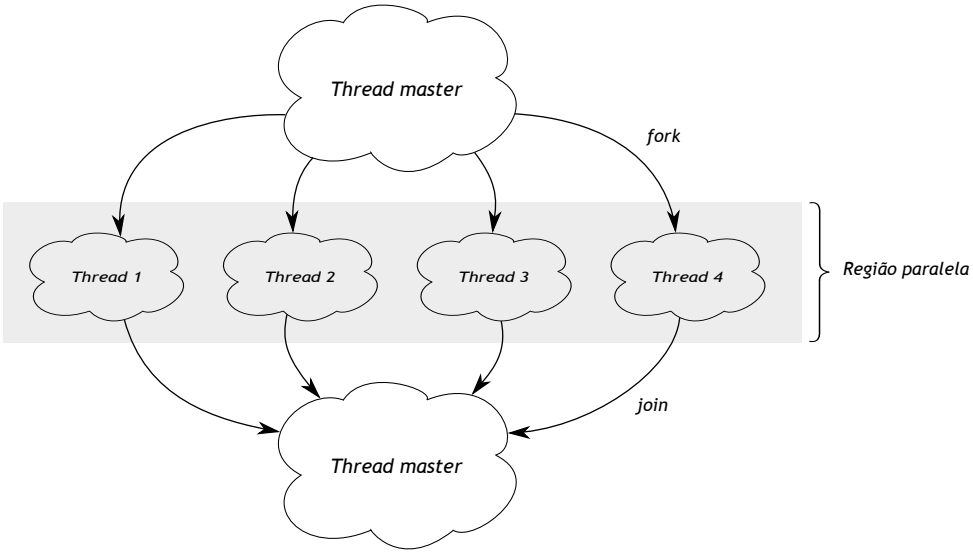


Figura 4.1: Modelo de programação *fork-join* utilizado pelo OpenMP, com quatro *threads*.

A distribuição das porções do programa entre as diversas *threads* depende da construção realizada pelo programador. O primeiro passo consiste em utilizar diretivas do compilador para delimitar a região paralela, isto é, a porção do código a ser executada por múltiplas *threads*. No caso do GCC, utiliza-se a diretiva #pragma. Dessa forma, caso o código-fonte não seja compilado utilizando o OpenMP, o programa ainda pode ser executado normalmente, de maneira serial.

Dentro da região paralela, o programador pode utilizar diretivas específicas do OpenMP para determinar a distribuição das operações entre as *threads*. Basicamente, podemos identificar dois tipos de construção, a *paralelização via laços* e a *paralelização via seções*. No primeiro caso, certos laços do programa, sinalizados explicitamente pelo programador, têm suas iterações distribuídas entre as várias *threads*. Naturalmente, esse tipo de construção só é possível quando uma iteração do laço não depende das iterações anteriores, isto é, não é preciso aguardar o resultado de outras iterações, o que impediria a execução simultânea. Para construções dessa forma, diz-se que há um *paralelismo de dados*, já

que temos uma grande quantidade de dados sendo processada da mesma forma. Ou seja, todas as *threads* executam o mesmo código (as iterações do laço) porém com dados diferentes (Figura 4.2). O escalonamento das iterações entre as *threads* pode ficar totalmente a cargo do compilador e/ou do sistema de *runtime*, ou ser especificada pelo programador.

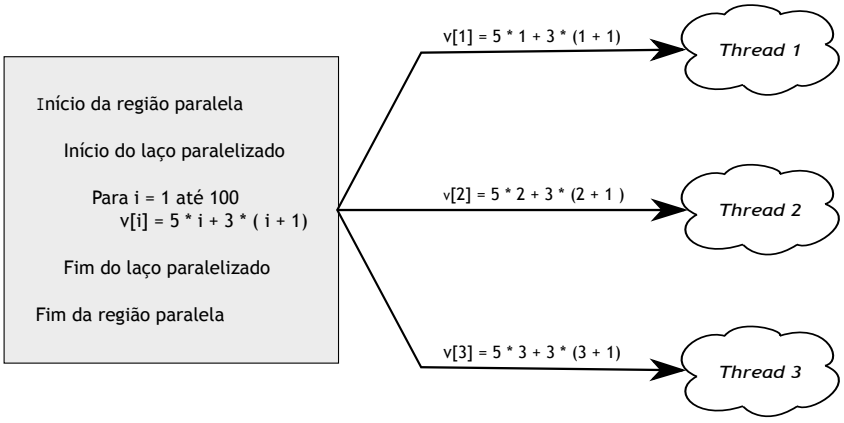


Figura 4.2: Exemplo simples de paralelização com distribuição das iterações do laço entre as *threads*.

Na segunda forma de construção, o programador divide a região paralela em seções e cada seção é executada por uma *thread*. Neste caso, temos um *paralelismo funcional*, já que os blocos funcionais associados a cada *thread* são diferentes. Cada seção é executada apenas uma vez por exatamente uma *thread*. Porém, havendo mais seções do que o número de *threads*, uma mesma *thread* pode executar mais de uma seção (Figura 4.3).

O OpenMP apresenta a facilidade de detectar automaticamente a quantidade de processadores (ou núcleos) disponíveis, de forma que um mesmo programa possa ser executado em diversos equipamentos com diferentes números de processadores. Além disso, se desejado, o número de *threads* criadas para cada região paralela pode ser controlado dinamicamente, a partir de chamadas especiais dentro do programa, ou por uma variável de ambiente específica. Uma outra vantagem do OpenMP é oferecer suporte a paralelismo aninhado.

4.1.2.3 gprof

Para analisar o comportamento durante a execução dos programas desenvolvidos, fez-se uso do gprof, uma ferramenta de análise dinâmica integrante do projeto GNU [28]. O

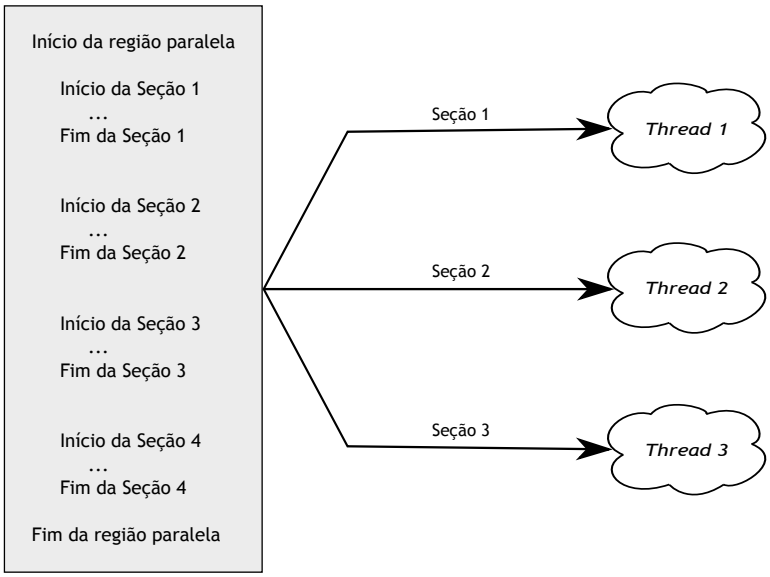


Figura 4.3: Exemplo de paralelização por seções. A Seção 4 poderá ser escalonada para a *thread* que concluir mais rapidamente a execução da primeira seção designada.

gprof coleta informações sobre a execução das funções presentes no programa analisado e as apresenta em conjunto com alguns dados estatísticos. Dessa forma, é possível determinar as porções onde o programa despende mais tempo, o número de chamadas de cada função, quais funções chamam outras funções, entre outros. Essas informações podem ser utilizadas tanto para determinar possíveis pontos de otimização quanto, no nosso caso, averiguar se certas operações estão se comportando conforme o esperado.

Para utilizar o gprof, simplesmente compilou-se o programa a ser analisado com o parâmetro -pg, para indicar que as informações desejadas deveriam ser geradas durante a execução. Em seguida, realizou-se a execução propriamente dita, de forma que os dados pudessem ser coletados. Por fim, invocou-se o gprof para organizar e analisar essas informações.

4.1.3 Equipamentos Utilizados

Todos os dados experimentais aqui apresentados se referem à execução das implementações desenvolvidas em um computador de mesa com processador Intel Core i5 (Quad-Core) @ 2.8 GHz e memória Kingston 4GB HiperX C9 1600 MHz, utilizando sistema operacional Linux Ubuntu 11.04 (natty), kernel Linux 2.6.38-8-generic-pae.

4.2 Implementações Realizadas

Conforme mencionado na Seção 4.1.1, as implementações realizadas para cada protocolo podem ser divididas em duas categorias, de acordo com a linguagem utilizada (C ou C++). Além disso, as implementações em C++ utilizam a biblioteca NTL, enquanto que as implementações em C fazem uso apenas das bibliotecas convencionais do C.

Uma outra forma de classificar os programas desenvolvidos diz respeito à classe de reticulados utilizados. Assim, algumas implementações consideraram reticulados genéricos, enquanto que outras apenas permitem o uso de reticulados ideais. Para essas últimas, as multiplicações de matrizes por vetores, realizadas a partir da multiplicação de polinômios, foram efetuadas a partir do algoritmo iterativo clássico da FFT [15], a princípio sem otimizações, no caso dos programas em C. As implementações que se apoiam na NTL, isto é, as implementações em C++, utilizaram as funções desta biblioteca para multiplicação de polinômios.

Por fim, podemos repartir as implementações segundo esforços de paralelização. Assim, de um lado temos aquelas que não permitem execução simultânea de operações, e do outro aquelas cujo código foi paralelizado com auxílio do OpenMP. Esse último grupo pode ainda ser dividido de acordo com o tipo de construção adotada, isto é, paralelização via laços (paralelismo de dados) ou via seções (paralelismo funcional).

A Tabela 4.1 apresenta um resumo das implementações realizadas, informando para cada uma delas o protocolo implementado , a linguagem empregada (C ou C++), as bibliotecas adicionais utilizadas (no caso, apenas a NTL), os reticulados considerados (genéricos ou ideais) e o tipo de paralelismo implementado, quando presente.

A numeração e disposição na qual as implementações aparecem na Tabela 4.1 de certa forma refletem a ordem cronológica de desenvolvimento. Inicialmente, trabalhou-se com o protocolo de Kawachi et al. em sua forma mais geral, isto é, aceitando qualquer classe de reticulados (implementações 1 e 2). Para efeito de comparação, uma das implementações foi realizada em C++, fazendo uso da NTL, e a outra em C. Neste último caso, a operação de multiplicação de matrizes por vetores foi desenvolvida utilizando o algoritmo clássico para essa finalidade.

Em seguida, procurou-se averiguar os efeitos de tentativas de paralelização sobre essas implementações. Assim, a implementação 3 corresponde à primeira implementação, porém com paralelismo funcional, adicionado com auxílio do OpenMP. Da mesma forma,

	Protocolo	Linguagem	Bibliotecas	Reticulados	Paralelismo
1.	*Kawachi et al.*	C++	NTL	Genéricos	-
2.	*Kawachi et al.*	C	-	Genéricos	-
3.	*Kawachi et al.*	C++	NTL	Genéricos	Funcional
4.	*Kawachi et al.*	C	-	Genéricos	Dados
5.	*Kawachi et al.*	C	-	Genéricos	Funcional
6.	*Kawachi et al.*	C++	NTL	Ideais	-
7.	*Kawachi et al.*	C	-	Ideais	-
8.	*Kawachi et al.*	C	-	Ideais	Dados/funcional
9.	*Cayrel et al.*	C++	NTL	Genéricos	-
10.	*Cayrel et al.*	C++	NTL	Genéricos	Funcional
11.	*Cayrel et al.*	C++	NTL	Ideais	-
12.	*Lyubashevsky II*	C++	NTL	Ideais	-

Tabela 4.1: Implementações realizadas.

as implementações 4 e 5 são resultado da paralelização da implementação 2, via laços (paralelismo de dados) e via seções (paralelismo funcional), respectivamente. No caso da implementação 4, a paralelização se deu nas iterações do laço principal do algoritmo de multiplicação de uma matriz por vetor. Nas implementações em C++, essa operação encontra-se encapsulada em um método da biblioteca NTL, de forma que não é possível utilizar uma estratégia de paralelização equivalente. Entretanto, sabemos que no protocolo de Kawachi et al. o demonstrador e o verificador computam, respectivamente, três e dois valores de compromisso por rodada. Dessa forma, é possível separar estes cálculos

em seções de uma região paralela definida para o OpenMP. Como resultado, as implementações 3 e 5 apresentam paralelismo funcional.

Finalmente, foram desenvolvidas versões do protocolo de Kawachi et al. que trabalhassem apenas com reticulados ideais, tanto em C++ (implementação 6) quanto em C (implementação 7). Para essa última, foram definidas regiões paralelas para cálculo dos valores de compromisso, e também para realização de algumas operações da FFT, tendo como resultado um misto de paralelismo funcional e dados, presentes na implementação 9. A princípio, considerou-se também paralelizar a implementação 6. Porém, esta tarefa mostrou-se mais complexa e trabalhosa do que nos casos anteriores (conforme apresentado na Seção 4.3.4), de forma que se julgou mais proveitoso iniciar o desenvolvimento dos outros protocolos selecionados. Por esse mesmo motivo, as implementações com reticulados ideais dos demais protocolos, no momento, também não possuem uma versão paralelizada.

Terminado o desenvolvimento do protocolo de Kawachi et al., iniciou-se a implementação dos outros dois protocolos selecionados. A partir deste ponto, todos os programas foram elaborados utilizando a biblioteca NTL, principalmente devido à facilidade de se trabalhar com reticulados ideais, decorrente das funções para manipulação de polinômios providas pela biblioteca. Para o protocolo de Cayrel et al., foram desenvolvidas tanto versões empregando qualquer classe de reticulados (implementações 9 e 10), quanto somente reticulados ideais (implementação 11). Além disso, a implementação 10 corresponde à implementação 9 com paralelismo funcional. No caso, foi utilizada a mesma estratégia de separar os cálculos dos valores de compromisso em seções da região paralela, juntamente com a seleção dos valores aleatórios $\mathbf{u}, \sigma, \mathbf{r}_0$ e $\mathbf{r}_1$. Contudo, no caso deste protocolo, somente o demonstrador computa mais de um valor de compromisso. Além disso, mesmo neste caso, há somente o cálculo de dois valores (contra três do protocolo de Kawachi et al.), de forma que já são esperados ganhos menores com essa forma de paralelização. Por fim, foi desenvolvida uma única versão para o protocolo de Lyubashevsky II (implementação 12), visto que o mesmo sempre trabalha com reticulados ideais.

4.3 Dificuldades Encontradas

Durante a fase de implementação, deparou-se com algumas questões a princípio não tratadas ou pouco comentadas nos artigos originais que apresentam os protocolos. Além

disso, algumas etapas do desenvolvimento apresentaram dificuldades inerentes do próprio processo de programação e testes. As subseções a seguir visam apresentar brevemente as questões mais relevantes levantadas e quais as decisões tomadas para solucionar ou contornar as dificuldades que se encontrou.

