RAFAEL JOSÉ DEITOS

ALGORITMOS DE PROGRAMAÇÃO LINEAR COM ATRIBUTOS DE PRIVACIDADE PARA O USO EM COMPUTAÇÃO SEGURA MULTI-PARTE

FLORIANÓPOLIS
2009

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

ALGORITMOS DE PROGRAMAÇÃO LINEAR COM ATRIBUTOS DE PRIVACIDADE PARA O USO EM COMPUTAÇÃO SEGURA MULTI-PARTE

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia
de Automação e Sistemas.

RAFAEL JOSÉ DEITOS

Florianópolis, Outubro de 2009.

ALGORITMOS DE PROGRAMAÇÃO LINEAR COM ATRIBUTOS DE PRIVACIDADE PARA O USO EM COMPUTAÇÃO SEGURA MULTI-PARTE

Rafael José Deitos

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Automação e Sistemas, Área de Concentração em *Sistemas Informáticos*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.'

Joni da Silva Fraga, Dr.
Orientador

Eugênio de Bona Castelan Neto, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

Joni da Silva Fraga, Dr.
Presidente

Eduardo Camponogara, Dr.

Rômulo Silva de Oliveira, Dr.

Ricardo Felipe Custodio, Dr.

Sempre que te perguntarem se podes fazer um trabalho,
respondas que sim e te ponhas em seguida a aprender como se faz.
- F. Roosevelt

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus por me dar força e inspiração para realizar este trabalho, além de colocar ao meu redor pessoas magníficas.

Agradeço ao meu orientador, prof. Joni da Silva Fraga, e ao meu supervisor na SAP, Florian Kerschbaum, Dr., pela compreensão, orientação e especialmente confiaça em mim depositadas.

Agradeço ao CNPq e à SAP pelo apoio financeiro pois sem ele a realização deste trabalho não seria possível.

Quero agradecer também aos amigos e colegas que, de uma forma ou de outra, sempre me ajudaram e incentivaram durante toda essa caminhada. Obrigado Tobias S., Tobias K., Karol, Horst, Christian, Peter, Axel, Katharina, Yuecel, Alex R., Alex Y., Florian R., Kevin, Maximilian, Mathias P., York, Laetitia, Julia, Irving, Fabbio, Priscila, Gisele, Lidi e Renata.

Agradeço, por fim, à minha família que sempre esteve presente, meus pais Maristela e Darcy, e meu irmão André, que são impreterivelmente o bem mais precioso que possuo. Eles formaram a base para que eu chegasse onde cheguei. Eles me deram força, apoio incondicional, sustento, carinho, paciência, enfim, devo tudo o que sou a estas pessoas tão queridas.

Maristela, Darcy e André, amo vocês!!!

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia de Automação e Sistemas.

ALGORITMOS DE PROGRAMAÇÃO LINEAR COM ATRIBUTOS DE PRIVACIDADE PARA O USO EM COMPUTAÇÃO SEGURA MULTI-PARTE

Rafael José Deitos

outubro/2009

Orientador: Joni da Silva Fraga, Dr.
Área de Concentração: Sistemas Informáticos
Palavras-chave: computação segura multi-parte, otimização de cadeias logísticas, programação linear
Número de Páginas: xviii + 96

Tradicionalmente a pesquisa na área de segurança tem focado na proteção contra ataques externos e internos. No entanto, recentemente, com modelos de negócios em rede, uma nova ameaça de segurança surgiu: o parceiro de negócios. Otimização de Cadeias Logísticas (CL) é um exemplo onde o compartilhamento de informação pode melhorar drasticamente o desempenho de toda a cadeia. Apesar de tais problemas poderem ser modelados e resolvidos utilizando-se programação linear, requisitos de segurança impedem a sua implementação de maneira tradicional. Entre as diversas medidas de segurança já conhecidas, computação segura multi-parte (CSM) é a única a oferecer a garantia de segurança necessária enquanto computa o problema de otimização da cadeia logística. CSM é uma técnica criptográfica que permite que um conjunto de participantes computem uma função conjunta sem que seja necessária a revelação de informação. Um dos maiores desafios de CSM é a sua realização prática. Esta dissertação tem seu foco em algoritmos de programação linear com preservação da privacidade para o uso em computação segura multi-parte que podem ser utilizados na resolução de proglemas de otimização da cadeia logística. Para essa classe de problemas, existem protocolos onde a seleção do índice do elemento pivô é realizada em claro. A primeira contribuição é um esquema probabilístico para a redução do número de permutações seguras requerido pelo protocolo seguro e privado de programação linear colaborativa. Nossa solução é capaz de reduzir em aproximadamente 40% o número de permutações seguras ao custo da revelação de uma pequena quantidade de informação, além de ser capaz de controlar a relação entre segurança e desempenho de tal protocolo. Nossa segunda contribuição compreende a introdução de dois protocolos seguros para permutação multi-parte. Primeiramente, propõe-se um protocolo com complexidade linear no número de participantes e comunicação. Considerando-se cenários reais onde as cadeias logísticas são formadas por vários participantes usualmente dispersos e, considerando-se também as condições da rede de comunicação, propõe-se um segundo protocolo com complexidade de base logarítmica. É feito um estudo detalhado e uma análise de tais protocolos além de uma avaliação, na prática, das melhorias observadas quando da utilização do algoritmo de base logarítmica. Resultados experimentais revelam uma forte relação entre o número de participantes, condições da rede, complexidade de rodadas e poder de paralelização, quando se considera a otimização do desempenho de protocolos de CSM. Adicionalmente, pode-se considerar protocolos de CSM onde o índice do elemento pivô é mantido como uma variável criptografada. Computação com valores criptografados é bastante cara e, as melhores soluções conhecidas normalmente se be-

neficiam de computação paralela para a redução dos custos computacionais e de comunicação. Nosso foco é otimizar a utilização dos processadores de máquinas *multi-core/processor* quando da resolução de programas lineares seguros. Para tal, duas abordagens de programação linear segura em paralelo são comparadas e uma delas é implementada. Dada esta implementação, o desempenho é praticamente independente das condições da rede de comunicação, mas o paralelismo, ou seja, o número de *threads*, precisa ser adaptado para a otimalidade. Nossa última contribuição consiste em um algoritmo de agendamento adaptativo para a seleção dinâmica do número de *threads*, de modo que não é necessário determinar-se tal número estaticamente e *a priori* para um *speed-up* ótimo. O algoritmo pode ainda lidar com variações nas condições da rede e é capaz de alcançar desempenho próximo da otimalidade.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Automation and Systems Engineering.

PRIVACY PRESERVING LINEAR PROGRAM ALGORITHMS FOR SECURE MULTI-PARTY COMPUTATION

Rafael José Deitos

October/2009

Advisor: Joni da Silva Fraga, Dr.
Area of Concentration: Informatics
Key words: secure multi-party computation, supply chain optimization, linear programming
Number of Pages: xviii + 96

Security research has long focused on protecting against outside attackers. This was augmented with protection against insider threats, but recently with networked businesses, a new threat has arisen: security against the business partner. Supply chain (SC) optimization is an example where information sharing can dramatically improve the performance of the SC. Although many such problems can be modeled and solved using linear programming, security requirements prevent their implementation in the traditional way. Among several established security measures, secure multi-party computation (SMC) is the only one to provide the necessary security guarantee while computing the supply chain optimization problem. SMC is a cryptographic technique that allows a set of parties to compute a join function without disclosing information. A major challenge of SMC is its practical realization. This thesis focuses on privacy preserving linear program algorithms for secure multi-party computation applied to supply chain optimization. For such class of problems, there are protocols in which the pivot index selection is made openly. The first contribution is a probabilistic scheme to reduce the overall number of secure permutations required by a secure and private collaborative linear programming protocol. Our solution is able to reduce around 40% of such number at the cost of disclosing a small amount of information, and is also able to control the relation between security and performance of the secure protocol. The second contribution regards introducing two protocols for secure multi-party permutation. First we propose a protocol with a complexity that is linear to the number of parties and communication. Considering real supply chain scenarios with several business partners often geographically dispersed and network conditions, we propose a second permutation protocol with logarithmic complexity. We give a detailed study and analysis of those protocols, and evaluate, in practice, the improvements due to the log round complexity solution. Experimental results reveal a strong relation between number of parties, network conditions, round complexity and parallelization power, when considering SMC protocols optimization. Additionally, one can consider protocols for SMC in which the pivot index element is kept as encrypted value. Computing on encrypted values is very expensive, and the best known solutions often benefit from parallel computing to reduce computation and communication costs. We focus on optimizing the overall utilization of the processors on multi-core/processor machines when solving secure linear programs. We compare two approaches for parallel secure linear programming and implement one of them. Given that implementation, the performance is almost independent of the network conditions, but parallelization, i.e., num-

ber of threads, needs to be adapted to such conditions. The last contribution is therefore an adaptive scheduling algorithm for the dynamic selection of the number of threads, such that it's not necessary to statically and up-front determine the number for optimal speed-up. The algorithm can even deal with a varying network conditions and is able to perform close to the optimum.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Quando se fala em aplicativos de *software* para empresas, segurança é sempre um aspecto chave. Muitos anos de pesquisa já foram dedicados à área de segurança contra ataques externos e são inúmeros os casos de sucesso, como criptografia, *firewalls*, etc. Apesar de todas estas medidas de segurança, peritos no assunto logo perceberam que existe também uma ameaça dentro da própria empresa, as chamadas ameaças internas, ou ataques internos. Por exemplo, a propriedade-$\star$ do modelo *Bell-LaPadula* pode ser violada por um ataque interno. A pesquisa nesta área tem sido contínua e diversos casos apresentam sucesso no que diz respeito a novos mecanismos de segurança, como a separação de direitos no controle de acesso baseado em papéis - RBAC [83].

Com o aparecimento da *Internet* e o atual movimento em direção a arquiteturas orientadas a serviço, vê-se uma forte tendência para o uso da rede em modelos de negócio. As empresas frequentemente se encontram envolvidas em transações eletrônicas utilizando a *Internet*, sendo que tais transações são vitais para o sucesso dos seus negócios. Enfim, toda uma indústria está apostando seu futuro nesta tendência.

Veja, por exemplo, o caso de otimização da cadeia logística[78]. A tarefa mais importante de um plano diretor para cadeia logística (PDCL), do inglês *Supply Chain Master Plan - SCMP*, consiste em determinar quantidades de produção e inventário para toda a cadeia logística em um horizonte médio de tempo, i.e., um certo número de empresas deseja planejar produção, estoque e transporte de maneira colaborativa. O desempenho da cadeia logística como um todo (i.e., lucro) é comprovadamente determinado pela maneira como os planos diretores individuais são coordenados e sincronizados. Existem dois mecanismos distintos para coordenar as decisões do plano diretor: (i) coordenação *upstream*; e (ii) coordenação centralizada. A primeira leva a uma alocação sub-ótima das quantidades de produção, inventário e transporte, enquanto a segunda, apesar de ser capaz de resolver esse problema de maneira ótima, é raramente aceita, visto requerer o compartilhamento de informações consideradas privadas ou sigilosas.

É verdade que o compartilhamento de informação dentro da cadeia logística (entre seus integrantes) reduz custos [14, 67]. E a razão para isto é que o compartilhamento de informação melhora o processo de decisão [39]. Devido à incongruência entre incentivos das empresas envolvidas na cadeia logística e, de maneira mais geral, dos objetivos desta, um plano diretor centralizado é usualmente rejeitado. Os parceiros de negócio temem o vazamento de informação e, por isso, simplesmente se negam a compartilhar tais informações privadas, ao custo de um plano sub-ótimo. Por este motivo as decisões relevantes são comumente coordenadas de maneira descentralizada.

Dentro desse contexto, o "parceiro de negócio"aparece como uma nova ameaça de segurança. Enquanto no passado os parceiros de negócio encontravam-se separados por barreiras físicas e as transações eram baseadas apenas em papel e produtos, atualmente tais entidades podem ter acesso aos sistemas internos de seus parceiros [61]. Estudos revelam que vários incidentes relacionados à segurança envolvem parceiros de negócio e esta tem sido uma tendência crescente com o passar dos anos [4]. Exemplos de negócios colaborativos em rede incluem inventário gerenciado pelo vendedor, ou organizações virtuais para *Web Services* de negócio [37].

Sob uma perspectiva de negócio, quando uma empresa se engaja em problemas de otimização

para cadeia logística, existe uma forte necessidade de garantias de privacidade das informações. Dois requisitos básicos podem ser identificados: (i) calcular o plano diretor ótimo para a cadeia logística; e (ii) garantir a privacidade da informação compartilhada. Nem o planejamento centralizado, nem o descentralizado são capazes de satisfazer estes requisitos, pelo menos não em seus formatos padrão.

Soluções seguras são de interesse da comunidade científica e, naturalmente, da indústria. Algumas medidas de segurança contra os parceiros de negócio podem ser encontradas na prática. Tecnologias como o gerenciamento de identidades com preservação da privacidade [17], ou até mesmo as redes de comunicação anônima [40], podem ser utilizadas para proteção de indivíduos. No entanto, cenários onde empresas participam de negócios em rede normalmente envolvem informações consideradas segredos industriais, e esquemas para proteção de indivíduos não são adequados, visto que empresas possuem muito mais ativos para proteger do que suas próprias identidades.

Sistemas de reputação [59] poderiam ser usados, mas não há garantia de segurança. Eles são considerados vulneráveis a ataques, sua argumentação é complicada e tentativas de relacionar um incidente de segurança a um parceiro de negócio são difíceis de se conseguir. Outra possibilidade são as políticas de segurança. Elas são capazes de controlar os dados em sistemas remotos mas, novamente, não existe garantia de que tais políticas estão sendo aplicadas corretamente. Por fim, existe a computação segura multi-parte (CSM) [8, 25, 97] que permite aos participantes computar uma função qualquer sem que as entradas sejam reveladas. CSM fornece uma garantia de segurança, i.e., ela oferece uma medida de segurança provável de que os dados não estão sendo revelados e, ao mesmo tempo, é capaz de resolver problemas de otimização da cadeia logística. Entretanto, questões de desempenho constituem uma deficiência de CSM. Usualmente os protocolos de CSM são bastante complexos e de lenta execução.

Tecnicamente, problemas de otimização da cadeia logística, ou mais especificamente o cálculo do PDCL, podem ser modelados como problemas de programação linear (PL). Em linhas gerais, PL consiste em uma iteração no passo principal de pivoteamento, no qual o problema é representado como uma matriz. Um elemento desta matriz é selecionado de acordo com algumas regras de otimização e, depois disto, é envolvido em uma computação com cada um dos outros elementos dela. Por fim, uma nova matrix é gerada. Com o intuito de satisfazer os dois requisitos básicos comentados acima, é necessário resolver problemas de PL de maneira segura (preservação da privacidade).

Uma solução desejável para o problema de calcular o plano diretor ótimo para a cadeia logística sem revelar informações sigilosas inclui modelar tal PDCL como um problema de PL e aplicar protocolos de CSM a fim de preservar a privacidade das entradas. Este modelo envolve o compartilhamento de informações sigilosas entre os parceiros de negócio, mas não a revelação de tais dados. A divulgação da informação não é necessária para que se consiga uma política colaborativa. Finalmente, a otimização do desempenho de tais protocolos figura como um aspecto chave para o sucesso da solução.

1.2 Objetivos

Esta dissertação é resultado dos trabalhos na área de computação segura multi-parte aplicada a problemas de programação linear. O objetivo geral é permitir que os participantes de um cenário colaborativo computem o PDCL ótimo sob duas condições: (i) sem que nenhuma informação seja revelada desnecessariamente; e (ii) que este processo tenha um custo aceitável.

O aspecto prático é uma das maiores preocupações em CSM: sua implementação é muito complicada e lenta. Sendo assim, a intenção é prover protocolos práticos para PL com preservação de privacidade. Tais protocolos devem apresentar duas propriedades essenciais: (i) serem capazes de resolver o problema de otimização da cadeia logística; e (ii) evitarem que os dados de entrada (informações privadas das empresas participantes) sejam revelados. Esta última propriedade deve ser implementada de modo que seja oferecida uma medida provável de segurança com um custo aceitável (realizável).

De maneira mais precisa, definem-se objetivos específicos para esta dissertação, a saber:

1. Estudar o custo de uma permutação segura em protocolos de PL com preservação de privacidade no qual o índice do elemento pivô é mantido em claro. Devido ao alto custo de tais protocolos, propor técnicas para a "redução do número de permutações", que resultarão em um aumento de performance;

2. Projetar protocolos práticos para permutação segura multi-parte que possam ser usados em PL com preservacção de privacidade no qual o índice do elemento *pivot* é selecionado de maneira pública;

3. Otimizar soluções seguras para permutação multi-parte. Isto pode ser feito por intermédio de protocolos com complexidade de rodada reduzida. Tais protocolos devem ser capazes de contrabalancear o efeito de atrasos na comunicação entre os parceiros da cadeia logística, quando estes se encontram geograficamente dispersos (cenário real);

4. Projetar um modelo de computação multi-parte segura para computação paralela (multi processador/*core*). Este modelo deve satisfazer os requisitos do problema de otimização de PDCL;

5. Propor técnicas para melhorar o desempenho de PL com preservação de privacidade no qual o índice do elemento *pivot* é mantido secreto (criptografado). Técnicas estas podem/devem explorar o poder de computação oferecido pelo processamento paralelo.

Os trabalhos desenvolvidos e apresentados nesta dissertação foram, em parte, financiados pela "Comissão Européia"dentro do "Programa ICT"sob o *Framework 7* concessão FP7-213531 ao projeto *SecureSCM* [1].

1.3 Organização do Documento

A organização deste documento reflete os passos tomados a fim de atingir os objetivos listados na Seção 1.2. As contribuições desta dissertação podem ser divididas em duas categorias: (i) projeto de novos esquemas e algoritmos para melhorar o desempenho de soluções de PL com preservação de privacidade nas quais o índice do elemento *pivot* é selecionado publicamente. E (ii) técnicas de otimização para PL paralelo com preservação de privacidade no qual os índices dos elementos da matriz que representa o sistema são mantidos como variáveis secretas (criptografadas).

O Capítulo 2 introduz os conceitos básicos necessários para o entendimento das contribuições deste trabalho. Além disso, são descritos protocolos seguros multi-parte considerados relevantes. Relacionam-se aqui primitivas eficientes para computação com inteiros que foram implementadas e constituem a base para a construção de aplicações CSM de alto nível, como a otimização de PDCL.

O Capítulo 3 visa contextualizar o problema, fundamentando este trabalho no valor do compartilhamento da informação e na necessidade de privacidade para o gerenciamento da cadeia logística. O parceiro de negócio é identificado como uma nova ameaça de segurança. Entre as medidas já estabelecidas contra este tipo de entidade, a computação segura multi-parte é a única a oferecer soluções capazes de satisfazer os requisitos impostos. Posteriormente, apresenta-se o estado-da-arte em PL com preservação de privacidade, incluindo soluções para computação paralela.

Os Capítulos 4 e 5 descrevem a primeira parte das contribuições deste trabalho. O primeiro traz uma técnica para reduzir o número de permutações seguras em soluções como [68]. A proposta é balancear segurança e desempenho, introduzindo-se uma solução probabilística na qual uma moeda é lançada a fim de determinar quando uma permutação deve ser realizada (ou permutar porque se faz necessário). Observa-se uma melhora no desempenho quando comparado à melhor solução conhecida, ao custo de se revelar uma pequena parcela de informação, a qual pode ser quantificada.

A permutação dos elementos da matriz que representa o sistema é uma técnica muito utilizada em protocolos de programação linear segura. A complexidade de rodada é normalmente responsável pelo desempenho da computação. O Capítulo 5 apresenta duas soluções para permutação segura multi-parte. Primeiro, propõe-se uma extensão do protocolo de Li e Atallah [68] para o caso multi-parte com complexidade linear de rodada. Depois, uma solução logarítmica é apresentada. O objetivo é reduzir o efeito das condições da rede, como o atraso na comunicação, por exemplo. A solução logarítmica é capaz de realizar um $n-1$ *coalition-safe mixing* de maneira semi-honesta, em um número logarítmico de rodadas. Isto é feito multiplicando-se matrizes por meio de protocolos para multiplicação segura multi-parte. Ambas as soluções foram implementadas e avaliadas em termos de complexidades teóricas e de tempo de execução absoluto.

[1] `http://www.securescm.org/`

A segunda parte refere-se à otimização de soluções para PL segura baseadas no uso de variáveis secretas, como em [93]. O Capítulo 6 revisa o protocolo seguro de PL proposto por [93] sob a perspectiva de recursos limitados para paralelização, i.e., número limitado de processadores. Além disto, um novo modelo para PL paralela e segura é apresentado. Ele é baseado no modelo BSP de [94] e no método simplex paralelo de [41]. Nele dois passos do método simplex tradicional são distribuídos entre um certo número de processadores para execução em paralelo. Este capítulo explora as questões práticas de tais protocolos em cenários reais, onde os parceiros encontram-se dispersos e a comunicação é afetada por atrasos na rede, por exemplo. Adicionalmente, propõe-se e avalia-se um algoritmo para a seleção dinâmica do número de *threads* em máquinas paralelas a fim de obter-se melhora no desempenho.

O Capítulo 7 conclui esta dissertação trazendo as considerações finais. Os objetivos iniciais são revistos e comentados à luz dos trabalhos desenvolvidos e dos resultados alcançados. Ao fim, propõem-se futuras direções de pesquisa relacionadas tanto aos protocolos considerados quanto aos resultados práticos para otimização segura da cadeia logística.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta os conceitos básicos da área de gestão da cadeia logística e da teoria de otimização que serão posteriormente utilizados na caracterização do problema foco desta dissertação. Adicionalmente, diversas ferramentas e técnicas criptográficas necessárias para a construção de protocolos de programação linear com preservação de privacidade são introduzidas (descritas). Em especial, serão descritos neste capítulo protocolos de criptografia homomórfica com limiar e protocolos para multiplicação segura multi-parte, os quais foram implementados durante os trabalhos e fazem parte das soluções propostas.

2.1 Programação Linear - PL

A observação, em particular, de que um número de sistemas econômicos, industriais, financeiros e militares podem ser modelados (ou razoavelmente aproximados) por sistemas matemáticos de equações e inequações lineares propiciou o desenvolvimento do campo de estudos chamado "programação linear". Mais recentemente, a programação linear (e suas extensões) encontrou seu caminho dentro da área de gerência de finanças, de modo que matemáticos têm sido contratados por empresas de *Wall Street* a fim de realizar análises de locação e *portfolio* [36].

Em linhas gerais, programação linear deu à raça humana a habilidade de definir objetivos gerais e descrever um caminho detalhado das decisões a serem tomadas a fim de atingir tais objetivos, quando deparada com situações práticas de grande complexidade [24]. Para que isto seja possível, é preciso que se utilizem "modelos"matemáticos para formular tais problemas práticos, "algoritmos"para que tais modelos possam ser resolvidos, e "computadores e *software*"para que os passos dos algoritmos sejam executados de maneira automática e eficiente [95].