4.3.1 Seleção Aleatória de Elementos

A primeira dificuldade encontrada diz respeito a seleção aleatória de alguns elementos, os quais devem estar restritos a um determinado domínio. Por exemplo, os protocolos de Kawachi et al. e Cayrel et al. utilizam uma matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ como parte integrante da chave pública. Essa matriz relaciona-se diretamente ao problema no qual a segurança do protocolo se baseia, sendo utilizada para definir o reticulado empregado. Embora não seja mencionado explicitamente, para garantir que a dimensão de $\mathbf{A}$ reflita a dimensão do reticulado, a matriz em questão deve possuir posto completo. Assim, não basta apenas selecionar os seus elementos aleatoriamente, é preciso assegurar que a condição de posto completo seja satisfeita. Da mesma forma, Lyubashevsky define uma série de domínios para os elementos selecionados ao longo do protocolo. Basicamente esses domínios correspondem ao anel $\mathbb{Z}_p[x]/\langle x^n + 1\rangle$ acrescido de uma restrição sobre a norma de seus elementos.

Portanto, ao implementar os protocolos, uma questão a ser considerada é como selecionar aleatoriamente elementos de um determinado domínio. Uma abordagem é considerar um domínio mais abrangente, isto é, com menos restrições, com o qual seja mais fácil trabalhar. Por exemplo o domínio $\mathbb{Z}_q^{n \times m}$, com matrizes de posto qualquer, e todos os elementos do anel $\mathbb{Z}_p[x]/\langle x^n + 1\rangle$, independente da norma. Em seguida, verifica-se se o elemento sorteado atende as condições do domínio mais restrito. Em caso negativo, o processo de seleção pode ser repetido até que seja obtido um elemento que satisfaça todas as restrições. Com esta abordagem, surgem duas novas questões: como verificar se o valor sorteado atende às condições do domínio e qual a probabilidade de isso acontecer.

No caso do protocolo de Lyubashevsky II, determinar a norma dos vetores gerados aleatoriamente, seja ela a norma de Manhattan ou a norma uniforme, é uma tarefa simples. Para as matrizes $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, podemos verificar se o posto da matriz a partir do método de eliminação de Gauss ou aplicando o algoritmo LLL. Em ambos os casos, os custos da verificação são muito superiores ao de simplesmente averiguar uma norma (o método de

eliminação de Gauss tem complexidade de operações de $O(n^3)$ [16]). Por esse motivo, considerou-se que a probabilidade da matriz sorteada não pertencer ao domínio desejado é suficiente pequena[1] e que as matrizes de posto incompleto geralmente resultarão em reticulados com dimensão não muito inferior à desejada.

4.3.2 Seleção de Parâmetros

Usualmente, descreve-se os algoritmos de geração de chaves e do protocolo iterativo utilizando elementos de tamanho abstrato, como as dimensões m e n. Para realização dos testes de implementação, entretanto, é indispensável o uso de valores concretos para esses parâmetros. Tais valores relacionam-se com a segurança provida pelo protocolo, ao determinar os erros de coerência e completude e, principalmente, definir o grau de dificuldade de se resolver o problema no qual o protocolo se baseia.

Neste trabalho, partiu-se de valores fornecidos pelos próprios autores para realizar os testes de execução. Para o protocolo de Cayrel et al. os parâmetros sugeridos eram $n = 64$, $m = 2048$ e $q = 257$. Para esta configuração, o melhor algoritmo de redução de bases de reticulados retorna vetores de norma euclidiana acima de 42, enquanto que a chave privada do protocolo tem norma 32. A partir deste conjunto inicial, foram aplicadas algumas variações para realização dos testes de execução, conforme será descrito na Seção 4.4.2.

No artigo em que o protocolo de Kawachi et al. é apresentado, não são fornecidos valores reais para serem utilizados na prática. Dessa forma, adotou-se o mesmo intervalo de valores do protocolo de Cayrel et al., devido à construção semelhante dos algoritmos e uso da mesma função de compromisso.

Lyubashevsky fornece alguns exemplos de parâmetros a serem utilizados com seu protocolo, mas cuja variação não é tão simples de ser interpretada em termos de efeito no desempenho do protocolo. Mais sobre esses parâmetros será comentado na Seção 4.4.3.

---

[1] De fato, a partir de testes isolados em que apenas selecionava-se aleatoriamente a matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ e verificava-se seu posto, sem preocupações com desempenho, pode-se averiguar experimentalmente que a probabilidade da matriz não possuir posto completo é ínfima.

4.3.3 Função de Compromisso para Reticulados Ideais

A função de compromisso utilizada pelos protocolos de Kawachi e Cayrel et al. envolvem a multiplicação de matrizes por vetores, cujas dimensões são determinadas pelos parâmetros m e n. Em termos gerais, a ordem das multiplicações em si está relacionada ao valor de n, enquanto que o inteiro m define, em conjunto com n, a quantidade de operações realizadas no total, devido à construção da função através do método de Merkle-Damgård.

Além disso, conforme mencionado na Seção 4.1.2.3, fez-se uso do gprof para verificar as porções de código onde os programas desenvolvidos despendiam mais tempo durante a sua execução. Verificou-se que para ambos os protocolos, os maiores gastos ocorrem no cálculo da função de compromisso, a qual é responsável por mais de 90% do tempo de execução.

Nas implementações que utilizam reticulados ideais, a matriz $\mathbf{A}$ utilizada como parte da chave pública é ideal por blocos, de forma que a sua multiplicação por vetores possa ser tratada como a soma de multiplicações por polinômios. Contudo, se as matrizes $\mathbf{B}$ e $\mathbf{C}$ empregadas na função de compromisso não assumirem também nenhuma estrutura especial, as operações nas quais estão envolvidas não poderão desfrutar da mesma conveniência.

Por este motivo, considerou-se que as matrizes $\mathbf{B}$ e $\mathbf{C}$ podem ser construídas de tal forma que as multiplicações por vetores também possam ser reduzidas para multiplicações de polinômios.

4.3.4 Paralelização com o OpenMP

No modelo de programação oferecido pelo OpenMP, por padrão, a maior parte das variáveis se encontra visível a todas as *threads*. Entretanto, quando necessário (usualmente para evitar condições de corrida), pode-se definir individualmente como será realizada a partilha de dados, através de cláusulas como private (os dados dentro da região paralela são individuais para cada *thread*, isto é, cada *thread* possui sua cópia local das variáveis) ou shared (os dados são compartilhados por todas as *threads*, e pode ocorrer acesso simultâneo).

Nas implementações desenvolvidas, a maioria das variáveis eram do tipo compartilhada, pois não havia necessidade de se precaver contra acessos simultâneos. Contudo, em alguns testes ocorreram erros de violação de acesso, devido à dimensão das estruturas

utilizadas. Nestes casos, foi necessário aumentar o tamanho da pilha de execução, para evitar falha de segmentação durante a execução paralela.

4.4 Dados Experimentais

Para cada uma das implementações realizadas, foram efetuados testes coletando o tempo de execução total do protocolo. Na maioria dos casos, não foram medidos separadamente os tempos da etapa de geração de chaves e da fase iterativa, visto que com um grande número de rodadas, o primeiro tempo tende a ser desprezível. Assim, os dados aqui apresentados foram coletados a partir do uso do comando time do Linux [64], sempre considerando o valor do *user time*, isto é, o tempo efetivamente gasto somente para executar as instruções do programa desenvolvido[2].

Nas subseções a seguir, serão apresentados os dados experimentais coletados por protocolo, seguidos de uma comparação geral. Como existem mais implementações para uns protocolos do que outros (conforme visto na Tabela 4.1 da Seção 4.2), aqueles que apresentaram maior quantidade de programas desenvolvidos, naturalmente proporcionaram mais dados experimentais para serem analisados.

4.4.1 Protocolo de Kawachi et al.

Conforme mencionado anteriormente, o protocolo de Kawachi et al. envolve três parâmetros inteiros, m, n e q, além do número de rodadas t. Uma forma de selecionar valores para esses parâmetros é considerar as implicações do ponto de vista teórico que essa escolha acarretará na segurança do protocolo (conforme comentado na Seção 4.3.2). Somente a execução das implementações desenvolvidas, entretanto, irá fornecer uma noção mais concreta do seu desempenho em uma determinada plataforma.

No caso do protocolo de Kawachi et al., o tempo de execução naturalmente será proporcional ao número de rodadas t. Além disso, sabemos que o parâmetro q afeta a ordem dos valores com que se irá trabalhar. As implementações em C assumem que todos os números envolvidos podem ser armazenados em inteiros de até quatro bytes, enquanto

[2]No caso das versões paralelizadas, o tempo apresentado é o valor do *user time* dividido pelo número de *threads*, visto que o comando time informa o tempo total gasto em todos os núcleos ou processadores.

que as implementações que utilizam a NTL aceitam números arbitrariamente grandes. De qualquer forma, considerou-se que a faixa de valores satisfatórios para q é relativamente pequena, de forma que variar esse parâmetro apresente pouca influência no tempo total do protocolo[3]. Já os parâmetros m e n determinam a dimensão das matrizes e vetores manipuladas pelos programas e portanto requerem uma análise mais cuidadosa.

Ao observar a estrutura geral do protocolo, podemos notar quatro tipos de operações:

1. seleção aleatória de valores;
2. cálculo de parâmetros para a função de compromisso;
3. cálculo dos valores de compromisso propriamente ditos, a partir dos parâmetros fornecidos;
4. comparação de valores (etapa de verificação).

As operações 1 e 4 são relativamente simples e rápidas, sendo realizadas em tempo $O(n)$. A multiplicação de matrizes por vetores, que corresponde ao cálculo mais custoso do protocolo, é realizada nas operações do tipo 2 ou 3. Contudo, para determinar os parâmetros para a função de compromisso (operação do tipo 2), realiza-se no máximo uma multiplicação dessa espécie para cada chamada da função, ao multiplicar uma matriz $n \times m$ por um vetor de dimensão m. Já nas operações do tipo 3, são realizadas diversas multiplicações, porém os elementos multiplicados possuem dimensão menor, devido à construção da função de compromisso via técnica de Merkle-Damgård. Para essa construção, podemos dizer que o parâmetro m determina quantas multiplicações serão realizadas (isto é, em quantas partes o vetor de entrada será dividido, no caso $\lceil m/n \rceil$), enquanto que a dimensão da matriz e dos vetores envolvidos é definida pelo parâmetro n. Para as implementações que consideram reticulados ideais, também vale essa relação de parâmetros, visto que os vetores de dimensão m e as matrizes de dimensão $n \times m$ são substituídos por $\lceil m/n \rceil$ polinômios de grau $n - 1$.

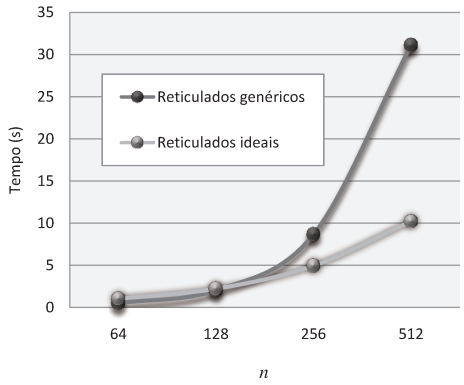
[3]Isto é, seria indiferente para o tempo total de execução se o valor de q resultasse em inteiros de 1, 2, 3 ou 4 bytes, visto que não há nenhuma otimização explícita para esses casos. De fato, foram realizados alguns testes nesse sentido que comprovaram essa hipótese.

Implementação em C		
n	Reticulados Genéricos	Reticulados Ideais
64	0.51 s	1.02 s
128	2.04 s	2.21 s
256	8.67 s	4.93 s
512	31.11 s	10.20 s

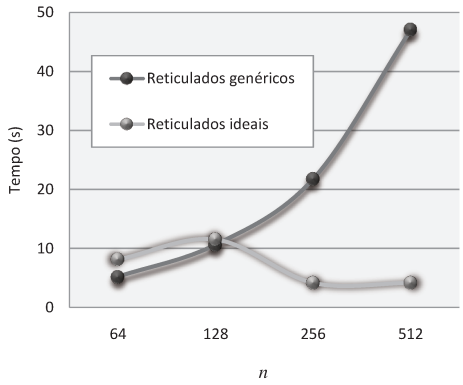
Implementação em C++		
n	Reticulados Genéricos	Reticulados Ideais
64	5.22 s	8.10 s
128	10.64 s	11.51 s
256	21.77 s	4.15 s
512	47.01 s	4.16 s

Tabela 4.2: Tempo médio de execução das implementações do protocolo de Kawachi et al. com variação do parâmetro n.

Dessa forma, é interessante observar o comportamento do protocolo em função de variações nos parâmetros m e n. A Tabela 4.2 e a Figura 4.4 exibem o tempo médio de execução quando os valores de n variam de 64 até 512, tanto para as implementações utilizando a NTL, quanto para as implementações em C. O valor de m ficou fixo em 2048, o de q em 257, e em cada execução foram realizadas 100 rodadas do protocolo iterativo. O tempo de execução apresentado corresponde à média das medidas para 30 execuções do protocolo completo.



(a) Implementação em C.



(b) Implementação em C++ utilizando biblioteca NTL.

Figura 4.4: Tempo médio de execução das implementações do protocolo de Kawachi et al. com variação do parâmetro n.

Como as implementações em C não utilizam bibliotecas adicionais, seu gráfico (Figura 4.4a) pode ser mais facilmente interpretado. No caso da implementação que considera reticulados genéricos, sabemos que as multiplicações levam tempo $O(n^2)$, enquanto que a utilização de reticulados ideais garante tempo de operação $O(n \log n)$, em função do uso da FFT. Além disso, embora as curvas do gráfico reflitam esse comportamento assintótico, para o menor valor $n = 64$, o programa desenvolvido com reticulados ideais levou cerca do dobro do tempo da implementação com reticulados genéricos, e para $n = 128$, o tempo de execução deste último também foi inferior, ainda que a diferença seja bem menos significativa.

Conforme mencionado anteriormente, nas implementações em C que trabalham com reticulados ideais, utilizou-se o algoritmo clássico da FFT, a princípio sem otimizações adicionais. A introdução de melhorias na porção do código referente a essa função poderia reduzir o tempo de execução de uma forma geral, de modo que mesmo para valores menores de n, a implementação com reticulados ideais se tornasse competitiva com a implementação que utiliza reticulados genéricos. Contudo, os maiores ganhos sempre ocorrerão de fato conforme o valor de n aumenta.

A curva da implementação em C++ (Figura 4.4b) utilizando reticulados genéricos segue um comportamento semelhante ao observado no gráfico da implementação em C, porém com valor inicial bem mais elevado e crescimento um pouco menos acelerado. O primeiro fato possivelmente se deve aos custos indiretos relacionados ao uso das classes da NTL, desenvolvidas para trabalhar com números inteiros de tamanho arbitrário. Porém, o algoritmo de multiplicação de matrizes por vetores da biblioteca provavelmente é mais eficiente que o da implementação em C, o que explicaria o crescimento um pouco mais lento da curva.

A implementação que utiliza reticulados ideais, por sua vez, apresenta um comportamento claramente distinto do programa equivalente desenvolvido em C. Neste caso, segundo a documentação da NTL, a operação de multiplicação (de polinômios) é implementada utilizando um de quatro algoritmos distintos: algoritmo clássico, algoritmo de Karatsuba [36, 37], algoritmo de Schoenhage-Strassen [60] e algoritmo baseado no Teorema Chinês do Resto e FFT. Sobre qual algoritmo é empregado em cada caso em si, não há muitas informações, o que dificulta a análise dos dados experimentais. É mencionado, apenas, que a seleção é realizada através de um método heurístico, de forma que a escolha

nem sempre é perfeita. Contudo, é evidente que na execução dos testes para $n = 128$ e $n = 256$ foram utilizados algoritmos distintos, pois mesmo ao se dobrar o tamanho das estruturas utilizadas, o tempo médio apresentou uma queda bastante acentuada. De qualquer forma, pode-se dizer que, assim como para implementação em C, o aprimoramento da função de multiplicação (neste caso, representado também pela seleção de um algoritmo mais adequado) provê melhores resultados, com os ganhos sendo mais pronunciados para maiores valores de n.

Considerando agora variações apenas no parâmetro m, os testes realizados resultaram nos dados experimentais apresentados na Tabela 4.3 e na Figura 4.5. Nestes casos, manteve-se n fixo em 64, q em 257, e variou-se m de 2048 a 16384, sempre dobrando o seu valor. Assim como no teste anterior, foram realizadas 30 execuções do protocolo completo, cada uma com 100 rodadas[4]. Naturalmente, a primeira linha da Tabela 4.2 apresenta os mesmos valores da primeira linha da Tabela 4.3, pois estas correspondem ao mesmo teste ($n = 64$ e $m = 2048$).

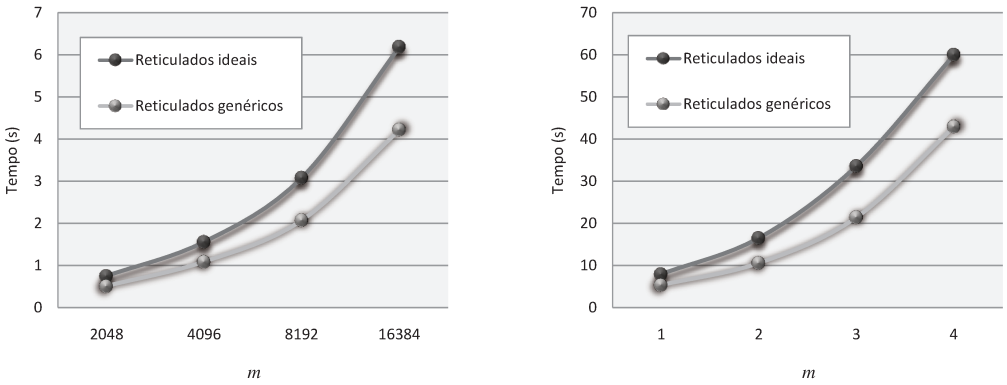
Implementação em C		
m	Reticulados Genéricos	Reticulados Ideais
2048	0.51 s	0.75 s
4096	1.09 s	1.56 s
8192	2.07 s	3.08 s
16384	4.21 s	6.18 s

Implementação em C++		
m	Reticulados Genéricos	Reticulados Ideais
2048	5.22 s	8.10 s
4096	10.57 s	16.52 s
8192	21.39 s	33.57 s
16384	42.85 s	59.92 s

Tabela 4.3: Tempo médio de execução das implementações do protocolo de Kawachi et al. com variação do parâmetro m.

Conforme esperado, o parâmetro m não apresenta a mesma influência de n na diferença de tempos entre as implementações utilizado reticulados genéricos e as implementações que empregam reticulados ideais. Em ambas, m determina principalmente o número de multiplicações de ordem n a serem realizadas no cálculo da função de compromisso.

[4]Este procedimento foi adotado em todos os testes, e portanto não será mais mencionado a partir deste ponto.



(a) Implementação em C.

(b) Implementação em C++ utilizando biblioteca NTL.

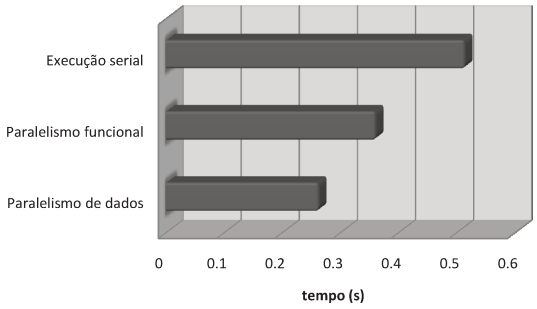
Figura 4.5: Implementações do protocolo de Kawachi et al. com variação do parâmetro m.

Segundo o gprof, os programas gastam mais de 90% do seu tempo de execução realizando esta tarefa, em ambos os casos, de forma que alterar o valor de m apresenta o mesmo efeito para os dois tipos de implementação. No caso, a relação é quase linear, pois ao dobrar o parâmetro m, o tempo médio de execução também é praticamente multiplicado por dois. Para a implementação que considera reticulados genéricos, a princípio haveria ainda a pequena diferença de que m também determina o número de colunas da matriz cuja multiplicação por vetor irá gerar alguns dos argumentos da função de compromisso. Todavia, esta etapa do protocolo na prática mostra-se menos significante do que o cálculo da função de compromisso em si, conforme apontado pelo gprof. Por fim, notamos que a implementação com reticulados ideais sempre apresenta desempenho inferior. Isto se deve ao fato dos programas que adotam reticulados genéricos, tanto os desenvolvidos em C quanto os em C++, serem levemente mais rápidos para $n = 64$, como visto anteriormente. Para valores maiores de n, pode ser observado que esta relação entre os dois tipos de implementação se inverte.

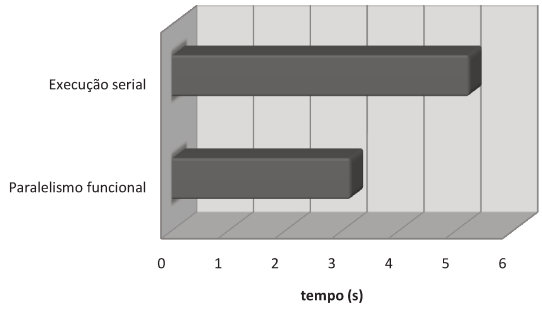
Passando para as implementações que apresentam algum tipo de paralelização, foram obtidos os tempos médios de execução apresentados na Figura 4.6, para $n = 64$ e $m = 2048$. Os maiores ganhos ocorreram na implementação com paralelismo de dados da Figura 4.6a, que corresponde à paralelização da operação de multiplicação de matriz por vetor. Embora estejam sendo utilizadas quatro *threads*, não é esperado que o tempo

de execução seja também reduzido para um quarto do tempo original do programa com execução serial, visto que a paralelização apresenta inúmeros outros custos para criação e sincronização das *threads*. No total, para a implementação com paralelismo de dados, houve uma redução para quase metade do tempo de execução original, o que já representa uma melhoria muito significativa.

Infelizmente, como para a implementação em C++ esta operação de multiplicação está encapsulada no método correspondente da classe de matrizes, essa forma de paralelização não foi possível. Porém, o uso de paralelismo funcional garantiu um ganho relativo semelhante, com diminuição do tempo total para cerca de 60% da execução serial. Para implementação em C, o mesmo tipo de paralelismo acarretou uma melhoria para cerca de 70% do tempo original. Possivelmente, esta diferença se deve ao fato de que as implementações não são iguais quando se considera a complexidade e o tempo demandado pelas operações na região paralela em relação aos custos de controle das *threads*. Além disso, devemos mencionar que a paralelização realizada define apenas até três seções para serem executadas simultaneamente, uma para cada valor de compromisso calculado. Assim, quando temos mais de três núcleos (como no caso destes testes), não estaremos aproveitando o potencial máximo do processador para execução paralela.



(a) Implementação em C.



(b) Implementação em C++ utilizando biblioteca NTL.

Figura 4.6: Implementações do protocolo de Kawachi et al. com paralelização via OpenMP.

4.4.2 Protocolo de Cayrel et al.

Assim como o protocolo anterior, o protocolo de Cayrel et al. apresenta os mesmos parâmetros inteiros m, n, q e t. Além disso, espera-se que a influência desses parâmetros no tempo total de execução também seja parecida, devido à construção semelhante dos protocolos e uso da mesma função de compromisso. Dessa forma, para coletar os dados de execução, procurou-se variar os mesmos parâmetros, isto é, m e n e, ao final, observou-se um comportamento similar ao dos testes anteriores.

Os resultados em função da variação do parâmetro m foram muito parecidos com os do protocolo de Kawachi et al. e, portanto, também neste caso este parâmetro apresenta pouca influência na diferença de desempenho entre as implementações que utilizam reticulados genéricos e aquelas que aceitam apenas reticulados ideais. Evidentemente, há um impacto no tempo total de execução do protocolo, mas seguindo o mesmo comportamento dos gráficos da Figura 4.5. Por esse motivo, os valores exatos de tempo de execução para diferentes m não serão apresentados neste texto para esse segundo protocolo.

n	Reticulados Genéricos	Reticulados Ideais
64	5.10 s	6.72 s
128	10.13 s	8.28 s
256	21.58 s	4.03 s
512	47.72 s	5.22 s

Tabela 4.4: Tempo médio de execução para o protocolo de Cayrel et al. com variação de n.

Por outro lado, embora a variação de n também proporcione resultados similares ao do protocolo anterior, é interessante apresentar os dados coletados de maneira detalhada, especialmente devido ao comportamento peculiar da implementação com reticulados ideais.

A Tabela 4.4 apresenta os tempos médios de execução do protocolo com o parâmetro n variando de 64 a 128, tanto para a implementação com reticulados genéricos quanto a implementação com reticulados ideais. A Figura 4.7 corresponde à representação gráfica desses dados.

Assim como para o protocolo de Kawachi et al., o desempenho da implementação com reticulados ideais é levemente inferior para $n = 64$. Entretanto, os tempos de execução desta implementação já são melhores que os do programa que utiliza reticulados genéricos mesmo para $n = 128$, embora para este primeiro valor a diferença seja pequena.

O mais interessante, porém, permanece por conta do comportamento da curva da

implementação de reticulados ideais, a qual não é estritamente crescente. Mais uma vez, de $n = 128$ para $n = 256$, temos uma queda no tempo de execução, o contrário do que seria esperado quando consideramos um único algoritmo de multiplicação. Contudo, como já comentado, a biblioteca NTL utiliza um de quatro algoritmos distintos de multiplicação de polinômios, com seleção via método heurístico. O único comentário que se pode fazer com certeza é, portanto, de que a ordem dos polinômios envolvidos na multiplicação certamente influencia a escolha do algoritmo empregado.

Sobre as possibilidades de paralelização, como todas as implementações foram realizadas utilizando a biblioteca NTL, a única estratégia adotada foi a de se calcular simultaneamente os valores de compromisso. Infelizmente, o demonstrador realiza apenas o cálculo de dois desses valores, enquanto que o verificador trabalha apenas com um valor, em função do desafio enviado. Na tentativa de se paralelizar também as operações do verificador, no caso de desafio $b = 0$, separou-se em duas seções da região paralela o cálculo do argumento $\mathbf{AP}_\sigma^{-1}\beta - \alpha\mathbf{y}$, de forma que $\mathbf{AP}_\sigma^{-1}\beta$ fosse determinado em uma *thread* e $\alpha\mathbf{y}$ em outra, embora não se esperasse ganhos significativos neste caso, como de fato ocorreu. Ao final, a implementação com paralelismo funcional, apresentou uma redução de 20% no tempo total de execução, conforme observado na Figura 4.8.

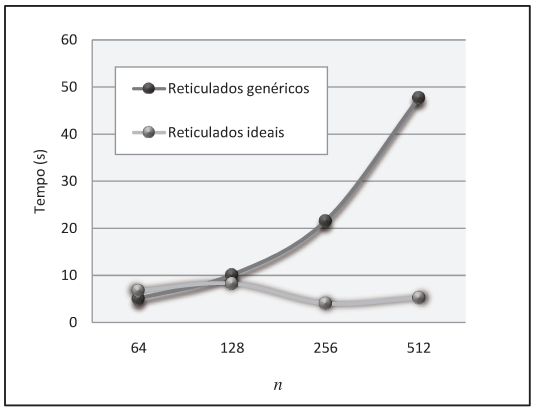


Figura 4.7: Representação gráfica do tempo médio de execução para o protocolo de Cayrel et al. com variação do parâmetro n.

4.4.3 Protocolo de Lyubashevsky II

Conforme mencionado na Seção 4.3.2, no artigo original, Lyubashevsky sugere alguns conjuntos de parâmetros para serem utilizados na prática com seu protocolo. As implicações da variação desses parâmetros no desempenho de execução não são tão simples de serem analisadas, pois não afetam apenas o tamanho das estruturas envolvidas, mas também a dimensão de certos domínios, ao restringir as normas máximas dos vetores.

Basicamente, os parâmetros m, n, σ e κ determinam os limites para as normas, mas as

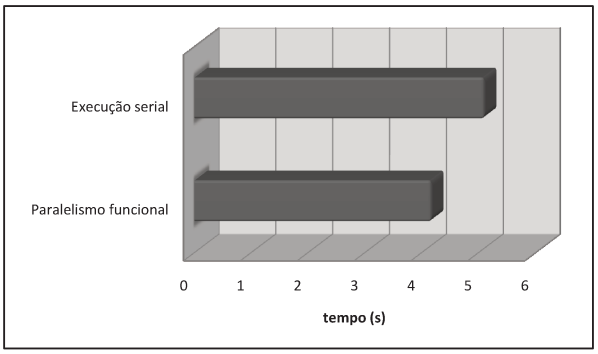


Figura 4.8: Implementações do protocolo de Cayrel et al. com execução serial e paralelização via OpenMP.

dimensões são estabelecidas por n. Além disso, o parâmetro m é de grande importância na definição de $p \approx (2\sigma + 1)^m.2^{-\frac{128}{n}}$, que por sua vez determina a ordem de grandeza dos coeficientes dos polinômios. Por exemplo, nos casos em que $n = 512$, $\sigma = 2047$ e $\kappa = 24$, a mudança de $m = 5$ para $m = 8$ faz com que p passe de $2^{59.8}$ para $2^{95.8}$.