PL foi inicialmente proposta por George B. Dantzig, em 1947, como uma forma de definir objetivos gerais e conseguir um planejamento detalhado de como atingí-los. Atualmente, PL tem sido largamente utilizada: ela possui diversas extensões não-lineares e inteiras que são conhecidas pelo campo matemático que estudam, como programação linear, programação não-linear, programação estocástica, otimização combinatorial e maximização do fluxo de redes [36].

2.1.1 Definição

Introduz-se agora uma definição formal deste campo de pesquisa que se tornou um importante ramo de estudo em matématica, economia, ciência da computação e ciências de decisão (i.e., pesquisa operacional e gerência científica):

> **Programação matemática** (ou teoria da otimização) é o ramo da matemática que estuda técnicas para maximização e minimização de uma função objetivo sujeita a restrições lineares, não-lineares, e inteiras nas variáveis em questão [36].

O caso especial, programação linear, possui uma relação próxima com este campo de programação matemática. Ela desempenha um papel análogo ao das derivadas parciais com relação a uma função em cálculo – é uma aproximação de primeira ordem.

Programação linear preocupa-se com a maximização ou minimização de uma função objetivo linear em diversas variáveis, sujeita à restrições representadas por um conjunto de inequações e equações lineares [36].

Em diversas aplicações a solução do sistema matemático pode ser interpretada utilizando-se um programa, definido como uma descrição do tempo e da quantidade de ações a serem tomadas pelo sistema de modo que este se mova do seu estado atual em direção a algum objetivo pré-estabelecido.

Entre os métodos mais conhecidos para resolução de programas lineares incluem-se os Métodos de Ponto-Interior [58], o Algoritmo de Nelder-Mead [74] e o método Simplex [36]. Dependência da estrutura do problema e dificuldade em oferecer soluções (com preservação da privacidade) eficientes impedem os métodos de Ponto-Interior e Nelder-Mead de serem utilizados nas atividades descritas nesta dissertação. Portanto, este trabalho baseia seus protocolos com preservação da privacidade no método simplex tradicional proposto por Dantzig.

2.1.2 Método Simplex

O método simplex é um algoritmo muito popular para soluções numéricas de problemas de programação linear. Ele usa o conceito de um simplex, que é um politopo de $N + 1$ vértices em N dimensões: um segmento de linha em uma dimensão, um triângulo em duas dimensões, um tetraedro em três dimensões e assim por diante.

O método simplex funciona basicamente se movimentando de uma solução factível para outra (caminhando sobre as bordas do simplex), aumentando o valor da função objetivo a cada passo. O método termina após um número finito de iterações.

Um problema de programação linear é composto por uma função objetivo e um conjunto de restrições, que são representadas por equações lineares. Considere o seguinte problema de programação linear:

$$\begin{aligned} \textbf{minimize} \quad & c^T \cdot x \\ \textbf{sujeito a} \quad & A \cdot x = b, \quad x \geq 0 \end{aligned}$$

onde A é uma matrix retangular com dimensões $m \times n$, b é um vetor coluna de dimensão m, c é um vetor coluna de dimensão n, x é um vetor coluna de dimensão n, e a letra T sobre-escrita representa transposição de matrix. O programa linear é então organizado como uma matrix e as operações são conduzidas sobre os elementos da mesma.

O algoritmo simplex presume que existe uma solução factível inicial (de fato, uma solução básica factível inicial). Também assume-se que o problema está na sua forma canônica.

Definição (Forma Canônica): Um sistema de m equações em n variáveis x_j é dito estar na *forma canônica* com respeito a um *conjunto ordenado* de variáveis $(x_{j_1}, x_{j_2}, ..., x_{j_m})$ se e somente se, x_{j_i} possuir um coeficiente unitário na equação i e coeficiente igual a zero em todas as equações restantes.

Definição (Solução Básica): A solução especial obtida quando se define as variáveis independentes como zero e se resolve o sistema em relação às variáveis dependentes é chamada de *solução básica*.

Por conseguinte, se o sistema está na sua forma canônica com as variáveis básicas definidas como $x_1, x_2, ..., x_m$, a solução básica correspondente é $x_B = \overline{b}$ and $x_N = 0$, i.e.,

$$x_1 = \overline{b}_1, x_2 = \overline{b}_2, ..., x_m = \overline{b}_m; \qquad x_{m+1} = x_{m+2} = ... = x_n; \tag{2.1}$$

Definição (Degeneração): Uma solução básica é chamada degenerada se o valor de uma ou mais variáveis (básicas) dependentes for igual a zero. Em particular, a solução básica para a Equação 2.1 é degenerada se $\overline{b_i} = 0$ para pelo menos um i.

Um conjunto de colunas (de um sistema de equações na forma desanexada de coeficientes) é dito *básico* se elas são linearmente independentes e todas as outras colunas podem ser geradas a partir de combinações lineares entre aquelas. A fim de simplificar a discussão, assume-se que o sistema de equações original é de rank completo.

Definição (Base): De acordo com o uso em programação linear, o termo *base* refere-se às colunas do sistema original (na forma desanexada de coeficientes), *assumidas como de rank completo*, correspondendo ao conjunto ordenado de variáveis básicas para as quais a ordem da variável básica é i na linha i do sistema canônico equivalente.

Na forma canônica, a solução básica fica definida como

$$z = \overline{z}_0, \qquad x_B = \overline{b}, \qquad x_N = 0 \tag{2.2}$$

O algoritmo simplex requer qua a solução básica inicial seja factível. Isto significa que

$$\overline{x} = \overline{b} \geq 0 \tag{2.3}$$

Se tal solução não está disponível *a priori*, o método simplex inclui uma fase chamada *Fase I*, que encontra uma solução básica factível, caso exista alguma.

Definição (Forma Canônica Factível): Se a Equação 2.3 é verdadeira, o programa linear é dito estar na *forma canônica factível*.

2.1.2.1 Passos do Método Simplex

Inicialmente, introduz-se variáveis adicionais não-negativas no sistema original com o objetivo de re-escrevê-lo em uma forma canônica factível. Historiamente, estas variáveis adicionais têm sido chamadas por muitos autores como *artificiais*, ou *erro*, ou ainda *variáveis lógicas*. Aqui será utilizado o termo variáveis artificiais como um lembrete de que será necessário retirá-las da base factível inicial durante a Fase I.

O método Simplex pode ser dividido em três passos principais: primeiro, verificar se o sistema original está na forma canônica. Segundo, caso ele não esteja, a Fase I é iniciada e ao fim, uma solução factível básica é encontrada (se existir, ou sinaliza-se sua não existência). E terceiro, uma sequência de operações de pivoteamento é executada até que a função objetivo seja minimizada (ou maximizada), ou identifique-se um problema ilimitado.

Pré-processamento

A fase de pré-processamento também pode ser vista como uma sequência de passos. De acordo com o tipo de restrição, certas regras devem ser aplicadas. Ao fim, o método identifica se o sistema resultante está ou não na forma canônica.

Adição das variáveis de folga

Primeiro, as ditas variáveis de folga são introduzidas no sistema original. O número de variáveis de folga é igual ao número de restrições. Para cada restrição, o seguinte conjunto de regras é aplicado:

- Se a restrição é do tipo "$\leq$", então define-se como 1 o valor do elemento na posição (linha da restrição, coluna da variável de folga);
- Se a restrição é do tipo "$\geq$", então define-se como -1 o valor do elemento na posição (linha da restrição, coluna da variável de folga);
- Se a restrição é do tipo "$=$", nenhuma ação é realizada.

O sistema resultante (matriz) é chamado de *sistema aumentado*. Uma propriedade especial deste sistema é que as colunas correspondentes às variáveis de folga contem somente um elemento diferente de zero, i.e., igual a 1 ou -1.

Adição das variáveis artificiais

O uso de variáveis artificiais nem sempre é necessário. Um outro conjunto de regras deve ser aplicado a fim de determinar se uma variável artificial deve ser introduzida ou não. As regras são aplicadas a cada uma das restrições, como segue:

- Se a restrição é do tipo "$\leq$" e

  "$b_i \geq 0$": não se adiciona variável artificial e a variável de folga referente a esta restrição é definida como parte da base;

  "$b_i < 0$": adiciona-se uma variável artificial ao lado da última variável adicionada e atribui-se o valor 1 ao elemento na posição (linha da restrição, coluna da variável artificial);

  Ao fim, adiciona-se à base a recém adicionada variável artificial.

- Se a restrição é do tipo "$\geq$" e

  "$b_i > 0$": adiciona-se uma variável artificial ao lado da última variável adicionada e, atribui-se o valor 1 ao elemento na posição (linha da restrição, coluna da variável artificial);

  "$b_i \leq 0$": não se adiciona variável artificial e a variável de folga referente a esta restrição é definida como parte da base;

  Ao fim, adiciona-se à base a recém adicionada variável artificial.

- Se a restrição é do tipo "$=$" e

  "$b_i \geq 0$": adiciona-se uma variável artificial ao lado da última variável adicionada e atribui-se o valor 1 ao elemento na posição (linha da restrição, coluna da variável artificial);

  "$b_i < 0$": adiciona-se uma variável artificial ao lado da última variável adicionada e atribui-se o valor -1 ao elemento na posição (linha da restrição, coluna da variável artificial);

  Ao fim, adiciona-se à base a recém adicionada variável artificial.

A completa execução deste conjunto de regras dá fim à fase de pré-processamento.

Identificação da forma canônica

É desejável que ao fim da fase de pré-processamento o sistema resultante esteja na forma canônica. Caso isto seja verdade, o sistema está pronto para a Fase II. Caso contrário, a Fase I deve ser iniciada a fim de re-escrever o sistema em uma forma canônica.

Com o intuito de verificar tal propriedade, um simples teste pode ser empregado: se o número de variáveis artificiais for maior que zero (existe pelo menos uma variável artificial no sistema resultante da fase de pré-processamento), então o sistema NÃO está na forma canônica. Na prática, para que a forma canônica seja identificada, um dado protocolo deve inspecionar os seguintes parâmetros: (i) tipo de restrições, e.g. "$\geq$" ou "$=$"; e (ii) o vetor b (inspecionar se os valores de b são ou não positivos é suficiente).

Fase I

A Fase I é responsável por re-escrever o sistema de equações de modo que ele esteja em uma forma canônica. O seguinte conjunto de passos descreve o procedimento.

1. A função objetivo é substituída por uma nova função, que normalmente é igual à soma das variáveis artificiais;

2. O algoritmo Simplex é aplicado a fim de se encontrar o *mínimo* da nova função objetivo. Uma sequência de operações de pivoteamento é executada com o objetivo de reduzir o valor da função objetivo a zero;

3. Se o passo anterior obtiver sucesso, ou seja, a avaliação da função objetivo assumir valor igual a zero, significa que uma solução básica factível foi encontrada e, por conseguinte, o sistema está pronto para a Fase II. Caso contrário, o sistema não possui solução.

Fase II

A Fase II do método Simplex constitui-se em uma sequência de passos que, de modo geral, define o algoritmo Simplex. Ela é responsável por maximizar ou minimizar a função objetivo original. A condição necessária para esta fase é um sistema de equações escrito em uma forma canônica. Caso a Fase I tenha sido requerida, considera-se o passo 0 (zero) como a re-introdução da função objetivo original no sistema.

O algoritmo inicia a partir da solução básica factível encontrada ao fim da Fase I, chamada (bfs), e testa sua otimalidade. Se as condições de otimalidade forem verificadas, então a solução ótima já foi encontrada e o algoritmo termina. Caso contrário, o algoritmo busca outra solução básica factível, com um valor objetivo melhor, adjacente a anterior. A otimalidade para esta nova solução é testada novamente, e estes passos se repetem até que uma solução ótima seja encontrada. Visto que a cada nova iteração uma nova bfs é encontrada e o valor da função objetivo é melhorado, considerando-se o fato de que o conjunto de bfs's é limitado, garante-se que o algoritmo termina em um número finito de passos (iterações).

Os passos do algoritmo Simplex podem ser descritos como segue:

1. *Identificação da variável que entra na base*: encontrar o índice s do elemento mais negativo da função objetivo. Se tal elemento não existir (todos os elementos da função objetivo são positivos) **pare**, i.e., a solução básica factível ótima foi encontrada;

2. *Teste da solução ilimitada*: se todos os elementos da coluna referente à variável que entra na base forem negativos, o programa linear é dito ilimitado. Relate e **pare**;

3. *Identificação da variável que sai da base*: encontrar o índice r, $1 \leq r \leq m$, onde

$$\frac{b_r}{a_{rs}} = \min_{\{i|a_{is}>0\}} \frac{b_i}{a_{is}} \geq 0$$

   Em caso de empate, escolher r como o elemento com o menor índice (regra de Bland);

4. *Pivoteamento*: executar o pivoteamento com o elemento a_{rs} a fim de encontrar uma nova solução básica factível. Retornar ao primeiro passo;

Escolhendo-se as variáveis que entram e saem da base por meio das regras de otimização 1 (elemento mais negativo) e 3 (menor relação desempatando com a regra de Bland) garante-se que não haverá ciclos e que o método Simplex terminará em um número finito de iterações, visto que nenhuma base degenerada será selecionada.

2.1.3 Programação Linear em Paralelo

Com o desenvolvimento das ciências e da tecnologia, problemas de computação científica em grande escala têm despertado interesse de vários cientistas. Sendo assim, computação distribuída e paralela tem atraído muito a atenção de diversos pesquisadores nos últimos anos [41]. Ela apresenta-se como uma abordagem proeminente para a solução de problemas de larga escala que eram considerados de solução difícil até então (utilizando-se a tradicional computação em série).

Computadores paralelos são utilizados para solução de problemas de larga escala, como programação linear de larga escala, que são conhecidos por serem muito esparsos. Programação linear é um problema fundamental no campo da pesquisa operacional. Implementações em paralelo de algoritmos de PL têm sido estudadas e propostas em diversas arquiteturas de máquinas, incluindo-se tanto máquinas paralelas distribuídas quanto de memória compartilhada. A paralelização eficiente deste tipo de algoritmo é um dos problemas mais desafiadores. Devido a matrizes muito esparsas e à comunicação intensa, a razão entre computação e comunicação é extremamente baixa. Se faz necessária a escolha cuidadosa dos algoritmos de paralelização, dos padrões de particionamento e da otimização da comunicação, a fim de conseguir-se um bom *speedup* [90]. Pode-se definir *speedup* como a razão entre o tempo de execução do algoritmo sequencial e o tempo de execução do algoritmo paralelo com p processadores, ou de maneira mais geral, o quão mais rápido é um algoritmo paralelo quando comparado à solução sequencial. Nesta seção, descreve-se três diferentes soluções para algoritmos paralelos de programação linear.

2.1.3.1 Simplex Tradicional

O método Simplex tradicional [24, 35, 95] é o algoritmo de otimização linear mais utilizado atualmente. Ele possui diversas vantagens. Primeiro, ele pode ser muito eficiente para problemas

densos. Segundo, ele pode ser fácil e efetivamente extendido para algoritmos distribuídos de alta granularidade. Implementações em paralelo do algoritmo Simplex tradicional incluem [41] e [73].

De maneira geral, uma abordagem para resolução de problemas em paralelo consiste na quebra do problema em partes menores (sub tarefas), cada qual sendo, então, manipulada por um processador específico. Estas tarefas são coordenadas utilizando-se um mecanismo de controle. A estrutura paralela mais comumente utilizada é conhecida como estrutura "mestre-escravo". Nela, várias tarefas escravas trabalham sincronamente sob o controle da tarefa mestre. Esta última controla de perto as computações executadas pelas tarefas escravas.

Em [73], o sistema de equações na forma de *tableau* (PL) é particionado por linhas, i.e., paralilazação horizontal. Existem p *subtableaus* independentes, onde p é o número de processadores disponíveis. Cada *subtableau* é tratado como um problema individual pelo processador o qual está sendo executado e as operações descritas para o método Simplex são aplicadas individualmente a ele. Existe uma clara distinção entre o processador mestre, que contem a função objetivo, e o resto dos processadores.

De modo similar, em [41] um algoritmo de Simplex paralelo é proposto. Dong et al. baseiam seu algoritmo paralelo no modelo para computação paralela chamado BSP, proposto por Valiant em [94]. Este modelo consiste basicamente em um conjunto de pares processador-memória que interagem por meio de uma rede de comunicação que entrega as mensagens de maneira ponto-a-ponto. Adicionalmente, existe um mecanismo global de sincronização que garante que as mensagens trocadas durante a etapa de comunicação sejam entregues com sucesso. O problema é particionado em linhas, assim como algoritmo de [73], e há uma distinção entre os processadores, i.e., o processador "zero" é responsável por armazenar a função objetivo.

Ambas as abordagens descritas apresentam um considerável *speedup*, aproximadamente linear no número de processadores. No caso do algoritmo proposto em [73], nota-se uma dependência forte da estrutura do problema.

2.1.3.2 Simplex Revisado

O método Simplex possui uma variante, conhecida como método Simplex Revisado. O método Simplex Revisado é de grande interesse nos dias atuais visto ser mais eficiente na resolução de problemas de PL esparsos, que são muito comuns na prática [98]. Ele não consiste em um método totalmente novo, mas apenas uma forma diferente de se executar os passos computacionais do método Simplex tradicional. O método Simplex Revisado oferece diversas vantagens sobre o tradicional:

1. Economia considerável nas computações se a fração de coeficientes diferentes de zero for menor do que $1 - (2m/n)$. Este é um caso comum na prática.

2. Uma menor quantidade de dados precisa ser armazenada, entre uma iteração e outra, o que permite que problemas maiores sejam resolvidos em computadores com capacidade de memória limitada. Além disto, uma boa economia na quantidade de computações pode ser conseguida utilizando-se o fato de que uma grande parte das entradas do *tableau* não precisa ser atualizada entre iterações.

3. Frequentemente, a equivalente funcional da inversa da base é re-calculada por duas razões: estabilidade numérica e eficiência (manutenção da esparsividade). O custo desta operação é menor do que o cálculo da forma canônica no caso do simplex tradicional [98].

Uma versão paralela do método Simplex Revisado para problemas esparsos é proposta em [90]. Nela, o método Simplex de duas fases é utilizado: na primeira fase as variáveis artificiais são levadas a zero e a solução ótima é encontrada na segunda fase. A estrutura utilizada na paralelização é baseada em colunas, visto que o uso de um método baseado em linhas introduz comunicação extra [90]. Este método funciona aplicando-se, repetidamente a cada iteração, multiplicações vetor-matriz em paralelo seguidas de uma redução mínima [90].

Resultados experimentais descritos em [90] apontam um *speedup* quase linear obtido utilizando-se o método Simplex Revisado (1.85 para dois, 3.5 para quatro e 6.3 para oito processadores). É importante ressaltar que os resultados dependem do tamanho da matriz e de sua densidade, i.e., 306x472 no caso dos testes mencionados. Segundo os autores, outra opção seria utilizar decomposição

LU para o método revisado [90]. Mas novamente, devido a matrizes muito esparsas e à comunicação intensa, o algoritmo Simplex Revisado com decomposição LU dificilmente consegue um bom *speedup*, mesmo com padrões de paralelização cuidadosamente escolhidos e otimização na comunicação [90].

2.1.3.3 Métodos de Ponto-Interior

Outra possibilidade para a resolução de problemas de programação linear são os métodos de Ponto-Interior [58, 96]. O elemento básico destes métodos consiste em uma função de barreira auto-concordante que é utilizada para codificar o conjunto convexo. Ao contrário do método Simplex, estes métodos buscam por uma solução ótima percorrendo o interior da região factível.

Uma grande vantagem destes métodos é que estes requerem um número de iterações que é praticamente independente do tamanho do problema [53]. As melhorias devido ao uso de tais métodos são diretamente influenciadas pela estrutura do problema. Os métodos de Ponto-Interior exploram a densidade das matrizes identificando certas estruturas de bloco pré-definidas e aplicando soluções especiais para casos conhecidos.

Em [53] os autores propõem uma implementação baseada em orientação a objetos para um método de ponto-interior que é executado em paralelo. Eles incorporam o conceito de polimorfismo definindo diferentes classes que são, então, responsáveis por resolver problemas específicos, possibilitando ao método a resolução de uma grande variedade de problemas.

Os resultados são consideravelmente bons: aproximadamente linear sobre o número de processadores. No entanto, os resultados dependem diretamente da estrutura do problema, i.e., densidade da matriz que representa o sistema.

2.2 Sistemas Criptográficos Probabilísticos e Homomórficos

O objetivo da criptografia é assegurar a confidencialidade e a integridade dos dados tanto durante a comunicação quanto durante seu armazenamento. A impossibilidade do uso da criptografia em dispositivos com certas restrições de funcionamento, como restrições na capacidade de processamento, tem feito com que recursos adicionais sejam considerados, como a habilidade de delegar computações a máquinas de apoio. Neste tipo de cenário, seria desejável fornecer a tal computador não confiável apenas uma versão criptografada dos dados. Este deveria então ser capaz de executar todas as operações necessárias usando apenas os dados criptografados, sem nada saber a respeito dos valores reais. Ao fim, tal computador enviaria de volta o resultado e, somente aí, os dados seriam decifrados. De maneira coerente, os dados decifrados (resultado da computação) devem ser iguais aos dados resultantes de uma computação usando os dados originais (em claro). Para que isto seja possível, o esquema criptográfico precisa apresentar uma certa estrutura.

Rivest et al. propuseram resolver este problema, em 1978, fazendo uso da chamada criptografia homomórfica [84]. Em 1987, Brickell e Yacobi identificaram certas falhas de segurança na proposta de Rivest et al. [12]. Desde a primeira tentativa, muitos outros artigos científicos propuseram soluções para as mais diversas aplicações: compartilhamento de segredos, esquemas baseados em limiar (ver e.g., [82]), *zero-knownlegde proofs* (ver, e.g., [31]), *oblivious transfer* (ver, e.g., [70]), commitment schemes (ver, e.g., [82]), anonimato, privacidade, votação eletrônica, leilões eletrônicos, protocolos para loterias (ver, e.g., [44]), proteção de agentes móveis (ver, e.g., [87]), computação multi-parte (ver, e.g., [50, 97]), e outros mais.

Antes de apresentar de maneira detalhada sistemas criptográficos probabilísticos e homomórficos, introduz-se algumas definições. Uma chave criptográfica corresponde a uma porção de informação (ou parâmetro) que determina o resultado de um algoritmo criptográfico. Sem a chave, o algoritmo não teria resultado. Texto em claro corresponde à informação que um emissor deseja enviar a um dado receptor. Texto criptografado, por sua vez, corresponde ao resultado do processo pelo qual a informação (ou texto em claro) é transformada, com o uso de um algoritmo, de modo a tornar-se secreta ou não legível a todos excetuando-se aqueles que possuem um conhecimento específico, comumente chamado de chave. No que se refere ao processo de cifragem, a chave especifica uma transformação particular do texto em claro para o texto criptografado, ou vice-versa (processo de decifragem).

2.2.1 Criptografia Probabilística

Os sistemas criptográficos mais conhecidos são os sistemas ditos determinísticos, ou seja, para uma chave criptográfica fixa, a cifragem de um mesmo texto em claro resultará sempre no mesmo texto criptografado [43].