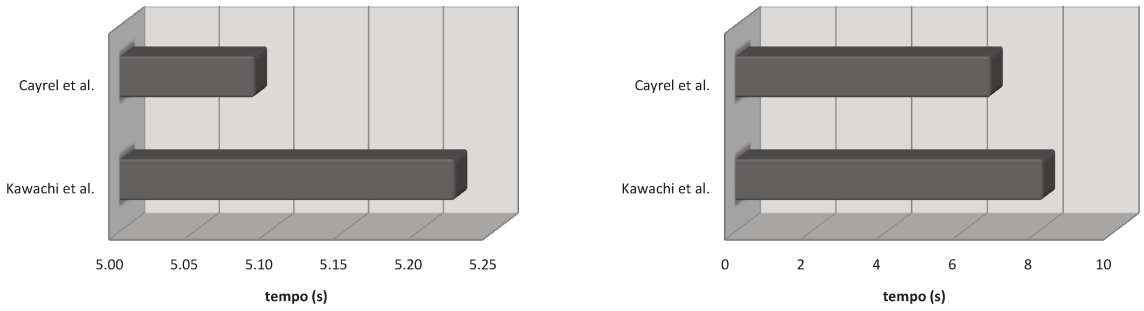
Em geral, podemos dizer que o protocolo trabalha com vetores e polinômios com dimensão e grau, respectivamente, menores do que os anteriores, mas onde os elementos e coeficientes podem ser bem maiores. No caso, como a NTL foi desenvolvida para trabalhar com inteiros de tamanho arbitrário, o desempenho dos programas parece ser mais afetado pelas dimensões dos vetores e grau dos polinômios do que pelo tamanho de seus elementos/coeficientes.

Assim, para os parâmetros $n = 512$, $m = 5$ e $p = 2^{59.8}$ sugeridos pelo autor, o tempo total medido para o protocolo de Lyubashevsky foi bastante inferior (em média 1.06s) ao das versões que utilizam a NTL dos outros dois protocolos, mas comparável ao tempo da implementação com reticulados ideais em C do protocolo de Kawachi et al., que já supõe que os inteiros envolvidos não serão tão grandes.

4.4.4 Comentários Gerais

Quando executamos o protocolo de Cayrel et al. com $n = 64$ (o valor sugerido pelos autores para ser utilizado na prática) e o comparamos com o protocolo de Kawachi et al. com o mesmo conjunto de parâmetros, notamos que o primeiro se mostrou um pouco mais rápido quando consideramos reticulados genéricos (Figura 4.9a). Como o número de valores de compromisso é diferente para estes dois protocolos (três para o primeiro, contra cinco

do último), porém calculados a partir da mesma função, é esperado que o protocolo que envolva cálculo de mais valores apresente um desempenho inferior, especialmente quando se leva em conta as informações do gprof de que a maior fração do tempo despendido pelo programa corresponde a esse tipo de operação.



(a) Reticulados genéricos.

(b) Reticulados ideais.

Figura 4.9: Tempo médio de execução das implementações não paralelizadas.

Por outro lado, quando comparamos as implementações envolvendo reticulados ideais (Figura 4.9b), a diferença é levemente mais acentuada. Embora os algoritmos sejam semelhantes e utilizem a mesma função de compromisso, o número de multiplicações substituídas por operações com polinômios não é o mesmo, de forma que a passagem da versão com reticulados genéricos para ideais não produz efeitos idênticos para os dois protocolos.

O protocolo de Lyubashevsky, por sua construção, permite somente o uso de reticulados ideais e, nesses casos, apresentou desempenho superior ao dos outros dois protocolos. Entretanto, conforme já comentado, isso se deve provavelmente ao uso das classes da NTL desenvolvidas para trabalhar com inteiros arbitrariamente grandes. Quando comparamos a implementação de testes do protocolo de Lyubashevsky à versão do protocolo de Kawachi et al. (empregando reticulados ideais) que não utiliza a NTL, o desempenho é bastante semelhante.

Em termos de dificuldade de programação, como o protocolo de Lyubashevsky utiliza coeficientes grandes, implementá-lo sem o auxílio de uma biblioteca adicional pode constituir um trabalho bem mais árduo. No caso dos outros dois protocolos, podemos abrir

mão deste recurso, ou então buscar uma biblioteca que melhor se adeque às necessidades de programação, isto é, que efetue multiplicações ou a Transformada Rápida de Fourier eficientemente, mas sem supor o uso de inteiros grandes.

De qualquer modo, para uma comparação mais precisa entre os protocolos, outros parâmetros devem ser ajustados, tais como o número de rodadas efetuadas. No capítulo seguinte, apresentaremos uma análise mais detalhada neste sentido.

Capítulo 5

Discussão

Ao estudar diversos algoritmos que compartilham uma mesma finalidade (no nosso caso, a identificação de entidades), surge um interesse natural em determinar, de alguma forma, qual destes apresenta maiores vantagens ou é mais adequado para o uso na prática. Neste capítulo, será apresentada uma análise nesse sentido, tendo como objeto de discussão os protocolos abordados. Serão considerados tanto os dados coletados a partir do experimento de implementação quanto as características teóricas levantadas no Capítulo 3.

Ainda, discutiremos sobre as diferentes categorias de otimização que podem ser aplicadas aos protocolos, e seu impacto na tentativa de comparação dos mesmos. Além das melhorias cuja influência buscamos testar no experimento de implementação, serão mencionadas algumas sugestões adicionais para novas otimizações.

Nessa mesma direção, iremos apresentar algumas possibilidades de estender o trabalho aqui apresentado, seja na forma de otimizações complementares, ou a partir da realização de diferentes formas de análise.

5.1 Discussão Sobre os Protocolos Implementados

Na tentativa de realizar a comparação de diferentes algoritmos, o primeiro passo consiste em determinar os critérios de avaliação a serem empregados. No caso dos protocolos de identificação aqui abordados, podemos dizer, de forma simples, que desejamos algoritmos ao mesmo tempo seguros e eficientes. Sabe-se, entretanto, que existem restrições na capacidade de escolha desses dois elementos. No geral, pode-se aprimorar as margens de

segurança providas pelos protocolos aumentando certos parâmetros a eles associados. Em contrapartida, este tipo de medida torna o tempo de execução do protocolo mais lento, em função das limitações dos dispositivos computacionais empregados (Figura 5.1). Deve-se, portanto, determinar um ponto de equilíbrio para esses dois critérios, quando do uso do algoritmo em situações práticas.

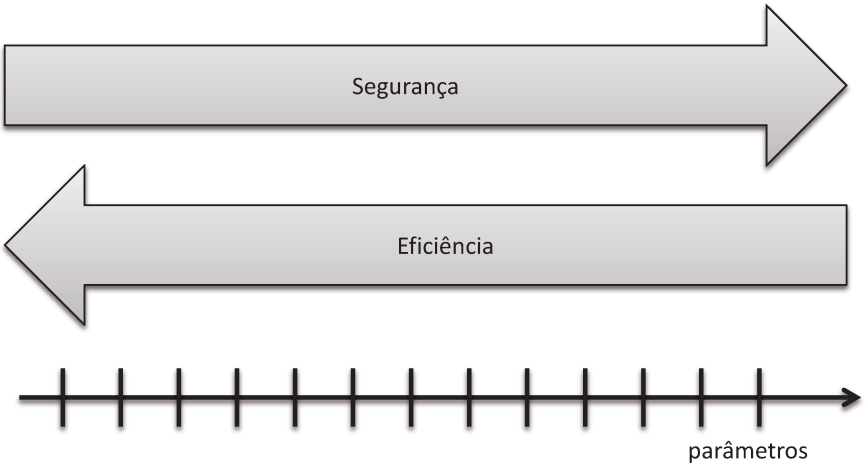


Figura 5.1: Simplificação da relação de compromisso entre segurança e eficiência dos protocolos.

Ainda em relação a eficiência, as preocupações não devem estar restritas ao tempo de execução do algoritmo. Considerando que a identificação de usuários muitas vezes poderá ser realizada a partir de equipamentos com capacidades bastante limitadas, como é o caso dos dispositivos móveis, deve-se observar quão eficaz é o aproveitamento da memória disponível. Usualmente, este critério também é diretamente influenciado pelos parâmetros de segurança adotados.

Neste mesmo sentido, os tamanhos em bytes dos valores trocados entre demonstrador e verificador também devem ser considerados, pois estão vinculados ao canal de comunicação empregado. Em certas situações, é possível que haja limitações na capacidade de transmissão do canal, de modo que grandes quantidades de dados podem ocasionar lentidão ou congestionamento.

Por fim, para uma análise completa, é interessante determinar não somente qual pro-

tocolo é mais promissor em relação a um determinado critério ou conjunto de critérios. Deve-se também procurar averiguar de quais características decorrem suas vantagens, e como as mesmas podem ser aproveitadas para o desenvolvimento de novos algoritmos ou aprimoramento dos algoritmos já existentes.

5.1.1 Tempo de Execução

No Capítulo 4, foram apresentadas medidas experimentais dos tempos de execução dos três protocolos implementados. Embora neste experimento tenha sido incluída a etapa de geração de chaves, a maior parte do processamento correspondeu ao protocolo interativo propriamente dito. Além disso, embora os protocolos tenham sido executados com 100 rodadas, devemos enfatizar que este número pode ser modificado de acordo com as necessidades de segurança. Dessa forma, os dados coletados devem ser interpretados como um indicativo do *tempo médio por rodada*. Na prática, para efeito de comparação, deve-se considerar que o número de rodadas necessárias para atingir um mesmo nível de segurança varia de protocolo para protocolo, de forma que mesmo que o tempo médio por rodada seja inferior, é possível que a quantidade mínima de repetições torne o algoritmo mais lento, de uma forma geral.

Sabemos que o tempo médio por rodada é determinado pelas operações realizadas ao longo do protocolo, conforme mencionado no capítulo anterior. O número mínimo de rodadas, por sua vez, pode ser definido, por exemplo, em função do erro de coerência, cujo valor deseja-se minimizar. Nos protocolos implementados, temos um erro de coerência de $(2/3)^t$ para o protocolo de Kawachi et al. e 2^{-t} para o protocolo de Cayrel et al. Desta forma, além de apresentar tempos médios de execução por rodada inferiores (conforme a Figura 4.4.6), este último leva vantagem também em relação ao número de repetições necessárias.

No caso do protocolo de Lyubashevsky II, o desafio binário é substituído por um polinômio. Dessa forma, enquanto nos outros dois protocolos o erro de coerência é controlado pelo número t de rodadas, no protocolo de Lyubashevsky, este erro passa a ser determinado pelo parâmetro κ, que define o tamanho do domínio do qual os desafios são extraídos. Nos testes realizados, o valor de κ adotado é tal que o erro de coerência é de 2^{-80}, o que corresponderia a aproximadamente 137 rodadas do protocolo de Kawachi et al. e 80 rodadas do protocolo de Cayrel et al.

Ainda em relação ao protocolo de Lyubashevsky, embora o erro de coerência seja determinado por κ, certo número de rodadas também deve ser realizado, em função do erro de completude. Isto é, como existe a possibilidade do demonstrador abortar uma interação do protocolo, deve-se garantir que, ao final, um demonstrador legítimo não tenha sua prova de identidade recusada. Considerando uma única interação, a probabilidade de uma entidade fraudulenta conseguir produzir uma resposta válida é muito baixa (erro de coerência de 2^{-80}). Dessa forma, o protocolo pode ser repetido até que o demonstrador consiga responder um desafio sem abortar a execução do protocolo. Ou então, em uma estratégia mais simples, porém possivelmente menos eficiente, pode-se realizar uma quantidade fixa de rodadas pré-estabelecida. Nos testes, o número de rodadas realizadas foi fixado em 30 repetições, de forma que a probabilidade de um demonstrador autêntico ser recusado (isto é, não conseguir responder a nenhum desafio) ficasse em torno de 2^{-20} [43].

A Tabela 5.1 apresenta os tempos médios por rodada dos protocolos de Kawachi et al. e Cayrel et al., calculados a partir dos dados experimentais, considerando que o tempo investido na etapa de geração de chaves é desprezível quando comparado ao tempo gasto no restante do algoritmo (porção interativa). Além disso, dado o número de rodadas necessárias para que o erro de coerência seja igual ou inferior a 2^{-80}, podemos estimar qual seria o tempo total de execução nessa condição (isto é, com erro de coerência $\leq 2^{-80}$). Para os demais parâmetros (n, m, q), foram utilizados os valores $(64, 2048, 257)$ recomendados por Cayrel et al. [11], e todos os tempos se referem às implementações em C++.

Protocolo / Reticulados	Tempo Médio por Rodada	Rodadas Necessárias para Erro de Coerência $\leq 2^{-80}$	Tempo Total para Erro de Coerência $\leq 2^{-80}$
Kawachi et al. / genéricos	52.2 ms	137	7.15 s
Kawachi et al. / ideais	81.0 ms	137	11.1 s
Cayrel et al. / genéricos	51.0 ms	80	4.08 s
Cayrel et al. / ideais	67.2 ms	80	5.38 s

Tabela 5.1: Comparação do número de rodadas necessárias e tempos de execução dos protocolos de Kawachi et al. e Cayrel et al. para erro de coerência $\leq 2^{-80}$.

Considerando os três protocolos abordados, em princípio, talvez fosse mais natural observar o comportamento dos protocolos a partir das implementações que utilizam reticulados ideais, já que o protocolo de Lyubashevsky não possui versão com reticulados genéricos. Porém, dado que para esse conjunto de parâmetros as implementações com reticulados ideais apresentaram desempenho um pouco inferior, construiu-se o gráfico da Figura 5.2 utilizando os tempos de execução das versões dos protocolos de Kawachi et al. e Cayrel et al. que empregam reticulados genéricos.

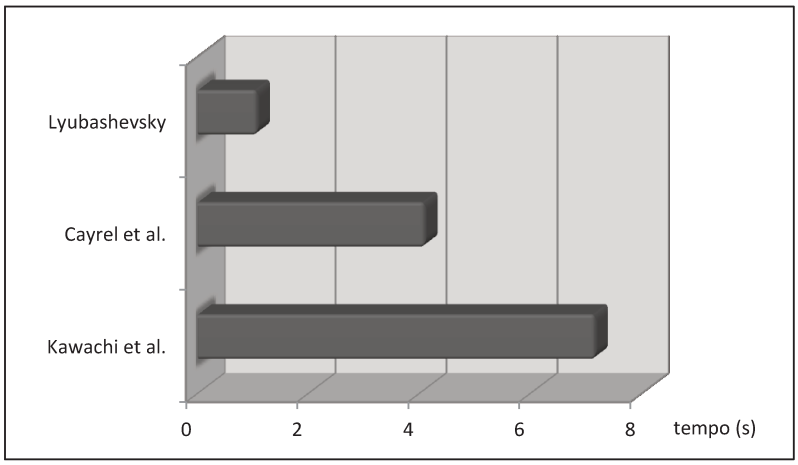


Figura 5.2: Tempos de execução dos protocolos para erro de coerência $\leq 2^{-80}$, considerando as implementações que utilizam a biblioteca NTL.

Assim, se confrontarmos os tempos de execução da Tabela 5.1 com as medidas obtidas para o protocolo de Lyubashevsky, notamos que esse último apresenta um desempenho superior (Figura 5.2). Porém, conforme já mencionado no capítulo anterior, uma das vantagens dos outros dois protocolos sobre o de Lyubashevsky consiste no uso de inteiros menores, ainda que em maior quantidade, enquanto que a biblioteca NTL foi construída para otimizar as operações envolvendo inteiros grandes.

Comparando, entretanto, a implementação em C do protocolo de Kawachi et al., executada também com 137 rodadas e o mesmo conjunto de parâmetros, com o protocolo de Lyubashevsky, verifica-se uma diferença bem menos acentuada, seja na versão com reticulados genéricos ou na com reticulados ideais (Figura 5.3). Como consequência, supondo que há uma certa transitividade na relação entre os tempos de execução dos protocolos,

podemos supor que uma implementação do algoritmo de Cayrel et al. sem o uso da NTL apresentaria um desempenho inferior, ou pelo menos equivalente, ao do protocolo de Lyubashevsky, já que requer menos rodadas que o protocolo de Kawachi et al. e apresenta tempo inferior por rodada.

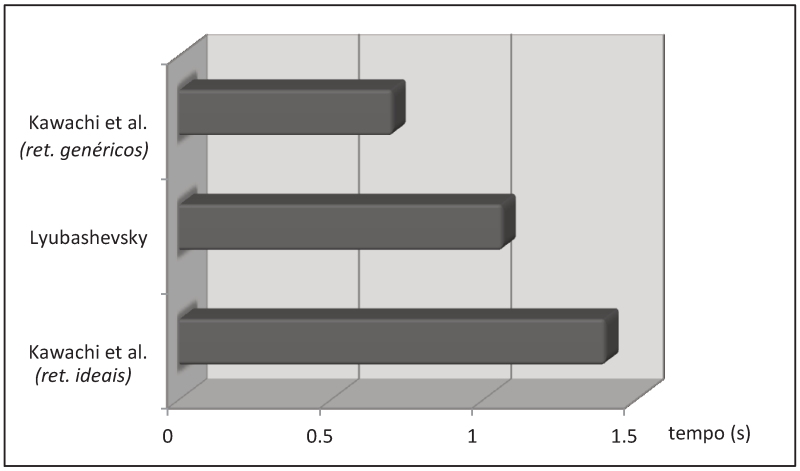


Figura 5.3: Tempos de execução do protocolo de Kawachi et al. para erro de coerência $\leq 2^{-80}$, considerando as implementações em C, e do protocolo de Lyubashevsky.

Em todos os dados apresentados nesta seção, para efeito de comparação, os parâmetros e números de rodadas foram ajustados de forma a obter um limite uniforme para o erro de coerência de todos os protocolos. Na prática, o valor real deste limite é determinado pelas necessidades de segurança e margens de risco de cada aplicação. Em 1999, a Organização Internacional para Padronização (ISO) e a Comissão Eletrotécnica Internacional (IEC) elaboraram em conjunto o padrão ISO/IEC 9798 [34], revisado duas vezes em 2004 e 2009, que aponta os valores no intervalo de 2^{-16} a 2^{-40} como sendo apropriados para o limite do erro de coerência.

5.1.2 Requisitos de Memória

A quantidade de memória utilizada por cada protocolo depende, efetivamente, de fatores de implementação, tais como as estruturas de dados utilizadas, as classes desenvolvidas, as bibliotecas empregadas, entre outros. Podemos determinar, entretanto, uma quantia mínima de memória que deve ser reservada, em função dos valores que precisam ser armazenados durante a execução do algoritmo. A própria linguagem de programação

utilizada, ou o uso de registradores do processador, impõe certos gastos adicionais a este mínimo necessário, mas nesta breve análise estes detalhes não serão contemplados. Assim, se um determinado valor puder ser representado, por exemplo, com até nove bits, iremos considerar o requerimento de memória como sendo exatamente nove bits, embora na prática provavelmente torne-se necessário reservar dois bytes completos para armazenar este valor.

Além disso, para determinar os requisitos de memória, em princípio, poderíamos considerar os diversos valores intermediários que acabam sendo calculados ao longo do protocolo. Contudo, como sua existência também está atrelada à implementação, iremos apenas contabilizar os custos de armazenamento das chaves privada e pública, que sabemos ser primordial.

Para o protocolo de Kawachi et al., a chave privada corresponde ao vetor binário $\mathbf{x}$ de dimensão m e, portanto, ocupa m bits. Já a chave pública é composta da matriz $\mathbf{A}$, de tamanho $n \times m$, com cada elemento variando entre 0 e $q - 1$. Assim, a matriz exige um espaço de armazenamento de ao menos $mn \lceil \log q \rceil$ bits. Também faz parte da chave pública o vetor $\mathbf{y}$, de dimensão n, e cujos elementos são módulo q, resultando num custo de memória de $n \lceil \log q \rceil$ bits. A Tabela 5.2 apresenta um resumo dos requisitos de memória do protocolo de Kawachi et al. e também valores reais em bytes para algumas instanciações dos parâmetros envolvidos.

			$n = 64$ $m = 2048$ $q = 257$	$n = 128$ $m = 2048$ $q = 257$	$n = 64$ $m = 4096$ $q = 257$	$n = 128$ $m = 4096$ $q = 257$
Chave Privada	$\mathbf{x}$	m	256 B	256 B	512 B	512 B
Chave Pública	$\mathbf{A}$	$mn \lceil \log q \rceil$	144 KB	288 KB	288 KB	576 KB
	$\mathbf{y}$	$n \lceil \log q \rceil$	72 B	144 B	72 B	144 B
	Com	$n \times 3n \lceil \log q \rceil$	13.5 KB	54 KB	13.5 KB	54 KB

Tabela 5.2: Requisitos de memória dos protocolos de Kawachi et al. e Cayrel et al.

Como as chaves do protocolo de Cayrel et al. são essencialmente as mesmas, os dados da tabela também são válidos para esse segundo protocolo. Porém, os autores apresentam a função de compromisso Com como sendo pertencente à chave pública, embora Kawachi

			$n = 64$ $m = 2048$ $q = 257$	$n = 128$ $m = 2048$ $q = 257$	$n = 64$ $m = 4096$ $q = 257$	$n = 128$ $m = 4096$ $q = 257$
Chave Privada	$\mathbf{x}$	m	256 B	256 B	512 B	512 B
Chave Pública	$\mathbf{A}$	$m\lceil\log q\rceil$	2.25 KB	2.25 KB	4.5 KB	4.5 KB
	$\mathbf{y}$	$n\lceil\log q\rceil$	72 B	144 B	72 B	144 B
	Com	$3n\lceil\log q\rceil$	216 B	432 B	216 B	432 B

Tabela 5.3: Requisitos de memória dos protocolos de Kawachi et al. e Cayrel et al., considerando reticulados ideais.

et al. não o façam explicitamente. Ainda assim, como para ambos os protocolos é sugerido o uso da mesma função de compromisso, podemos incluí-la na chave pública também para o protocolo de Kawachi et al.

Devido ao uso da construção de Merkle-Damgård, a função de compromisso é definida pelas matrizes $\mathbf{B}$ e $\mathbf{C}$, cujas dimensões são $n \times n$ e $n \times 2n$, respectivamente, com elementos módulo q. Dessa forma, a função completa exige um espaço de armazenamento dado por $n \times 3n\lceil\log q\rceil$ bits.

Evidentemente, os maiores requisitos de memória são decorrentes do armazenamento da matriz $\mathbf{A}$. Como visto, o uso de reticulados ideais pode reduzir estes custos, por permitir uma representação mais sucinta nos casos em que a matriz é ideal por blocos. Assim, para cada um dos m/n blocos, é suficiente especificar apenas n elementos, de modo que no total são necessários $m\lceil\log q\rceil$ bits. Da mesma maneira, a quantidade de memória necessária para representar as matrizes $\mathbf{B}$ e $\mathbf{C}$, que compõem a função de compromisso Com, também tem o seu valor reduzido por um fator n, se adotada a mesma estratégia, como descrito na Seção 4.3.3. Alguns exemplos de custos de memória quando são utilizados reticulados ideais estão apresentados na Tabela 5.3.

No caso do protocolo de Lyubashevsky, a chave privada $\hat{\mathbf{s}}$ corresponde a um conjunto de m polinômios pertencentes ao anel $R = \mathbb{Z}_p[x]/\langle x^n + 1\rangle$, onde cada coeficiente deve ter módulo igual ou inferior a σ. Assim, é preciso armazenar mn coeficientes, e cada um deles ocupa $\lceil\log(2\sigma+1)\rceil$ bits. Em relação à chave pública, $\mathbf{S}$ corresponde a um único polinômio com limite de grau $n-1$ e cujos coeficientes pertencem $\mathbb{Z}_p$. Portanto, exige um espaço

na memória de $n\lceil \log p \rceil$ bits. Além disso, também faz parte da chave pública a função de *hash* h, que é definida basicamente por um vetor $\hat{\mathbf{a}} \in R^m$, que ocupa $mn\lceil \log p \rceil$ bits. Requisitos de memória reais para alguns valores sugeridos pelo autor para os parâmetros do protocolo estão apresentados na Tabela 5.4.

Claramente, as versões com reticulados genéricos dos protocolos de Kawachi et al. e Cayrel et al. apresentam requisitos de memória mais elevados, quando comparadas ao protocolo de Lyubashevsky. Entretanto, em uma comparação mais concludente, que considera apenas as versões envolvendo reticulados ideais (já que este é o caso do protocolo de Lyubashevsky), esta vantagem é perdida.

Neste caso, a sensibilidade aos parâmetros m e n, que determinam o grau dos polinômios e as dimensões das matrizes e vetores, parece ser a mesma para todos os protocolos. Ou seja, qualquer variação no valor desses parâmetros é transmitida através de uma relação linear para algum requerimento de memória dos algoritmos. Porém, quando consideramos os parâmetros que influenciam o tamanho dos inteiros envolvidos, isto é, q para os protocolos de Kawachi et al. e Cayrel et al., e p e σ para o protocolo de Lyubashevsky, algumas observações a mais podem ser feitas.

A rigor, também permanece uma relação igual para todos os protocolos, com algum custo de armazenamento na memória variando com o logaritmo do parâmetro envolvido. Todavia, os valores desses parâmetros, segundo as sugestões dos respectivos autores, podem apresentar um impacto bastante diferente no que diz respeito às técnicas de implementação que devem ser adotadas. Em particular, o parâmetro p do protocolo de Lyubashevsky, por assumir valores reais muito grandes, faz com que os coeficientes dos

			$n = 512$ $m = 4$ $\sigma = 127$ $p = 2^{31.7}$	$n = 512$ $m = 5$ $\sigma = 2047$ $p = 2^{59.8}$	$n = 512$ $m = 8$ $\sigma = 2047$ $p = 2^{95.8}$	$n = 1024$ $m = 8$ $\sigma = 2047$ $p = 2^{95.9}$
Chave Privada	$\hat{\mathbf{s}}$	$mn\lceil \log(2\sigma + 1) \rceil$	2 KB	3.75 KB	6 KB	12 KB
Chave Pública	$\mathbf{S}$	$n\lceil \log p \rceil$	2 KB	3.75 KB	6 KB	12 KB
	h	$mn\lceil \log p \rceil$	8 KB	18.75 KB	48 KB	96 KB

Tabela 5.4: Requisitos de memória do protocolo de Lyubashevsky.

polinômios excedam os limites de armazenamento dos tipos primitivos da maioria das linguagens de programação, os quais são restringidos, de alguma forma, pelo tamanho dos registradores do processador. Além disso, embora certas linguagens ofereçam suporte para aritmética de inteiros com precisão arbitrária, nestes casos os requisitos de memória talvez sejam mais complexos de serem determinados, em função do sistema de representação utilizado, de forma semelhante ao que ocorre quando se utiliza bibliotecas.

5.1.3 Custos de Comunicação

No experimento realizado no capítulo anterior, tanto o demonstrador quanto o verificador foram representados em um único executável e compartilhavam a mesma região de memória. Dessa forma, o experimento não contempla os custos de comunicação dos protocolos. Felizmente, tais custos podem ser estimados com base no tamanho dos dados transmitidos, de forma semelhante à realizada para os requisitos de memória.

Na prática, a relevância destes custos será determinada pela capacidade do canal de transmissão utilizado pelas entidades comunicantes. Havendo limitações na velocidade do tráfego de dados, é possível que os passos de comunicação sejam significativos no tempo total de execução do protocolo. Essa situação pode ser agravada se houver uso simultâneo do canal por outros processos ou entidades. Neste caso, não apenas o desempenho do protocolo de identificação pode ser afetado, mas os demais processos podem também sofrer prejuízo com a concorrência. Ainda, embora essa situação seja menos provável, caso o uso do canal de transmissão esteja sujeito à contratação de serviços, é possível que existam outras limitações, como uma quota finita para a quantidade de dados transferidos.

No protocolo de Kawachi et al., os custos de comunicação são representados basicamente pelos valores de compromisso enviados pelo demonstrador, bem como a resposta ao desafio. O desafio em si é praticamente desprezível, pois representa apenas dois bits por rodada. Cada valor de compromisso corresponde a um vetor de dimensão n, cujos elementos são módulo q. Portanto, considerando os três valores de compromisso, em uma rodada temos um custo de $3n \lceil \log q \rceil$ bits.

Já em relação à resposta ao desafio, pode ser enviado o vetor $\mathbf{r}$, sua soma com o vetor $\mathbf{x}$, permutações de $\mathbf{r}$ ou $\mathbf{x}$, ou uma representação da permutação σ. Exceto σ, os demais valores correspondem a vetores de dimensão m e elementos módulo q, representando um custo de comunicação de $m \lceil \log q \rceil$ bits. Já para a permutação σ, iremos considerar um

vetor de m posições, onde cada elemento pode variar de 0 a $m-1$ (ou de 1 a m, de modo equivalente). Dessa forma, a permutação resulta em $m \lceil \log m \rceil$ bits sendo transmitidos.

Em uma rodada, portanto, podemos ter $2m \lceil \log q \rceil$ bits enviados como resposta ($\sigma(\mathbf{x})$ e $\sigma(\mathbf{r})$, se $Ch = 0$), ou então $m \lceil \log m \rceil + m \lceil \log q \rceil$ bits (σ e $\mathbf{x}+\mathbf{r}$, se $Ch = 1$, ou σ e $\mathbf{r}$, se $Ch = 2$). Considerando que os três valores de desafio ocorrem com igual probabilidade, a primeira situação corresponde a 1/3 dos casos, e a última, a 2/3. Assim, na média temos um custo de $\frac{m}{3}(4 \lceil \log q \rceil + 2 \lceil \log m \rceil)$ bits.

A Tabela 5.5 apresenta custos de comunicação em função de alguns valores adotados para os parâmetros do protocolo, considerando uma execução completa, com diversas rodadas. Em todos os casos, foi adotado $t = 137$ rodadas, conforme determinado na Seção 5.1.1 para que o erro de coerência fosse igual ou inferior a 2^{-80}.