De maneira informal, um esquema de criptografia probabilístico é aquele no qual com uma probabilidade $\epsilon > 0$ o resultado de duas crifragens da mesma mensagem utilizando-se as mesmas chaves resultam no mesmo texto criptografado. Em outras palavras, a cifragem probabilística de uma mensagem m possui sempre um número aleatórico r como entrada. O primeiro esquema de criptografia probabilística foi proposto por Goldwasser e Micali em [52] (para uma descrição detalha favor referir-se a [52]).

A utilização de um sistema criptográfico probabilístico garante que nenhuma informação a respeito do texto em claro pode ser obtida a partir do texto criptografado. Além disso, esse sistema, quando combinado com um sistema homomórfico, oferece a possibilidade de esconder uma cifragem. Esta propriedade será apresentada na seção seguinte.

2.2.2 Cifragem Homomórfica

Seja $\mathcal{M}$ $(resp., \mathcal{C})$ o conjunto de textos em claro $(resp., ciphertexts)$. Um esquema criptográfico é dito homomórfico se para qualquer chave criptográfica k a função (transformação) de criptografia E satisfaz a seguinte condição

$$\forall m_1, m_2 \in \mathcal{M}, \quad E(m_1 \odot_{\mathcal{M}} m_2) \leftarrow E(m_1) \odot_{\mathcal{C}} E(m_2)$$

para certos operadores $\odot_{\mathcal{M}}$ in $\mathcal{M}$ e $\odot_{\mathcal{C}}$ in $\mathcal{C}$, onde $\leftarrow$ significa "pode ser diretamente computado de", ou seja, sem que seja necessário nenhum processo de decifragem intermediário [43].

Se $(\mathcal{M}, \odot_{\mathcal{M}})$ e C forem grupos, tem-se um homomorfismo de grupo. Homomorfismo de grupo é um homomorfismo entre dois grupos. Um grupo é uma estrutura algébrica consistindo de um conjunto qualquer de elementos e uma operação (adição, por exemplo) que combina dois elementos do conjunto a fim de formar um terceiro elemento, também contido no conjunto. Diz-se que um esquema é homomórfico na adição se forem considerados operadores de adição e homomórfico na multiplicação se consideram-se operadores multiplicativos.

Além disto, é interessante a obtenção de homomorfismos de anel ou algébricos. Homomorfismo de anel é um homomorfismo entre dois anéis. Um anel é definido como uma estrutura algébrica consistindo de um conjunto qualquer de elementos e duas operações binárias (usualmente adição e multiplicação), no qual cada operação combina dois elementos do conjunto a fim de formar um terceiro elemento, também contido no conjunto. Tais esquemas são capazes de satisfazer a seguinte relação:

$$\begin{aligned} \forall m_1, m_2 \in \mathcal{M}, \quad & E(m_1 +_{\mathcal{M}} m_2) \leftarrow E(m_1) +_{\mathcal{C}} E(m_2), \\ & E(m_1 \times_{\mathcal{M}} m_2) \leftarrow E(m_1) \times_{\mathcal{C}} E(m_2). \end{aligned}$$

De maneira menos formal, homomorfismo criptográfico ou cifragem homomórfica implica que, para uma chave fixa k, é equivalente executar operações sobre os textos em claro antes da cifragem, ou opearções (possivelmente diferentes) sobre os correspondentes textos criptogrados. Então é necessário haver uma certa comutatividade entre as operações criptográficas e as operações de processamento dos dados. Para tal, consideram-se apenas esquemas probabilísticos e, ainda, considera-se que E se comporta como definido nas relações acima.

Durante o decorrer deste trabalho será utilizado o sistema criptográfico probabilista e homomórfico proposto por Paillier em [77] (para uma descrição detalhada e provas, por favor referir-se a [77]). O esquema proposto por Paillier é um sistema criptográfico homomórfico na adição, o que significa que, para uma dada chave pública e textos criptografados $E(m_1)$ e $E(m_2)$, pode-se calcular, de maneira direta, o valor de $m_1 + m_2$ sem que seja necessário revelar os valores de m_1 ou m_2 (decifrar).

Outra propriedade muito útil dos esquemas de criptografia probabilísticos e homomórficos, também presente no esquema de Paillier, é a habilidade de **esconder** uma cifragem. Em alguns protocolos (e.g., Garay and Schoenmakers em [46]) as partes envolvidas precisam fazer um *broadcast* de uma

mensagem criptografada $E_r(sm)$, onde $s \in_R \{-1, 1\}$, dado $E_r(m)$. Isto pode ser feito calculando-se $E_r(sm) = E(m)^s$, fazendo-se uso das propriedades homomórficas de E. Note que todos podem calcular $(E_r(m), E_r(-m))$ dado $E_r(m)$, de modo que se alguma parte faz um *broadcast* de $E_r(sm)$, todas as outras partes são capazes de extrair o valor de s. A fim de solucionar este problema, o emissor pode **esconder** o valor de $E_r(sm)$ calculando $E_r(sm) \cdot E_{r'}(0) = E_{r+r'}(sm)$, sendo que r' é mantido em segredo. Para o caso do esquema de Paillier, pode-se utilizar uma chave pública para cifrar $E^P_{r+r'}(sm)$. O resultado pode ser transmitido à todas as outras partes, ficando impossível extrair s de maneira eficiente. Essa propriedade será explorada mais adiante nos protocolos propostos.

2.3 Sistemas Criptográficos Baseados em Limiar

De maneira geral, um sistema criptográfico baseado em limiar é aquele no qual existem uma chave pública e uma chave privada, sendo esta última é compartilhada entre os participantes. Desta forma, qualquer participante pode cifrar dados, mas os participantes precisam cooperar para que um certo texto criptografado seja decifrado. O termo limiar refere-se ao número mínimo de participantes necessários para decifrar uma certa mensagem. Esta idéia é originada da teoria de compartilhamento de segredos. De fato, basicamente a mesma técnica será utilizada durante a fase de decifragem.

Um protocolo para a geração distribuida de chaves pode ser encontrado em [11]. Protocolos para decifragem com limiar podem ser encontrados em [44], [56], [89], entre outros. Os trabalhos descritos aqui fizeram o uso de uma versão baseada em limiar do sistema criptográfico probabilístico e homomórfico de Paillier proposta por Jurik em [57].

2.4 Computação Segura Multi-Parte - CSM

Computação segura multi-parte [8, 50, 51, 97] pode ser definida como o problema em que n participantes computam uma determinada função de suas entradas de maneira segura, sendo que segurança implica a garantia da corretude da(s) saída(s) assim como da privacidade das entradas de cada participante, mesmo que alguns deles tentem trapacear. De maneira mais concreta, assume-se a existência de entradas $x_1, \cdots, x_n$, sendo que o participante i conhece x_i, e deseja-se calcular $f(x_1, \cdots, x_n) = (y_1, \cdots, y_n)$ de modo que seja garantido que o participante i conheça somente y_i e nada além disto.

O primeiro exemplo de um protocolo de computação segura multi-parte foi apresentado por Yao em [97]. Ele é conhecido como o "Problema do Milionário de Yao": dois milionários se encontram na rua e desejam descobrir quem é o mais rico. Será possível calcular tal função sem que nenhum dos participantes revele sua fortuna? Neste caso, a função a ser computada é uma simples comparação de inteiros. Se o resultado indicar que o segundo milionário é o mais rico, então este saberá que o outro participante possui menos dinheiro, mas esta deve ser toda a informação que um participante pode descobrir sobre a fortuna do outro. Outro exemplo é um esquema de votação: aqui todos os participantes possuem um inteiro como entrada, que determina o candidato para o qual cada participante pretende votar, e o objetivo é calcular quantos votos cada candidato recebeu. Se faz necessário garantir que a contagem dos votos seja correta, mas somente tal valor deve ser tornado público. Em ambos os exemplos todos os participantes obtêm o mesmo resultado, i.e., $y_1 = \cdots = y_n$, mas pode ser útil ter resultados diferentes para participantes diferentes.

Fica claro que, se pudermos calcular qualquer função de maneira segura, teremos em nossas mãos uma ferramenta muito poderosa. No entanto, alguns problemas requerem maneiras ainda mais genéricas de pensar. Um sistema de pagamentos seguro, por exemplo, não pode, naturalmente, ser formulado como uma computação segura de uma única função: o que se deseja neste cenário é acompanhar quanto dinheiro cada participante possui e evitar casos nos quais, por exemplo, alguém gaste mais dinheiro do que possui. Tais sistemas devem se comportar como um computador genérico seguro: pode receber entradas de participantes a qualquer momento e é capaz de produzir resultados para cada participante de maneira específica, baseados nas entradas atuais e nos dados previamente armazenados.

Uma ferramenta muito importante em CSM, interessante por si só, é o compartilhamento verificável de segredo (do inglês, *verifiable secret sharing - VSS*). Nela, um jogador distribui um valor secreto s entre os participantes, na qual esse jogador e/ou alguns dos participantes eventualmente trapaceiam.

É garantido que, se o jogador é honesto, os trapaceiros não obtêm informação alguma sobre s e todos os participantes honestos são capazes de, posteriormente, reconstruir o valor de s, mesmo sob a ação de participantes desonestos. Esta é, na verdade, a base de esquemas de decifragem com limiar, como mencionado na Seção 2.3. Mesmo que o jogador trapaceie, tal valor único s será determinado durante a fase de distribuição e, novamente, tal valor pode ser reconstruído pelos participantes honestos, mesmo sob a ação de participantes desonestos.

2.4.1 Adversários em CSM

É comum modelar a desonestidade (trapaça de alguns participantes) por meio da existência de um adversário que eventualmente corrompe um sub-conjunto dos participantes. Quando um participante é corrompido, o adversário obtém todos os dados que aquele possuía até então, incluindo-se aí informações completas a respeito de todas as ações e mensagens que aquele enviou até o presente momento.

Definem-se aqui, dois tipos distintos de corrupção: passiva e ativa. Corrupção passiva implica que o adversário obtém toda a informação mantida pelos participantes corrompidos, mas estes ainda executam o protocolo corretamente. Corrupção ativa implica que o adversário obtém controle total sobre os participantes corrompidos.

Os participantes honestos não sabem, pelo menos inicialmente, qual sub-conjunto de participantes está corrompido. Entretanto, nenhum protocolo pode ser seguro se admitir-se que qualquer sub-conjunto de participantes pode ser corrompido. Por exemplo, não seria possível se definir segurança, ao menos de maneira significativa, se todos os participantes estivessem corrompidos. Sendo assim, se faz necessária uma maneira de especificar tal limitação no sub-conjunto de participantes que um adversário pode corromper. Para tal, define-se uma estrutura para o adversário $\mathcal{A}$, que consiste simplesmente de uma família de sub-conjuntos de participantes. Além disso, define-se um $\mathcal{A}$-adversário como sendo um adversário capaz de corromper somente um sub-conjunto de participantes contido em $\mathcal{A}$. A estrutura do adversário poderia, por exemplo, consistir de todos os sub-conjuntos com cardinalidade menor que algum valor de limiar t. Com o intuito de que isto faça sentido, precisa-se assegurar para qualquer estrutura de adversário que, se $A \in \mathcal{A}$ e $B \subset \mathcal{A}$, então $B \in \mathcal{A}$. Intuitivamente, se o adversário for poderoso o suficiente para corromper o sub-conjunto A, então é razoável assumir-se que ele também pode corromper qualquer sub-conjunto de $\mathcal{A}$ [27].