		$n = 64$ $m = 2048$ $q = 257$ $t = 137$	$n = 128$ $m = 2048$ $q = 257$ $t = 137$	$n = 64$ $m = 4096$ $q = 257$ $t = 137$	$n = 128$ $m = 4096$ $q = 257$ $t = 137$
Valor de Compromisso	$t\, 3n \lceil \log q \rceil$	28.8 KB	57.8 KB	28.8 KB	57.8 KB
Resposta ao Desafio	$t \frac{m}{3}(4 \lceil \log q \rceil + 2 \lceil \log m \rceil)$	662.2 KB	662.2 KB	1.34 MB	1.34 MB

Tabela 5.5: Custos de comunicação do protocolo de Kawachi et al.

A interação no protocolo de Cayrel et al. se inicia com o envio de dois valores de compromisso com a mesma dimensão do protocolo de Kawachi et al. Isto é, vetores que representam, cada um, um custo de comunicação de $n \lceil \log q \rceil$ bits. Em seguida, o verificador envia α, um inteiro módulo q, e recebe β, um vetor de dimensão m e elementos módulo q. Assim, α e β representam em conjunto um custo de comunicação de $(m + 1) \lceil \log q \rceil$ bits. No total, toda essa etapa inicial, incluindo os valores de compromisso, apresenta um custo de $(2n + m + 1) \lceil \log q \rceil$ bits por rodada.

Para este protocolo, o desafio também pode ser considerado desprezível, representando um bit por rodada. A resposta, porém, pode ser um vetor de dimensão m e elementos módulo q mais a representação de uma permutação ($\mathbf{r}_0$ e σ, se $b = 0$), ou dois vetores

também com as características mencionadas ($\mathbf{r}_1$ e $\mathbf{P}_\sigma\mathbf{x}$, se $b = 0$). Portanto, o custo por rodada pode ser $m\lceil\log q\rceil + m\lceil\log m\rceil$ bits ou $2m\lceil\log q\rceil$ bits, dependendo do desafio recebido. Considerando uma distribuição uniforme para o valor do desafio, temos um custo médio por rodada de $\frac{m}{2}(3\lceil\log q\rceil + \lceil\log m\rceil)$ bits.

Os custos de comunicação do protocolo de Cayrel et al. considerando parâmetros reais estão apresentados na Tabela 5.6. Para manter o erro de coerência também igual ou inferior a 2^{-80}, foram utilizadas 80 rodadas do protocolo interativo.

		$n = 64$ $m = 2048$ $q = 257$ $t = 80$	$n = 128$ $m = 2048$ $q = 257$ $t = 80$	$n = 64$ $m = 4096$ $q = 257$ $t = 80$	$n = 128$ $m = 4096$ $q = 257$ $t = 80$
Etapa inicial	$t\,(2n + m + 1)\lceil\log q\rceil$	191.3 KB	202.6 KB	371.3 KB	382.6 KB
Resposta ao Desafio	$t\,\frac{m}{2}(3\lceil\log q\rceil + \lceil\log m\rceil)$	380 KB	380 KB	780 MB	780 MB

Tabela 5.6: Custos de comunicação do protocolo de Cayrel et al.

Por fim, no protocolo de Lyubashevsky, três valores são trocados: $\mathbf{Y}$, o desafio c e a resposta $\hat{\mathbf{z}}$. O primeiro deles, $\mathbf{Y}$, é obtido a partir da aplicação da função h, assim como a chave pública $\mathbf{S}$. Dessa forma, corresponde a um polinômio cuja representação exige $n\lceil\log p\rceil$ bits.

O desafio c compreende um polinômio de R, cuja norma de Manhattan é inferior a κ. Existem diversas maneiras através das quais os coeficientes de c poderiam ser transmitidos. Por exemplo, uma possibilidade consiste em enviar todos os coeficientes em blocos de tamanho fixo, independente do seu valor real. Outra alternativa seria transmitir apenas os coeficientes diferentes de zero, acompanhados do expoente dos respectivos termos, como forma de identificação. Neste último caso, a quantidade total de bits envolvidos na comunicação não seria constante para todos os possíveis valores de c. Por simplicidade, iremos considerar que são enviados n coeficientes, onde cada um pode ter valor de zero até κ, já que não buscamos nenhuma compressão de dados para os demais valores trocados. Assim, a transmissão de c envolve $n\lceil\log\kappa\rceil$ bits.

Para que o protocolo não seja abortado, a resposta $\hat{\mathbf{z}}$ deve pertencer a G^m, isto é, $\hat{\mathbf{z}}$ deve ser um conjunto de m polinômios pertencentes a R, nos quais todos os coeficientes devem ter módulo igual ou inferior a $mn\sigma\kappa - \sigma\kappa$. Portanto, neste caso, realiza-se a transmissão de $mn\lceil\log(2mn\sigma\kappa - 2\sigma\kappa + 1)\rceil$ bits. É razoável supor que utilizamos um único bit para indicar se o verificador se recusa a responder o desafio. Assim, se o protocolo é abortado, podemos considerar que $\hat{\mathbf{z}}$ corresponde a um bit e é desprezível.

Devido à possibilidade do protocolo abortar, é mais difícil determinar a quantidade média de dados transmitidos. Na realidade, esse valor irá depender também de como este caso é tratado e de quantas rodadas serão realizadas. Por exemplo, repetindo-se o protocolo um número fixo de vezes tem-se custos de comunicação razoavelmente constantes[1]. Por outro lado, caso sejam enviados desafios até que o demonstrador consiga não abortar, os custos serão representados pelos inúmeros $\mathbf{Y}$ e desafios transmitidos, e mais uma única resposta. Outra possibilidade é enviar uma quantidade fixa de desafios, porém recebendo apenas uma resposta (válida). Por esse motivo, a Tabela 5.7 apresenta os custos de comunicação para uma única repetição do protocolo, considerando que o mesmo não é abortado.

			$n = 512$ $m = 4$ $\sigma = 127$ $\kappa = 24$ $p = 2^{31.7}$	$n = 512$ $m = 5$ $\sigma = 2047$ $\kappa = 24$ $p = 2^{59.8}$	$n = 512$ $m = 8$ $\sigma = 2047$ $\kappa = 24$ $p = 2^{95.8}$	$n = 1024$ $m = 8$ $\sigma = 2047$ $\kappa = 21$ $p = 2^{95.9}$
Etapa inicial	$\mathbf{Y}$	$n\lceil\log p\rceil$	2 KB	3.75 KB	6 KB	12 KB
	c	$n\lceil\log\kappa\rceil$	320 B	320 B	320 B	640 B
Resposta ao Desafio	$\hat{\mathbf{z}}$	$mn\lceil\log(2mn\sigma\kappa - 2\sigma\kappa + 1)\rceil$	6 KB	8.75 KB	14.5 KB	30 KB

Tabela 5.7: Custos de comunicação do protocolo de Lyubashevsky.

[1]Pois a fração de interações abortadas é sempre aproximadamente a mesma.

5.1.4 Possibilidades de Paralelização

Em princípio, diversas operações dos algoritmos abordados são relativamente independentes. Ou seja, embora certos cálculos dependam do recebimento de alguns valores, existem conjuntos de passos que poderiam ser executados simultaneamente, especialmente quando consideramos diversas rodadas ou repetições. Assim, caso os protocolos fossem executados em dispositivos que permitam concorrência, uma estratégia natural para paralelizar as operações, seria realizar diversas rodadas em simultâneo.

Contudo, em se tratando de protocolos criptográficos, o método de paralelização empregado não deve interferir nas propriedades de segurança do algoritmo. Como visto na Seção 3.1.2, certos protocolos apresentam a propriedade de conhecimento zero, no qual o demonstrador consegue provar sua identidade sem revelar nenhuma informação adicional. Nestes casos, as distribuições das saídas obtidas a partir da interação com um algoritmo que simule o comportamento do demonstrador deve ser indistinguível das produzidas na interação com um demonstrador legítimo.

Num cenário concorrente, construir protocolos que mantenham essa propriedade pode ser mais desafiador. Um protocolo que apresente prova de conhecimento zero para execução serial, não necessariamente permanecerá igualmente seguro quando diversas demonstrações de identidade[2] foram realizadas em simultâneo.

A indistinguibilidade de evidência é uma variante do conhecimento zero, com condições mais fracas. Os protocolos que apresentam esta propriedade asseguram apenas que o verificador não conseguirá distinguir entre dois demonstradores que usem evidências distintas. Porém, não há garantias que outras informações não serão expostas, como o conjunto de evidências possíveis. Em particular, a própria evidência utilizada pode acabar sendo revelada, caso exista apenas uma evidência possível. Por outro lado, embora a condição de conhecimento zero seja enfraquecida, a indistinguibilidade de evidência é mantida quando múltiplas demonstrações forem realizadas em paralelo.

Felizmente, todo protocolo de conhecimento zero apresenta a propriedade de indistinguibilidade de evidência [17]. Assim, embora a realização de repetições em paralelo provavelmente ocasione a perda da condição de conhecimento zero [24], ainda haverá

[2]Isto é, rodadas ou repetições do protocolo.

garantias providas pela indistinguibilidade de evidência.

O protocolo de Lyubashevsky apresenta indistinguibilidade de evidência, e portanto suas repetições podem ser paralelizadas sem prejuízo à segurança. Já os protocolos de Kawachi et al. e Cayrel et al. são de conhecimento zero. Assim, também podem ter suas rodadas executadas de forma concorrente, porém com uma possível redução nas garantias fornecidas. No caso, os autores destes dois protocolos não demonstram, ou afirmam, que a propriedade de conhecimento zero será mantida mesmo com paralelização das rodadas. No entanto, por conservarem a indistinguibilidade de evidência, ambos os algoritmos equiparam-se ao protocolo de Lyubashevsky, no cenário concorrente, e apresentam vantagem na execução sequencial.

Outra possibilidade de paralelização dos algoritmos consiste na realização simultânea das operações de uma mesma rodada. Dessa forma, não há alteração nas saídas produzidas, pois as repetições em si continuam sendo realizadas de modo serial. Assim, a propriedade de conhecimento zero, se presente, é mantida, e ainda consegue-se tirar proveito da capacidade de execução concorrente, quando houver.

Ao optar por essa forma de paralelização, deve-se, portanto, determinar quais operações serão realizadas simultaneamente. A princípio, pode-se considerar como uma operação cada passo do protocolo interativo e tentar efetuar diversos passos ao mesmo tempo. Infelizmente, a maioria dos passos não são totalmente independentes e, portanto, não podem ser executados simultaneamente. Ainda assim, é possível determinar alguns que permitem esse tipo de paralelização funcional, como os cálculos dos valores de compromisso, conforme comentado no Capítulo 4. Contudo, pela estrutura dos protocolos, tais passos são em número reduzido, geralmente inferior à quantidade de *threads* disponíveis, de forma que com essa estratégia nem sempre é possível aproveitar ao máximo os recursos de execução concorrente do processador. Se ainda assim este método de paralelização for empregado, o protocolo de Kawachi et al. é o que apresenta maior número de passos independentes, enquanto que no protocolo de Lyubashevsky todos os passos de uma mesma entidade (demonstrador ou verificador) devem ser executados de maneira sequencial.

Aparentemente, paralelizar os cálculos efetuados em um único passo compreende uma alternativa interessante. Também neste caso, as repetições do protocolo interativo são realizadas sequencialmente, e o conhecimento zero, quando presente, não é afetado. De modo geral, o cálculo envolvendo matrizes e vetores, as operações com polinômios, e a

grande maioria da seleção aleatória de elementos podem ser facilmente distribuídos em *threads*, de forma que os três protocolos são igualmente beneficiados com esta técnica.

5.1.5 Modelos de Segurança

Ambos os protocolos de Kawachi et al. e Cayrel et al. têm sua segurança baseada na hipótese de que resolver o SIS compreende uma tarefa difícil. Além disso, supõem o uso de uma função de compromisso que atenda as seguintes condições:

1. Nenhum adversário consegue distinguir entre dois compromissos gerados a partir de duas entradas distintas.

2. Após enviar o valor de compromisso, um adversário com poder de computação polinomial não consegue modificar o vetor de entrada e manter a relação entre essas duas informações consistentes.

A partir dessas suposições, os autores conseguem demonstrar a segurança dos respectivos protocolos no modelo de ataque ativo concorrente, bem como a propriedade de conhecimento zero estatístico. Como visto, a indistinguibilidade de evidência é uma variante mais fraca do conhecimento zero, e os protocolos que apresentam esta última propriedade, também desfrutam da primeira.

Já o protocolo de Lyubashevsky apresenta somente a propriedade de indistinguibilidade de evidência e é seguro contra ataques ativos. Outra diferença fundamental diz respeito às suas hipóteses de segurança. Basicamente, o protocolo assume fatores de aproximação maiores, resultando em suposições mais fortes, e tornando necessário o uso de parâmetros também maiores.

Ainda em relação aos parâmetros empregados, Cayrel et al. e Lyubashevsky, ao fornecerem valores concretos a serem empregados com seus protocolos, também comparam suas escolhas com os melhores ataques atualmente conhecidos. No caso, para os parâmetros sugeridos, os autores listam dados que comprovam que os algoritmos são seguros contra estes ataques. Conforme já mencionado, Kawachi et al. não apresentam parâmetros concretos para seu protocolo.

5.1.6 Conversão para Protocolos de Assinatura

Em princípio, todos os três protocolos podem ser convertidos em esquemas de assinatura, através da Transformação de Fiat-Shamir. Entretanto, apenas Lyubashevsky apresenta o protocolo de assinatura derivado de seu algoritmo [43].

Conforme observado na Seção 5.1.3, no caso dos protocolos de Cayrel et al. e Kawachi et al., embora os desafios sejam representados, respectivamente, por um ou dois bits, a resposta enviada pelo demonstrador corresponde a uma quantidade bem superior de dados. Dessa forma, quando se utiliza a Heurística de Fiat-Shamir, o tamanho da assinatura produzida também é afetado por essa relação.

Neste sentido, o protocolo de Lyubashevsky leva vantagem na conversão para esquema de assinatura, por empregar desafios maiores. Além disso, a possibilidade do protocolo abortar não traz grandes prejuízos ao desempenho do algoritmo, bem como nenhuma alteração no tamanho da assinatura gerada, como já apontado na Seção 3.3.4.

5.2 Discussão Sobre Otimização

Muitos algoritmos, quando utilizados em sistemas computacionais reais, necessitam de certa otimização para que possam ser empregados, devido ao número limitado de recursos para a sua execução. Além disso, mesmo a otimização não sendo estritamente necessária, em quase todas as situações é possível se beneficiar dos ganhos advindos deste processo.

Frequentemente, por otimizar, ou tornar ótimo, subentende-se atingir o maior conceito possível em relação a um determinado critério ou conjunto de critérios. Isto é, idealmente, otimizar significa alcançar um grau máximo de aperfeiçoamento, a partir do qual não é possível aplicar mais melhorias. Entretanto, em muitos casos, essa situação ideal não pode ser medida ou comprovada, de forma que o conceito de otimização refere-se a qualquer modificação que gere resultados superiores.

No caso dos protocolos de identificação abordados, busca-se essencialmente melhorar o desempenho dos algoritmos, em termos de tempo de execução, e aperfeiçoar o aproveitamento de memória. O grau de segurança provido pelo protocolo, como visto, em geral pode ser ajustado a partir de certos parâmetros, porém usualmente com algum prejuízo à eficiência de processamento e utilização da memória.

Quando examinamos simultaneamente o tempo de execução e uso de memória de

dois programas, estamos fundamentalmente analisando duas implementações, ainda que o nosso interesse primordial seja o de comparar dois algoritmos. Isto é, considerando que as implementações podem ser resultado de esforços de otimização diferentes, as variações nos seus desempenhos não necessariamente refletirão de maneira exata as diferenças dos algoritmos nos quais se baseiam. Por exemplo, considere que um programador P_A implementa um algoritmo A para resolver um determinado problema, e um programador P_B implementa um algoritmo B para solucionar a mesma tarefa. Caso o programa de P_A seja mais rápido que o programa de P_B, não necessariamente o algoritmo A será pior do que o algoritmo B. Talvez simplesmente P_A seja um programador melhor que P_B, e as otimizações aplicadas em sua implementação sejam mais eficientes. Ou então, pode ter ocorrido de P_A ter se esforçado mais em otimizar seu programa e obtido um resultado superior.

De qualquer maneira, existe uma certa dificuldade em comparar dois algoritmos em um cenário no qual podem existir critérios de avaliação que dependam das implementações realizadas, como é o caso do tempo de execução. Por esse motivo, neste trabalho optou-se por produzir versões de implementação que não contivessem, ao menos intencionalmente, nenhum tipo de otimização nos protocolos implementados. Por exemplo, os métodos de multiplicação de matriz por vetor e cálculo da FFT foram desenvolvidos de acordo com os algoritmos tradicionais, encontrados na maioria dos livros didáticos.

Para tornar a comparação também mais justa entre as implementações que utilizassem alguma biblioteca adicional, preferiu-se empregar uma mesma biblioteca em todos esses casos. Contudo, em alguns exemplos, essa solução pode ser potencialmente falha, já que não há garantias de que os esforços de otimização se deram de maneira uniforme no desenvolvimento de toda a biblioteca. Observamos, ainda, a influência do uso de heurísticas, conforme discutido no Capítulo 4. Ainda assim, consideramos que o fato das implementações utilizarem uma única biblioteca permite uma comparação suficientemente confiável.

O uso de reticulados ideais também pode ser visto como uma tentativa de alcançar um melhor desempenho. Neste caso, trata-se de uma modificação conceitual no algoritmo, e que, portanto, é propagada para todas as implementações nele baseadas. Assim, podemos dizer que trata-se de uma otimização *independente da plataforma* de desenvolvimento e execução utilizada.

A grande vantagem deste tipo de melhoria é, evidentemente, sua facilidade de aplicação, nos mais diversos cenários, em função justamente da independência de plataforma. Por outro lado, esta classe de otimizações deixa de explorar características particulares do ambiente no qual será realizada a execução. Desta forma, na prática, otimizações mais específicas para uma determinada plataforma também são extremamente vantajosas, ainda que seus resultados não possam ser diretamente reproduzidos em infraestruturas computacionais diferentes.

Não se pode afirmar que um tipo de otimização seja mais fácil de ser obtido do que outro, apenas que ambos apresentam as suas vantagens e importância, devendo ser empregados de maneira complementar. Além de, em geral, poderem ser aplicadas de modo mais simples, as otimizações mais estruturais ou independentes de plataforma permitem uma comparação razoavelmente imparcial entre dois algoritmos. Contudo, a busca por melhorias dessa natureza pode se esgotar mais rapidamente, no sentido de que nem sempre o protocolo permite um grande número de alterações, antes de perder suas características primordiais. Por outro lado, casos as modificações venham a produzir novos algoritmos, ainda que bastante semelhantes ao original, este resultado pode ser encarado como mais uma contribuição para o progresso da área de Criptografia.

Estando determinado o cenário no qual o protocolo será utilizado na prática, abre-se espaço para as otimizações mais específicas para uma determinada plataforma. Neste caso, pode-se, por exemplo, utilizar instruções particulares de um certo processador, ou preocupar-se com questões de nível mais baixo. É interessante que, neste momento, também sejam aplicadas as otimizações mais estruturais e independentes de plataforma, visto que para o uso em aplicativos reais, deseja-se que os protocolos estejam otimizados ao máximo. De fato, podemos dizer que o objetivo final da otimização de sistemas computacionais é permitir os melhores resultados e desempenho na prática, e não apenas em aplicações hipotéticas.

A busca por maior eficiência na realização de tarefas, sejam estas computacionais ou de uma forma geral, frequentemente leva à realização de operações simultâneas. Dessa forma, adicionar paralelismo a sistemas é uma maneira usual de tentar melhorar o seu desempenho. Atualmente, mesmo os dispositivos computacionais mais simples já permitem a execução de algoritmos concorrentes, de forma que podemos considerar essa forma de otimização razoavelmente independente de plataforma.

Conforme visto na Seção 5.1.4, existem diversas maneiras de paralelizar os protocolos abordados e algumas delas acabam por afetar a segurança do algoritmo, em particular no que se refere à propriedade de conhecimento zero. Além disso, as diferentes formas de paralelização resultam em ganhos também distintos, conforme verificado nos dados de execução da Seção 4.4.

Muitas vezes, não é possível paralelizar o algoritmo como um todo, apenas certas porções, como foi o caso, inclusive, das implementações do Capítulo 4. Na realidade, a mesma observação vale para qualquer forma de otimização. Uma maneira de estimar a melhoria máxima para o sistema como um todo, quando apenas uma porção é otimizada, é aplicar a lei de Amdahl. Assim, supondo que um determinado programa executa originalmente em tempo T, se uma fração P for otimizada e tiver uma melhoria representada por um fator M (por exemplo, se conseguimos fazer com que 10% do programa execute duas vezes mais rápido, então $P = 0.1$ e $M = 2$), a lei de Amdahl afirma que o tempo total de execução será melhorado para no máximo $\frac{1}{(1-P)+\frac{P}{M}}T$. Em termos de execução simultânea, frequentemente substitui-se M pelo número de processadores disponíveis. Entretanto, sabemos que algumas técnicas de paralelização podem introduzir custos adicionais ao desempenho, de forma que a melhoria máxima esperada acaba não sendo atingida.

Em relação ao aproveitamento de memória, nas implementações realizadas neste trabalho, não foi aplicada nenhuma otimização neste sentido, exceto as oriundas do uso de reticulados ideais. Particularmente, diversos elementos empregados nos protocolos podem ser representados com um número de bits cujo total não corresponde a uma quantidade inteira de bytes. Por exemplo, se um dado valor pode ser representado com nove bits, mas é reservada uma variável de dois bytes para seu armazenamento, sete bits acabam nunca sendo aproveitados. Dessa forma, para realizar o armazenamento de valores de modo a não resultar no desperdício de bits, é necessária a realização de certa manipulação dos dados. Contudo, estas operações podem encarecer os custos em termos de tempo de execução, de forma a serem justificadas apenas quando a quantidade de memória disponível é um fator limitante e não há otimizações mais eficazes, como o uso de reticulados ideais.

5.3 Trabalhos Futuros

Neste trabalho, foram exploradas apenas otimizações independentes de plataforma, particularmente o uso de reticulados e paralelização de código. Conforme mencionado na seção anterior, este tipo de modificação apresenta a vantagem de poder ser aplicada nos mais diversos ambientes. Existem ainda outras otimizações nesta categoria das quais se pode tirar proveito. Por exemplo, como já comentado, o algoritmo da FFT admite inúmeras melhorias, especialmente em um cenário de execução concorrente. A princípio, estas otimizações foram evitadas, para efeito de comparação dos protocolos, conforme exposto na Seção 5.2.

Além disso, os resultados experimentais revelaram que a utilização de reticulados ideais pode resultar em protocolos mais eficientes, especialmente quando as dimensões dos reticulados e, portanto, das estruturas envolvidas, são relativamente grandes. Contudo, para dimensões menores, pode haver perda de desempenho, dependendo da maneira como o protocolo foi implementado. Os esforços de otimização podem reverter essa última situação e tornar as implementações com reticulados ideais mais rápidas do que as que empregam reticulados genéricos, para qualquer dimensão.

Desta forma, uma possível extensão deste trabalho consiste em buscar diversas formas de otimização para as operações de multiplicação de matriz por vetor ou multiplicação de polinômios, que constituem a tarefa mais custosa na execução dos protocolos. A implementação de funções de *hash* SWIFFTX [44], por exemplo, apresenta diversas técnicas para tornar operações como estas mais eficientes, nos casos em que o vetor de entrada é binário. Os métodos envolvem desde a escolha cuidadosa do valor de certos parâmetros até a utilização de tabelas pré-calculadas para agilizar a execução. Neste caso, alguns elementos precisam ter seu valor pré-determinado, da mesma forma que Kawachi et al. e Cayrel et al. sugerem que a matriz pública $\mathbf{A}$ seja fixa para todos os demonstradores, de modo a poder ser incorporada ao código.

Outra possibilidade de extensão deste trabalho diz respeito a investigar outras formas de melhorias estruturais ou independentes de plataforma. Nestes casos, é interessante realizar uma validação experimental para comprovar que os ganhos previstos irão de fato ocorrer, ou determinar a partir de qual ponto as modificações são mais proveitosas (por exemplo, a partir de qual dimensão das estruturas).

Otimizações mais específicas para uma determinada plataforma também podem ser

examinadas, ainda que momentaneamente não exista intenção de desenvolver uma aplicação real[3] que utilize algum dos protocolos. Desta maneira, será possível se aproximar ao máximo dos limites de otimização que os algoritmos permitem e compará-los com outros protocolos de identificação atualmente empregados na mesma plataforma escolhida.

Neste sentido, pode-se considerar o desenvolvimento para dispositivos relativamente mais restritos, como *tablets* e celulares, nos quais os métodos de autenticação de entidade encontram diversas aplicações. Equipamentos mais simples, como diversos sistemas embarcados, também podem encontrar utilidade nos protocolos de identificação. Neste caso, em muitas situações, não apenas o usuário deve ser identificado, mas também o próprio aparelho pode necessitar de alguma validação, especialmente quando há comunicação com outros equipamentos. Sendo assim, dadas as capacidades de processamento, memória e armazenamento usualmente mais limitadas destes dispositivos, os métodos de otimização assumem grande importância na elaboração do aplicativo final.

No caso de computadores pessoais, pode-se tentar explorar ao máximo a capacidade de processamento, aproveitando as peculiaridades da arquitetura do processador utilizado. Havendo uma GPU dedicada e com grande capacidade de processamento, esta também pode ser engajada na execução do algoritmo. Atualmente, o avanço no desenvolvimento de GPUs, especialmente no sentido de expandir as possibilidades de computação paralela, tem gerado inúmeros trabalhos apontando o uso dessas unidades em diversas aplicações [33], o que poderia incluir também os protocolos abordados.

Independente do tipo de otimização, o uso de bibliotecas pode auxiliar no desenvolvimento de implementações mais eficientes. Por exemplo, o MATLAB apresenta pacotes específicos para computação paralela e suporte a GPUs [27, 56]. Além disso, atualmente a maioria das bibliotecas para operações matemáticas provê funções otimizadas para o cálculo da FFT. Em especial, no caso dos protocolos considerados, poderia ser utilizada uma biblioteca como a Apfloat [5], um pacote de alto desempenho para aritmética de precisão arbitrária, disponível para C++ e Java, que implementa a NTT, a transformação da Teoria dos Números equivalente a FFT que não emprega números complexos.

Outra interessante biblioteca disponível é a FFTW [18], conhecida como a mais eficiente

[3]Isto é, uma aplicação cuja finalidade não se resuma à realização de testes, mas que seja de fato projetada para algum uso prático.

implementação livre em software para cálculo da Transformada Rápida de Fourier. Tal posição é confirmada pelo *benchmark* benchFFT, e outra vantagem desta biblioteca é a de possuir versões já paralelizadas para realizar as operações, suportando tanto sistemas de memória compartilhada quanto distribuída. Considerando que o uso de bibliotecas de certa forma restringe a liberdade para modificar o modo como as operações são realizadas, o fato de o recurso de paralelização já estar disponível é bastante conveniente.

Capítulo 6

Considerações Finais

A necessidade de autenticação de entidades em diversas aplicações práticas e atividades cotidianas torna os protocolos criptográficos de identificação um meio interessante de prover um controle seguro de acesso a informações, recursos e serviços, entre outros. O uso de reticulados para essa finalidade parece promissor, tendo em vista sua provável resistência aos futuros ataques que empreguem computadores quânticos. Dessa maneira, é importante continuar a estudá-los e verificar como estes podem ser melhor explorados em situações práticas, seja através do desenvolvimento de novos algoritmos ou da tentativa de aprimorar os algoritmos já existentes.

Neste trabalho, as principais propostas recentes para protocolos de identificação baseados em reticulados foram reunidas e analisadas, tanto a partir de informações e comentários feitos pelos próprios autores, quanto do exame das estruturas dos algoritmos, ou através da realização de alguns experimentos práticos. Diversos desses protocolos apresentam similaridades em sua construção, decorrentes provavelmente do uso de premissas de segurança semelhantes. Pode-se de certa maneira tomar tal fato como um indicativo de que ainda há muitas possibilidades a serem perscrutadas no que diz respeito ao aproveitamento de reticulados na produção de algoritmos criptográficos. De qualquer modo, a elaboração de propostas de protocolos mais diversificados só viria a beneficiar o desenvolvimento da área. Porém, sabe-se que esta empreitada não é trivial, requerendo não apenas uma visão bastante abrangente, mas também uma capacidade criativa elevada.

O uso de reticulados ideais, em particular, parece oferecer boas perspectivas no que diz respeito a melhoria da eficiência dos protocolos, sem prejuízo a sua segurança. Pode-se

destacar, inclusive, a facilidade de se adaptar algoritmos que empreguem reticulados genéricos para utilizar reticulados ideais, beneficiando-se, portanto, das vantagens destacadas na Seção 2.3.4. Caso futuramente se prove que esta classe de reticulados apresenta alguma vulnerabilidade em particular, algoritmos originalmente projetados para uso com qualquer classe de reticulados poderão continuar a ser empregados normalmente com reticulados genéricos.