Ambos os tipos de adversários, i.e., passivo e ativo, podem ser tanto estáticos quanto adaptativos. O primeiro caso implica que o conjunto de participantes corrompidos é escolhido uma única vez, antes da inicialização do protocolo. O segundo, por outro lado, implica que um adversário pode, a qualquer momento durante a execução do protocolo, corromper um novo participante, baseado na informação que ele possui naquele instante, desde que o total de participantes corrompidos esteja dentro dos limites definidos pelo conjunto $\mathcal{A}$, i.e., cardinalidade de $\mathcal{A}$.

Além disto, assume-se que um adversário tem acesso a todas as mensagens enviadas, mas não pode modificar as mensagens trocadas entre os participantes honestos. Isto significa que a noção de segurança somente pode ser garantida em um sentido criptográfico, i.e., assumindo-se que o adversário não é capaz de resolver algum problema computacional.

2.4.2 Modelos de Comunicação

Assume-se, durante este trabalho, que a comunicação é síncrona, i.e., processadores possuem relógios que são de alguma maneira sincronizados e, quando uma mensagem é enviada, esta chegará ao destino em um tempo limitado. De maneira mais detalhada, assume-se que os protocolos são executados em rodadas: em cada rodada cada participante eventualmente envia uma mensagem para os outros participantes e todas as mensagens são entregues antes que a próxima rodada se inicie. Assume-se, ainda, que em cada rodada o adversário primeiramente vê todas as mensagens enviadas pelos participantes honestos para os participantes desonestos (ou, para o caso no qual assume-se segurança no sentido criptográfico, todas as mensagens). Se tal adversário for adaptativo, ele pode decidir corromper, neste momento, algum participante que até então era honesto. E, somente depois disso, o adversário será capaz de decidir quais mensagens ele irá enviar no lugar do participante, agora corrupto.

Em um modelo de comunicação assíncrono, no qual a entrega de mensagens e os limites de tempo não são garantidos, ainda é possível resolver-se a maioria dos problemas considerados aqui.

No entanto, assume-se o modelo síncrono – por simplicidade, mas também devido a problemas que somente poderiam ser resolvidos de maneira fraca quando da utilização de comunicação assíncrona. Note, por exemplo, que se não houver garantia de entrega de mensagens, também não haverá garantia de que um protocolo gerará algum resultado.

2.4.3 Definição de Segurança

Considerando-se a generalidade dos problemas que a computação segura multi-parte se propõe a resolver, definir segurança de protocolos CSM se torna uma tarefa bastante difícil. A fim de solucionar este problema, usaremos a seguinte abordagem: além da noção de mundo real no qual os protocolos são executados de fato e os ataques de segurança acontecem, definiremos um mundo ideal, que consiste basicamente na especificação do que gostaríamos que o protocolo fizesse. A idéia é, então, dizer que um protocolo é bom se o que ele produz como saída não puder ser distinguido do que é produzido quando da sua aplicação no mundo ideal.

Presuma a existência de um computador incorruptível, chamado de *Funcionalidade Ideal F*, no mundo ideal. Todos os participantes podem, de maneira privada, enviar dados (entrada) para F e receber dados (saída) de F. F é programada para executar um certo número de comandos e irá sempre executá-los corretamente (visto ser incorruptível) de acordo com a especificação (que é pública), sem que nenhum dado seja revelado, exceto as saídas que devem ser enviadas aos participantes. Considere, ainda, a existência de um simulador S capaz de representar tal funcionalidade, sem a interação dos participantes e, ainda assim, retornar a saída correta. A este conjunto de entidades dá-se o nome de *mundo ideal* [27].

O objetivo de um protocolo π é criar, sem a ajuda de partes confiáveis e, na presença de algum adversário, uma situação "equivalente"àquela na qual F estaria disponível. Se isso for verdade, pode-se dizer que π *realiza* F *de maneira segura*. Por exemplo, o objetivo de computar-se uma função de maneira segura pode ser especificado por uma funcionalidade ideal que recebe entradas de participantes, avalia tal função nas entradas e retorna o resultado (saída) aos participantes. De fato, qualquer tarefa criptográfica, como os esquemas de *commitment* ou sistemas de pagamento (anteriormente citados), podem ser naturalmente modeladas utilizando-se essa funcionalidade ideal.

O *mundo real* contém o ambiente Z e os participantes $P_1, \cdots, P_n$, os quais são modelados como máquinas de Turing interativas. Os participantes se comunicam por meio de uma rede síncrona utilizando canais abertos ou mediante comunicação par-a-par perfeitamente segura, como especificado anteriormente. O ambiente Z inclui o adversário, de modo que Z pode fazer tudo o que já foi descrito anteriormente em relação a um adversário, i.e., participantes podem ser corrompidos de maneira passiva ou ativa, estática ou adaptativamente, de acordo com a estrutura do adversário $\mathcal{A}$. Os participantes seguem seus respectivos programas, especificados pelo protocolo π, até que sejam corrompidos e possivelmente controlados por Z [27].

Basicamente, a idéia consiste no fato de que o ambiente Z deve ser capaz de agir da mesma maneira tanto no mundo real quanto no mundo ideal. Sendo assim, qualquer funcionalidade que possa ser imaginada, poderia ser realizada de maneira segura simplesmente programando F apropriadamente. Entretanto, o mundo ideal não existe na vida real. Ele fornece somente uma especificação da funcionalidade que gostaríamos de ter. A questão é que podemos ter convicção de que qualquer requisito (sensato) de segurança imaginável pode ser automaticamente satisfeito no mundo ideal. Isto porque precisamente tudo é executado por uma entidade incorruptível. Desta forma, se for possível projetar um protocolo que é, em um sentido amplo, equivalente à funcionalidade ideal, pode-se ter certeza de que a utilização de tal protocolo garantirá as mesmas propriedades de segurança no mundo real.

Considera-se, ainda, a definição de modelos de segurança para computação segura multi-parte. As definições utilizadas são estáticas no sentido de que o conjunto de participantes desonestos é fixado antes do inicio da execução do protocolo, ao invés de ser determinado adaptativamente durante a execução do mesmo.

2.4.3.1 O Modelo Semi-Honesto

Este modelo é definido exatamente como no caso dos adversários (Seção 2.4.1). Relembre que um participante semi-honesto é aquele que segue o protocolo de acordo, mas recorda todas as computações

intermediárias. De maneira geral, no modelo semi-honesto de segurança, considera-se a existência de um conjunto de participantes semi-honestos durante a execução de um dado protocolo [50].

Uma propriedade resultante da definição do modelo é o valor escolhido para o limiar, quando forem utilizados protocolos multi-parte baseados em limiar. Para o modelo semi-honesto, o valor do limiar é definido como $t < (p-1)$, onde p é o número de participantes.

2.4.3.2 Os Modelos Desonestos

Considera-se agora um número arbitrário de participantes maliciosos para um dado protocolo e definem-se, portanto, dois modelos alternativos:
1) O modelo no qual o número de participantes que desvia do protocolo é totalmente arbitrário [50]. Este caso assemelha-se ao caso dos adversários adaptativos. Em particular, neste modelo não se pode evitar que participantes maliciosos abortem a execução do protocolo e a definição de segurança deve levar em conta tal fato, i.e., terminação atencipada. Consequentemente, o valor do limiar para tal modelo é comumente definido como $t < p/3$, para p participantes [27].
2) O modelo no qual o número de participantes que desvia do protocolo é estritamente menor que metade do total dos participantes [50]. Neste caso, não há necessidade de se considerar o caso de terminação antecipada. Assume-se, para este segundo modelo de segurança, o valor do limiar como $t < p/2$, no qual p é o número total de participantes [27].

2.4.4 Multiplicação Segura Multi-Parte

Nesta seção descreveremos o protocolo para multiplicação segura proposto por Cramer, Damgard e Nielsen, aplicável a cenários multi-parte [29]. Assuma $\overline{\alpha}$ como sendo a representação de uma variável qualquer α na forma criptografada. Adição, subtração e multiplicação (por uma constante) homomórficas são representadas por $\oplus$, $\ominus$ e $\odot$, respectivamente. Além disto, baseados nos Σ-protocolos para esquemas de cifragem homomórfica com limiar para provas de conhecimento do texto em claro e corretude da multiplicação, Cramer et al. constróem versões para o caso de n participantes, chamadas de $POPK$ e $POCM$, respectivamente (para detalhes a respeito dos protocolos, favor referir-se a [29]).

Assuma que todos os participantes honestos P_i conhecem valores públicos $k_N = \{k_i\}_{i \in N}$, pk, e valores cifrados $\overline{a}$, $\overline{b} \in E_{pk}(a)$, para algum $a, b \in R_{pk}$, possivelmente desconhecidos, e valores privados sk_i. Ainda, assuma a existência de um conjunto de participantes N', que corresponde àqueles não foram pêgos trapaceando durante a execução do protocolo e é conhecido por todos os outros participantes. Os participantes corrompidos são controlados por um adversário e o objetivo (de todos eles, honestos ou não) é computar um valor comum $\overline{c} \in E_{pk}(ab)$ sem que ninguém descrubra algo novo sobre os valores de a, b, ou $a \cdot b$.
Implementação

1. Primeiro todos os participantes compartilham, de maneira aditiva e segura, o valor de a.
   (a) P_i, para $i \in N'$ escolhe um valor d_i uniformemente aleatório em R_{pk}, calcula o valor de $\overline{d}_i \leftarrow E(d_i)$ (cifragem), realiza um *broadcast* do resultado e, participa do protocolo $POPK$ a fim de provar que todo P_i conhece r_i e d_i de forma que $\overline{d}_i = E_{pk}(d_i)[r_i]$.
   (b) Seja N'' o conjunto de participantes que completaram a prova de (a) de maneira correta, e seja $d = \sum_{i \in N''} d_i$. Todos os participantes então calculam $\overline{d} = \oplus_{i \in N''} \overline{d}_i$ e $\overline{e} = \overline{a} \oplus \overline{d}$.
   (c) Os participantes contidos em N'' invocam um protocolo de decifragem homomórfica a fim de calcular (de maneira colaborativa) o valor de $a + d$ a partir de $\overline{e}$.
   (d) O participante com o menor índice de N'' então define $\overline{a}_i \leftarrow \overline{e} \ominus \overline{d}_i$ e $a_i \leftarrow a + d - d_i$. Os outros participantes contidos em N'' definem $\overline{a}_i \leftarrow \ominus \overline{d}_i$ e $a_i \leftarrow -d_i$.
2. Cada participante P_i para $i \in N''$ calcula $\overline{f}_i \leftarrow a_i \odot \overline{b}$, realiza um *broadcast* de $\overline{f}_i$, e participa do protocolo $POCM$ a fim de provar que todos os $\overline{f}_i$ foram calculados corretamente. Considere X o sub-conjunto dos participantes que falharam a prova, e seja $N''' = N'' \backslash X$.
3. Os participantes calculam $\overline{a}_X = \oplus_{i \in X} \overline{a}_i$ e decifram o resultado utilizando novamente um protocolo de decifragem homomórfica, a fim de obter o valor de $a_X = \sum_{i \in X} a_i$.

4. Todos os participantes calculam $\overline{c} \leftarrow (\oplus_{i \in N'''} \overline{f}_i) \oplus (a_X \odot \overline{b}) \in E_{pk}(ab)$.

Este protocolo é comprovadamente seguro e correto [29]. Considerando-se a complexidade e assumindo-se o modelo de segurança semi-honesto com uma implementação em paralelo e um protocolo de decifragem constante no número de rodadas, como o apresentado por Jurik em [57] (baseado na teoria apresentada na Seção 2.3), o protocolo de multiplicação segura multi-parte de Cramer et al. possui um número constante de rodadas, i.e., 5 rodadas.

2.4.5 Comparação Segura Multi-Parte

A literatura possui vários exemplos de protocolos seguros para comparação em cenários multi-parte. Nesta seção serão descritos três destes protocolos, os quais poderiam ser implementados e empregados em soluções para programação linear com preservação da privacidade.

1) Toft: O Capítulo 10 de [93] introduz protocolos para função desigualdade aplicável a cenários multi-parte nos quais as variáveis são criptografadas ou secretas. Duas propostas são apresentadas, uma mais genérica e outra com requisitos especiais para as entradas, i.e., entradas limitadas. A primeira solução é capaz de comparar quaisquer duas variáveis secretas a um custo computacional de $84l+5$ multiplicações seguras em 25 rodadas (das quais 13 podem ser vistas como pré-processamento), na qual l representa a precisão das variáveis, em bits. Em [93], Toft prova que tal protocolo é perfeitamente correto e seguro [93]. A segunda solução é mais eficiente, porém a segurança perfeita precisa ser sacrificada. Assuma computações executadas sobre o grupo $\mathbb{Z}_q$. Existe a possibilidade de encontrar variáveis de tamanho limitado (i.e., k-bit), com $2^{k+\kappa} \ll q$. Assim, as variáveis são restritas a k bits, enquanto o tamanho em bits de q é pelo menos κ vezes maior. Toft introduz um protocolo que realiza a comparação de duas variáveis secretas a um custo computacional de $15k + 2\kappa$ multiplicações em 16 rodadas (valor este que pode ser melhorado utilizando-se um pré-processamento, resultando em um total de 12 rodadas).

2) Garay, Schoenmakers and Villegas: Em [46] os autores apresentam dois protocolos para comparação segura de inteiros. Assuma que as computações são realizadas sobre $\mathbb{Z}_q$, para um primo grande q (de, digamos, tamanho igual a 160 bits). Primeiramente, é apresentado um protocolo com complexidade logarítmica de rodadas, baseado em um circuito booleano elegante para comparações de inteiros com profundidade $\log_2 m$ para inteiros com tamanho m-bits. No pior caso, são necessárias $3m-2$ portas lógicas condicionais, resultando em aproximadamente $150m$ exponenciações e $102m|q|$ bits trocados (por meio de *broadcast*). A profundidade do circuito é exatamente $\lceil \log_2 m \rceil$, o que implica em uma complexidade de $\mathcal{O}(\log m)$, com uma constante escondida igual a 1 para logaritmo base 2. Segundo, os autores propõem um protocolo que pode ser executado em um número constante de rodadas e requer um número de multiplicações seguras que é um múltiplo de m. No entanto, tal protocolo é restrito ao caso de duas partes (ou, pode-se assumir, qualquer número constante de parceiros). Em particular, os autores apresentam uma técnica eficiente para retornar o bit de saída de forma criptografada. Em se tratando de complexidade, o número de rodadas é no máximo 9. O número de exponenciações é igual a $124m$, no total. E, de maneira similiar, o número de bits trocados durante a execução do protocolo é igual a $77m|q|$. Note que este protocolo pode ser extendido para o caso multi-parte, mas devido ao uso de uma técnica de embaralhamento, executada em sequência, não é se possível obter um número constante de rodadas.

3) Damgard, Geisler and Kroigard: Em [33], Damgard et al. propõem um algoritmo de criptografia homomórfica muito eficiente. Eficiência é obtida em parte pelo uso de uma variante da idéia de Groth de explorar sub-grupos de Z_n^* para um modulo RSA n [55], e parcialmente pelo fato de tal algoritmo utilizar espaços menores para textos em claro, de tamanho igual a $\theta(l)$, sendo que l é o tamanho em bits de um dado inteiro. Para comparar dois inteiros, o protocolo apresentado troca 2 mensagens de tamanho $\mathcal{O}(l \log l + k)$ bits, mais 2 mensagens de tamanho $\mathcal{O}(lk)$ bits, onde k é o tamanho em bits do módulo RSA utilizado pelo sistema criptográfico. No que diz respeito a complexidades, cada participante precisa executar $\mathcal{O}(l(t+\log l))$ multiplicações $mod\ n$, onde t é um parâmetro de segurança. Valores realísticos para os parâmetros podem ser, por exemplo, $k = 1024$, $t = 160$ e $l = 16$.

2.4.6 Considerações sobre CSM

Os resultados clássicos para o modelo teórico de informação de Ben-Or, Goldwasser e Wigderson [8], e Chaum, Crepeau e Damgard [19], afirmam que qualquer função pode ser computada de maneira

segura garantindo-se a segurança perfeita, mesmo na presença de um adversário adaptativo e passivo (adaptativo e ativo), se e somente se, tal adversário corromper um número menor do que $n/2$ ($n/3$) do total de participantes.

Quando um canal para *broadcast* de mensagens estiver disponível, qualquer função pode ser computada de maneira segura garantindo-se a segurança estatística, mesmo na presença de um adversário adaptativo e ativo, se e somente se, o adversário corromper um número menor do que $n/2$ do total de participantes. A prova para esta afirmação foi primeiramente demonstrada por Rabin e Ben-Or em [80]. Os protocolos mais eficientes para este cenário foram propostos por Cramer, Damgard, Dziembowski, Hirt e Rabin em [28]. Os resultados mais genéricos para o modelo criptográfico são de Goldreich, Micali e Wigderson [51], que mostram que, assumindo-se a existência de uma permutação *trapdoor one-way*, qualquer função pode ser computada de maneira segura na presença de um adversário estático e ativo que corrompe um número inferior a $n/2$ dos participantes. Canetti et al. mostraram em [18], que segurança contra adversários adaptativos para o modelo criptográfico também pode ser obtida. No entanto, há uma perda de eficiência associada a tal solução. Sob pressupostos de número teórico específicos, Damgard e Nielsen mostraram que segurança adaptativa pode ser obtida sem perda de eficiência quando comparados com os melhores resultados conhecidos de segurança estática [75].

Este trabalho tem o objetivo de resolver problemas de programação linear de maneira segura. Para tal, serão consideradas, durante todo o resto deste documento, as seguintes premissas de segurança:

- O modelo de segurança semi-honesto [50] é assumido, no qual os participantes seguem o protocolo, mas tentam obter mais informações do que o permitido;
- Assume-se a existência de adversário passivos, o que implica que participantes corrompidos executam o protocolo corretamente;
- O modelo de comunicação segue o modelo criptográfico. Os protocolos são executados em rodadas e a camada de comunicação garante a entrega de todas as mensagens na rodada atual, assegurando que nenhuma mensagem seja perdida ou entregue de maneira incorreta;
- No contexto do modelo de segurança semi-honesto, o valor para o limiar utilizado pelo sistema criptográfico é definido como $t < p - 1$, em oposição ao valor definido para o modelo malicioso, $t < p/2$, por exemplo, no qual p é o número de participantes;

2.5 Conclusão

Este capítulo apresentou os conceitos básicos mais importantes e necessários para a compreensão dos resultados deste trabalho. Diferentes ferramentas teóricas como o método Simplex, sistemas criptográficos homomórficos com limiar e protocolos seguros para multiplicação multi-parte também foram descritas. Tais ferramentas formam a base para a construção de protocolos de programação linear com preservação da privacidade aplicáveis a problemas de otimização da cadeia logística e serão, portanto, utilizados durante o decorrer deste trabalho.

Capítulo 3

Contextualização e Estado-da-Arte

Dentro do contexto de algoritmos de programação linear para computação segura multi-parte e aplicações, como para a otimização da cadeia logística, este capítulo está dividido em três partes. Primeiramente, serão apresentadas questões de segurança envolvidas na solução colaborativa do problema de otimização em programação linear, com foco em nosso problema exemplo, a geração do plano diretor da cadeia logística. Depois, discutiremos as razões pelas quais somente computação segura multi-parte é capaz de resolver este problema de otimização com garantia de segurança. Por fim, o estado da arte em programação linear com preservação da privacidade será descrito em detalhes.

3.1 Introdução

O plano diretor da cadeia logística objetiva a optimalidade no alinhamento das decisões referentes à produção, armazenamento e transporte entre os múltiplos parceiros da cadeia. Na prática, é comum observar-se um mecanismo de coordenação descentralizado (conhecido como planejanemto *upstream*) que possibilita somente alcançar um ótimo local, em contraste com o almejado planejamento ótimo global para toda a cadeia. Pelo menos em teoria, um plano diretor ótimo para toda a cadeia pode ser gerado se alguma entidade de planejamento possuir à sua disposição a união de toda a informação (que cada parceiro da cadeia possui) relevante ao planejamento. No entanto, é sabido que as empresas tipicamente não compartilham informações consideradas sensíveis (privadas), e.g., dados sobre capacidade e custos. Elas temem que outras empresas envolvidas na mesma cadeia logística venham a explorar esta informação, tornando-se uma desvantagem. Os maiores obstáculos ao planejamento centralizado podem ser removidos se um mecanismo para geração segura e com respeito da privacidade do plano diretor for colocado em ação.

Em se tratando da otimização da cadeia logística, o desempenho da cadeia como um todo é determinado pela maneira como os planos individuais são coordenados e sincronizados [79]. No caso do planejamento *upstream*, cada domínio de planejamento gera seu plano diretor ótimo (baseado nas ordens recebidas ou na demanda prevista) sem considerar seu efeito nos domínios restantes. Como consequência, este tipo de coordenação tipicamente leva a alocação sub-ótima da produção, inventário e quantidades de transporte, assim como a sub-optimalidade na definição das conexões de transporte de toda a cadeia. Por outro lado, o planejamento centralizado leva comprovadamente a um ótimo global, o que reduz significativamente os custos e também previne efeitos colaterais da falta de troca de informação, e.g., o chamado "efeito *bullwhip*". No entanto, o mecanismo centralizado enfrenta problemas como a dificuldade no alinhamento das decisões individuais referentes aos objetivos globais da cadeia, a delegação das decisões de planejamento a uma entidade central, e a resistência ao compartilhamento de informações privadas [79].

Ao passo que negócios colaborativos em CL estão se tornando cada vez mais comuns, os parceiros de negócio estão sendo encarados como um novo tipo de ameaça de segurança. Se no passado as transações eram limitadas a papel e bens e os parceiros encontravam-se claramente separados, na atual conjuntura dos negócios, virtualmente não existem mais barreiras físicas e a informação tornou-se um elemento crucial nas transações, de modo que os parceiros são capazes de acessar informações internas dos demais. Os trabalhos de [4] reportam que parceiros de negócio têm sido envolvidos em diversos incidentes de segurança e esta é uma tendência crescente nos últimos anos.

Este capítulo é dividido como se segue. Nas Seções 3.2 e 3.3 são caracterizados o valor da informação e a necessidade de privacidade para o gerenciamento da cadeia logística, respectivamente. A Seção 3.4 discute soluções comumente aplicadas e suas desvantagens. CSM é uma técnica criptográfica que pode ajudar na resolução de tais problemas ao passo que possibilita a computação correta do plano diretor ótimo, com garantias de segurança. A Seção 3.5 examina o estado da arte para programação linear com preservação de privacidade aplicado a problemas de gerenciamento da cadeia logística. Devido ao enorme crescimento no poder de computação, a possibilidade de utilizarem-se algoritmos paralelos figura como uma boa oportunidade para aumento do desempenho em aplicações práticas. A Seção 3.6 considera diversas abordagens e descreve nossa melhor opção. Finalmente, a Seção 3.7 conclui o capítulo.

3.2 O Valor da Informação na Gestão da Cadeia Logística

A assimetria da informação é conhecida por criar ineficiências no gerenciamento de cadeias logísticas. Entre elas cita-se sub-investimento em capacidade levando à escassez de recursos; má alocação de inventário, transporte e gerenciamento de recursos deficiente; preços demasiadamente altos; além da redução nos serviços ao cliente. Tais ineficiências possivelmente levam, também, ao uso demasiado de envio *premium*, à penalidades crescentes resultantes de paradas de linha e à perda de contratos de negócio.