Em princípio, pode-se considerar a existência de outras classes de reticulados com propriedades particulares que possam ser aproveitadas em construções criptográficas. Deve-se apenas atentar que as estruturas adicionais podem potencialmente tornar certos problemas envolvendo reticulados mais fáceis de serem resolvidos, facilitando o desenvolvimentos de ataques aos algoritmos. Ainda assim, diversificar a categoria de reticulados empregados pode ser bastante proveitoso, especialmente quando existirem questões em aberto acerca das classes atualmente utilizadas. Dessa forma, caso em momento posterior seja demonstrado que um determinado conjunto de reticulados não é adequado para o desenvolvimento de algoritmos criptográficos, ainda existirá uma certa variedade de opções a qual se poderá recorrer.

Aparentemente, a subárea da Criptografia Pós-Quântica que aborda protocolos de identificação, ou mesmos criptossistemas de uma forma geral, baseados em reticulados, caminha justamente na direção de tentar prover algoritmos mais eficientes, o que possivelmente pode ser melhor alcançado aproveitando essas estruturas particulares de certos reticulados. Além dos protocolos aqui apresentados que recorrem aos reticulados ideais, o NTRU [31] é outro exemplo de sistema que faz uso de reticulados específicos.

Considerando ainda a preocupação com o desempenho do algoritmo, é interessante que, para todos os protocolos desenvolvidos, uma vez determinados os conjuntos de parâmetros que asseguram um certo nível de segurança, seja verificado se os mesmos são adequados para aplicação na prática. Além das previsões teóricas, experimentos de implementação também podem auxiliar no processo de validar ou contestar o protocolo como um bom candidato para uso em aplicações reais. Durante esta etapa, inclusive, é possível que surjam dúvidas e questionamentos não previstos anteriormente. Ademais, as tentativas de otimizar os algoritmos durante a fase de implementação podem resultar em técnicas aplicáveis não somente àquele sistema em questão, mas também a diversos outros algoritmos de mesma natureza ou com construção semelhante.

Glossário

Tradução para o português	Termo em inglês	Tradução alternativa
coerência	*soundness*	corretude
coerência perfeita	*perfect soundness*	corretude perfeita
erro de coerência	*soundness error*	erro de corretude
completude	*completeness*	integridade
completude perfeita	*perfect completeness*	integridade perfeita
erro de completude	*completeness error*	erro de integridade
(valor de) compromisso	*commitment*	
função de compromisso	*commitment function*	
conhecimento zero	*zero knowledge*	conhecimento nulo
demonstrador	*prover*	provador
(reticulado de) dimensão completa	*full-dimensional (lattice)*	
indistinguibilidade de evidência	*witness indistinguishability*	indistinguibilidade de testemunha
evidência	*witness*	testemunha
NTT	*Number-Theoretic Transform*	
verificador	*verifier*	

Notações

Reticulados

Símbolo	Significado
$L(B)$	Reticulado gerado pela base $B = \{\mathbf{v}_1, ..., \mathbf{v}_n\}$: $L(B) = \{a_1\mathbf{v}_1 + ... + a_n\mathbf{v}_n : a_1, ..., a_n \in \mathbb{Z}\}$.
$L(\mathbf{B})$	Reticulado gerado pelos vetores colunas de $\mathbf{B} \in \mathbb{R}^{m\times n}$: $L(\mathbf{B}) = \{\mathbf{v} \in \mathbb{R}^m : \mathbf{v} = \mathbf{Bu}, \mathbf{u} \in \mathbb{Z}^n\}$.
$L_q^{\perp}(\mathbf{A})$	Reticulado correspondente aos vetores ortogonais módulo q às linhas de $\mathbf{A} \in \mathbb{Z}^{n\times m}$: $L_q^{\perp}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Ax} = 0 \bmod q\}$.
$L_{\mathbf{G}}$	Reticulado cuja matriz de Gram é $\mathbf{G}$.
$L_1 \cong L_2$	Reticulados L_1 e L_2 são isomorfos (possuem a mesma forma normal de Hermite).

Polinômios

Símbolo	Significado
$R[x]$	Polinômios com coeficientes em R.
$F[x]/\langle f(x)\rangle$	Polinômios com coeficientes em F e grau menor do que o grau de $f(x)$.
$p * q$	Convolução dos polinômios p e q.

Normas

Símbolo	Significado
$\|\mathbf{v}\|$	Norma euclidiana de $\mathbf{v} = (v_1, ..., v_n)$: $\|\mathbf{v}\| = \sqrt{v_1^2 + ... + v_n^2}$.
$\|\mathbf{v}\|_\infty$	Norma uniforme de $\mathbf{v} = (v_1, ..., v_n)$: $\|\mathbf{v}\|_\infty = \max\{\|v_1\|, ..., \|v_n\|\}$.
$\|\mathbf{v}\|_1$	Norma de Manhattan de $\mathbf{v} = (v_1, ..., v_n)$: $\|\mathbf{v}\|_1 = \sum_{i=1}^{n} \|v_i\|$.

Operações de Protocolos

Símbolo	Significado
$a \xleftarrow{\$} A$	Seleção aleatória de a a partir do conjunto A.
$\mathbf{a} \parallel \mathbf{b}$	Concatenação dos vetores $\mathbf{a}$ e $\mathbf{b}$.
$\sigma(\mathbf{x})$	Aplicação da permutação σ sobre o vetor $\mathbf{x}$.
P_σ	Matriz de permutação $m \times m$ que representa a permutação σ.

Referências Bibliográficas

[1] Ajtai, M. Generating Hard Instances of Lattice Problems (Extended Abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1996), STOC '96, ACM, pp. 99–108.

[2] Ajtai, M., and Dwork, C. A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1997), STOC '97, ACM, pp. 284–293.

[3] Ajtai, M., Kumar, R., and Sivakumar, D. A Sieve Algorithm For the Shortest Lattice Vector Problem. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2001), STOC '01, ACM, pp. 601–610.

[4] Ajtai, Miklós. The Shortest Vector Problem in L2 is NP-Hard for Randomized Reductions (Extended Abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1998), STOC '98, ACM, pp. 10–19.

[5] Apfloat. http://www.apfloat.org.

[6] Babai, L. On Lovász' Lattice Reduction and the Nearest Lattice Point Problem (Shortened Version). In *STACS '85: Proceedings of the 2nd Symposium of Theoretical Aspects of Computer Science* (London, UK, 1985), Springer-Verlag, pp. 13–20.

[7] Blömer, J., and Seifert, J.-P. On the Complexity of Computing Short Linearly Independent Vectors and Short Bases in a Lattice. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1999), STOC '99, ACM, pp. 711–720.

[8] Bracewell, R. *The Fourier Transform and Its Applications* , 3 ed. McGraw-Hill, New York, 1999, ch. Convolution Theorem, pp. 108–112.

[9] Bradley, G. H. Algorithms for Hermite and Smith Normal Matrices and Linear Diophantine Equations. *Mathematics of Computation 25*, 116 (1971), 897–907.

[10] Brigham, E. O. *The Fast Fourier Transform and Its Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[11] Cayrel, P.-L., Lindner, R., Rückert, M., and Silva, R. Improved Zero-Knowledge Identification with Lattices. In *Proceedings of the 4th International Conference on Provable Security* (Berlin, Heidelberg, 2010), ProvSec'10, Springer-Verlag, pp. 1–17.

[12] Cohen, H. *A Course in Computational Algebraic Number Theory*, 3 ed. Springer, 1993.

[13] Cohen, H. *Theory of Linear and Integer Programming*, 3 ed. Springer, 1993.

[14] Coppersmith, D., and Winograd, S. Matrix Multiplication via Arithmetic Progressions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1987), STOC '87, ACM, pp. 1–6.

[15] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009, ch. Polynomials and the FFT.

[16] Fang, X. G., and Havas, G. On The Worst-case Complexity of Integer Gaussian Elimination. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation* (New York, NY, USA, 1997), ISSAC '97, ACM, pp. 28–31.

[17] Feige, U., and Shamir, A. Witness Indistinguishable and Witness Hiding Protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1990), STOC '90, ACM, pp. 416–426.

[18] FFTW: Fastest Fourier Transform in the West. http://www.fftw.org.

[19] Fiat, A., and Shamir, A. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proceedings on Advances in Cryptology—CRYPTO '86* (London, UK, 1987), Springer-Verlag, pp. 186–194.

[20] Gauss, C. F. *Disquisitiones Arithmeticae*. 1798.

[21] GCC, the GNU Compiler Collection. http://gcc.gnu.org.

[22] Gentry, C., Peikert, C., and Vaikuntanathan, V. Trapdoors for Hard Lattices and New Cryptographic Constructions. In *Proceedings of the 40th annual ACM symposium on Theory of computing* (New York, NY, USA, 2008), STOC '08, ACM, pp. 197–206.

[23] Goldreich, O., Goldwasser, S., and Halevi, S. Public-Key Cryptosystems from Lattice Reduction Problems. In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology* (1996), Springer-Verlag, pp. 112–131.

[24] Goldreich, O., and Krawczyk, H. On the Composition of Zero-Knowledge Proof Systems. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming* (London, UK, UK, 1990), ICALP '90, Springer-Verlag, pp. 268–282.

[25] Goldreich, O., Micciancio, D., Safra, S., and Seifert, J.-P. Approximating Shortest Lattice Vectors is not Harder than Approximating Closet Lattice Vectors. *Inf. Process. Lett. 71* (July 1999), 55–61.

[26] GOMP, An OpenMP implementation for GCC. http://gcc.gnu.org/projects/gomp/.

[27] GPUmat: GPU toolbox for MATLAB. http://gp-you.org/.

[28] Graham, S. L., Kessler, P. B., and Mckusick, M. K. gprof: a Call Graph Execution Profiler. *SIGPLAN Not. 17* (June 1982), 120–126.

[29] Hankerson, D., Menezes, A. J., and Vanstone, S. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[30] Hermite, C. Sur l'Introduction des Variables Continues dans la Théorie des Nombres. vol. 1851, pp. 191–216.

[31] Hoffstein, J., Pipher, J., and Silverman, J. H. NTRU: A Ring-Based Public Key Cryptosystem. In *Lecture Notes in Computer Science* (1998), Springer-Verlag, pp. 267–288.

[32] Hoffstein, J., Pipher, J., and Silverman, J. H. NSS: An NTRU Lattice-Based Signature Scheme. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology* (London, UK, 2001), EUROCRYPT '01, Springer-Verlag, pp. 211–228.

[33] Hwu, W.-m. W. *GPU Computing Gems Emerald Edition*, 1st ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[34] International Organization for Standardization. ISO/IEC 9798-5:1999 - Information technology – Security techniques – Entity authentication assurance framework, 1999.

[35] Kao, M.-Y., Ed. *Encyclopedia of Algorithms*. Springer, 2008, ch. Shortest Vector Problem (1982; Lenstra, Lenstra, Lovasz).

[36] Karatsuba, A. Complexity of Computations. In *Proceedings of Steklov Math. Institute* (1995), no. 211.

[37] Karatsuba, A., and Ofman, Y. Multiplication of Many-Digital Numbers by Automatic Computers. In *Proceedings of the USSR Academy of Sciences* (1962), no. 145.

[38] Kawachi, A., Tanaka, K., and Xagawa, K. Concurrently Secure Identification Schemes Based on the Worst-Case Hardness of Lattice Problems. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (Berlin, Heidelberg, 2008), ASIACRYPT '08, Springer-Verlag, pp. 372–389.

[39] Khot, S. Hardness of Approximating the Shortest Vector Problem in Lattices. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 126–135.

[40] Lagrange, J. L. *Recherches d'Arithmétique*. 1775.

[41] Lenstra, A., Lenstra, H., and Lovász, L. Factoring Polynomials with Rational Coefficients. *Math. Ann. 261* (1982), 515–534.

[42] Lyubashevsky, V. Lattice-Based Identification Schemes Secure Under Active Attacks. In *Proceedings of the Practice and Theory in Public Key Cryptography, 11th International Conference on Public Key Cryptography* (Berlin, Heidelberg, 2008), PKC'08, Springer-Verlag, pp. 162–179.

[43] Lyubashevsky, V. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (Berlin, Heidelberg, 2009), ASIACRYPT '09, Springer-Verlag, pp. 598–616.

[44] Lyubashevsky, V., Micciancio, D., Peikert, C., and Rosen, A. SWIFFT: A Modest Proposal for FFT Hashing. In *FSE* (2008), pp. 54–72.

[45] McEliece, R. A Public-Key Cryptosystem Based on Algebraic Coding Theory. DSN Progress Report, 1978.

[46] Menezes, A. J., Oorschot, P. C. V., Vanstone, S. A., and Rivest, R. L. *Handbook of Applied Cryptography*, 1 ed. CRC Press, 1996.

[47] Merkle, R. C. A Certified Digital Signature. In *Advances in Cryptology – CRYPTO '89 Proceedings* (1990), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 218–238.

[48] Micciancio, D. Efficient Reductions among Lattice Problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2008), SODA '08, Society for Industrial and Applied Mathematics, pp. 84–93.

[49] Micciancio, D., and Regev, O. Worst-Case to Average-Case Reductions Based on Gaussian Measures. *SIAM J. Comput. 37* (April 2007), 267–302.

[50] Micciancio, D., and Regev, O. *Lattice-based Cryptography.* Springer, 2009.

[51] Micciancio, D., and Vadhan, S. Statistical Zero-Knowledge Proofs with Efficient Provers: Lattice Problems and More. In *Advances in cryptology - CRYPTO 2003, proceedings of the 23rd annual international cryptology conference* (Santa Barbara,

California, USA, Aug. 2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 282–298.

[52] Nguyen, K. An Identification Scheme from Lattice Distinguishing Problem. In *Applied Cryptography and Network Security* (2003), ACNS '03, ICISA Press, pp. 109–119.

[53] Nguyen, P. Q., and Regev, O. Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures. *Journal of Cryptology 22* (April 2009), 139–160.

[54] NTL: A Library for doing Number Theory. http://www.shoup.net/ntl.

[55] OpenMP. http://openmp.org.

[56] Parallel Computing Toolbox - MATLAB. http://www.mathworks.com/products/parallel-computing/.

[57] Rivest, R. L., Shamir, A., and Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM 21*, 2 (1978), 120–126.

[58] Rückert, M. Adaptively Secure Identity-Based Identification From Lattices without Random Oracles.

[59] Schneider, M. Sieving for Shortest Vectors in Ideal Lattices. Cryptology ePrint Archive, Report 2011/458, 2011. http://eprint.iacr.org/.

[60] Schonhage, A., and Strassen, V. Schnelle Multiplikation grosser Zahlen. *Computing 7* (1971), 281–292.

[61] Shor, P. W. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *SFCS '94: Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1994), IEEE Computer Society, pp. 124–134.

[62] Stern, J. A New Paradigm for Public Key Identification. In *IEEE Transactions on Information Theory* (1996), vol. 42, pp. 13–21.

[63] Szydlo, M. Hypercubic Lattice Reduction and Analysis of GGH and NTRU Signatures. In *Proceedings of Eurocrypt'03* (2003), Springer-Verlag, pp. 433–448.

[64] time(1) - Linux Manual Page. http://www.kernel.org/doc/man-pages/online/pages/man1/time.1.html.