Por outro lado, o compartilhamento de informações a respeito dos níveis de inventário, estado dos pedidos, previsões de demanda, agendamentos de produção e entrega, entre outros, pode dramaticamente melhorar o desempenho da cadeia logística. O trabalho de [67] descreve diversos exemplos reais nos quais isto acontece. A razão para tal melhoria não é o compartilhamento de informação, por si só, mas, de fato, porque informações compartilhadas melhoram o processo de decisão.

Diversos artigos científicos têm focado no valor do compartilhamento da informação para a cadeia logística, em uma grande variedade de cenários. Chen [20] faz uma revisão no papel do compartilhamento da informação na busca de colaboração na cadeia logística e revisa as consequências na falha em compartilhar (ou em transmitir) tais informações. Lee [64, 65] mostra que a falha no compartilhamento de informações como a demanda de venda pode levar a um fenômeno conhecido como o "efeito *bullwhip*". Devido à instabilidade na demanda de consumo, empresas normalmente possuem um inventário reserva. Em períodos de demanda crescente, parceiros da CL de níveis mais altos (*down-stream*) irão aumentar seus pedidos. Em períodos de demanda decrescente, pedidos irão cair ou parar com o intuito de reduzir o inventário. O efeito é que as variações são sempre amplificadas quando se move para a base da cadeia logística (para mais longe do cliente). Entre as possíveis causas incluem-se o mau uso das políticas de estoque básico, os falsos *feedbacks* e atrasos, os erros nas previsões, a variação no *lead time* (erro na previsão durante o reaprovisionamento do *lead time*). Além de grandes reservas no inventário, o efeito *bullwhip* pode levar tanto a produção ineficiente, quanto a inventário excessivo, ao passo que o produtor precisa atender a demanda de seus predecessores da CL. Outra consequencia é a baixa utilização do canal de distribuição. Adiciona-se, ainda, o perigo da falta de estoque, o que resulta em má qualidade no atendimento ao cliente e, uma série de outras despesas financeiras. Ao lado das consequências financeiras devido ao péssimo serviço prestado ao cliente e aos danos causados à imagem da empresa e à fidelidade, uma organização precisa lidar com as consequências do não cumprimento dos pedidos, o que pode levar a sanções contratuais. Por fim, relacionam-se aos efeitos do não compartilhamento adequado da informação, os custos induzidos pela contratação e demissão de empregados para atender as variações na demanda.

Exemplos de compartilhamento de informação a jusante como níveis de inventário incluem [14], [21], e [48]. O trabalho de [66] quantifica o valor do compartilhamento de informações sobre demanda em um modelo de cadeia logística com processo de demanda não-estacionário. Outros exemplos de compartilhamento de informação na cadeia logística incluem [22] (informação sobre custos), [23] (informação sobre *Lead-time*), e [38] (informação sobre capacidades e potencial de mercado).

Existe ainda uma extensa literatura na área de construção de esquemas de incentivo à colaboração em CL [13]. Por exemplo, Corbett [26] considera o impacto da assimetria da informação na coordenação de cadeias logísticas. Para um cenário com dois escalões, Lee and Whang [63] construíram um esquema não-linear para alinhamento dos incentivos entre fornecedor e vendedor com um sistema centralizado.

Em [16], um sistema de pagamento linear é construído a fim de se conseguir níveis ótimos de estoque básico usando equilíbrio Nash como parte da solução. Considerando-se um modelo de período único, utilizando-se um *framework* de Stackelberg, [15] e [72] demonstraram que existem incentivos para o vendedor inflar previsões de demanda.

Na prática, tecnologia da informação tem facilitado o compartilhamento da informação e a tomada de decisão em cadeias logísticas. Algumas empresas, mais notavelmente Wal-Mart, têm demonstrado os retornos extraordinários resultantes da tomada de decisão baseada em informação compartilhada. Em particular, o serviço *RetailLink* do Wal-Mart tem se tornado um padrão pelo qual outras cadeias logísticas são medidas.

3.3 A Necessidade de Privacidade

Mesmo conhecendo os enormes benefícios, muitas empresas evitam o compartilhamento de informações ditas *privadas*. Existem diversas razões para esta carência no compartilhamento de informações, na prática. Lee e Whang [67] relatam que parceiros da cadeia logística raramente compartilham informações ligadas a custos. Uma das razões para o não compartilhamento de informações sensíveis é o medo de que outros parceiros da CL tirem vantagem do conhecimento de tais informações reduzindo preços. Outra preocupação diz respeito à confidencialidade das informações a serem compartilhadas. Por exemplo, um vendedor pode não querer compartilhar com seus fornecedores informações a respeito de promoções, com medo de que tal informação possa vazar para um competidor. Questões relacionadas à confiança entre parceiros, ou à falta dela, podem surgir ao passo que alguns parceiros podem abusar das informações compartilhadas destruindo todos os possíveis benefícios do compartilhamento da informação. Se um dos parceiros for o governo, então existem razões de segurança nacional para proteger informações secretas. Finalmente, empresas podem não querer compartilhar informações devido ao medo de violar leis anti-truste (ou governamentais).

Literatura na área de privacidade em cadeias logísticas devido ao medo do vazamento de informação é bastante escassa. Li [69] mostra que o medo do vazamento de informação eventualmente faz com que o vendedor não compartilhe informações de demanda com o fornecedor. Recentemente, Amand [2] formalizou o impacto do vazamento de informação no momento de adquirir e compartilhar informações sobre a demanda. Uma corrente relacionada da literatura foca em como a informação pode ser adquirida a partir das ordens criadas pelos vendedores. Ahuja [1] e Tarantola [92] criaram o termo *otimização inversa* para representar o problema inverso de inferir valores para os parâmetros do modelo a partir de valores obtidos de certos parâmetros observáveis. Graves [54] e Raghunathan [81] mostram que o fornecedor pode inferir demanda a partir das ordens criadas pelos vendedores em um processo AR (auto regressivo). Recentemente, Gaur [47] forneceu uma completa caracterização das condições sob as quais o compartilhamento de informações sobre a demanda é importante, para processos de demanda ARMA (média móvel auto regressiva) genéricos. Raghunathan [81] também conjectura que informação compartilhada possui valor somente quando não pode ser deduzida a partir de outros parâmetros.

De maneira geral, cientistas da área de computação tem começado a integrar técnicas de preservação da privacidade com mecanismos de projeto enquanto pesquisadores da área de cadeias logísticas tem começado a integrar mecanismos de projeto a interações de cadeias logísticas.

Novos direcionamentos apontam para a superação de tais barreiras por intermédio da substituição de métodos de compartilhamento de informação tradicionais por protocolos seguros de colaboração entre parceiros da cadeias logísticas. A necessidade de privacidade requer protocolos com garantias de segurança que possibilitem aos parceiros da cadeia logística atingir os objetivos desejados a nível de sistema, sem que seja preciso revelar informações consideradas sensíveis, mesmo que as computações e decisões envolvidas utilizem tais informações. Em outras palavras, divulgação da informação não é requisito necessário para se atingirem políticas ótimas de colaboração em cadeias logísticas.

3.4 Computação Segura Multi-Parte e Praticalidade

Existem algumas medidas de segurança contra os parceiros de negócio (da cadeia logística) que são bem conhecidas. Para proteção de indivíduos citam-se tecnologias de aumento de privacidade

como gerenciamento de identidades com preservação de privacidade [17], comunicação anônima em redes [40], entre outras. No entanto, empresas possuem muito mais ativos para proteger. O fato de empresas decidirem participar de negócios em rede geralmente implica em utilização de segredos vitais para o sucesso do negócio. As pessoas comumente usam o termo segredo industrial.

O que se pode fazer em tais cenários? Pode-se considerar a utilização de sistemas de reputação [59]. Um sistema de reputação coleta informações sobre transações passadas realizadas pelos parceiros de negócio e calcula uma pontuação para a sua fidedignidade. O pressuposto é que, quanto mais alta for a pontuação, mais baixa será a chance de você trapacear ou atacar outros parceiros e, em virtude disto, maiores são as chances de atrair oportunidades de negócios. Basicamente a idéia consiste em consultar o sistema de reputação antes de participar de um determinado negócio. No entanto, sistemas de reputação possuem problemas inerentes. Primeiro, eles não são capazes de dar garantias. O que fornecem são apenas "boas dicas". Pode naturalmente acontecer que, exatamente durante a transação na qual sua empresa está envolvida, um parceiro resolva trapacear, pondo em risco toda a operação. Segundo, o problema de classificação da reputação, i.e., escala de pontuação, é complexo. Raramente uma empresa maliciosa sobrevive um longo tempo no mundo dos negócios. Ao invés disto, existem empresas com reputação excelente e empresas com reputação muito boa, porém é muito difícil diferenciá-las. Terceiro, praticamente todos os sistemas de reputação são vulneráveis. Uma boa pontuação pode ser atingida por outros meios que não bons negócios. Poucos sistemas de reputação foram analizados no que diz respeito à sua resistência a ataques e, menos ainda, sistematicamente. Quarto, processos legais são complicados e estabelecer ligações entre um incidente de segurança e um parceiro é uma tarefa bastante complexa. Agindo de maneira esperta, empresas maliciosas podem agir impunes durante muito tempo. Em particular, se tais empresas oferecerem bons serviços, elas podem ter excelentes pontuações e ao mesmo tempo serem um enorme risco às outras empresas.

Existem também as chamadas políticas grudentas (*sticky policies*). Um política grudenta é aquela transferida juntamente com os dados e aplicada (executada) no sistema remoto. A idéia básica é controlar os dados em sitemas remotos. O maior problema é, novamente, a inexistência de garantias de que tais políticas serão corretamente aplicadas, i.e., não existe garantia de segurança. Ao invés de garantias, o sistema é construído com base em relações de confiança e/ou obrigações contratuais.

Terceiras partes confiáveis também pode ser consideradas. Um entidade central possui todas as informações dos parceiros e é, portanto, capaz de calcular o plano ótimo para toda a cadeia logística. Os riscos são óbvios. Primeiro, é necessária a existência de uma rede de confiança entre os parceiros e tal entidade central. Segundo, não existe nenhuma garantia de segurança de que as informações privadas compartilhadas com a terceira parte não serão reveladas a outros parceiros (intencionalmente ou não). E terceiro, visto que nenhum dos parceiros tem conhecimento do problema como um todo, é extremamente difícil provar a corretude dos resultados. Mesmo considerando-se uma cadeia de confiança, na prática, os parceiros (empresas) relutam em compartilhar informações privadas.

Os dados necessários para o cálculo do plano diretor da cadeia logística incluem custos de produção e capacidades, de modo que sua revelação possivelmente implica um impacto negativo na posição de negócio. Sistemas de reputação são ineficazes, visto que a decisão de participar do processo de negócio já foi tomada. Todas as empresas são parte de uma cadeia logística e existe um nível básico de confiança, mas isso claramente não é suficiente. Altos níveis de confiança são extremamente caros de se construir e, mesmo assim, não são capazes de oferecer uma garantia de segurança. Políticas grudentas também não são de grande ajuda, ao passo que os dados são revelados aos parceiros. Isto é inaceitável no cenário de negócios. Se alguém tiver lido os dados, não há como evitar que eles sejam utilizados em uma outra transação, mesmo que tais políticas tecnicamente proíbam tal ação. Mais do que isso, políticas grudentas não são aplicáveis a humanos.

No entanto, existe a chamada computação segura multi-parte CSM [8, 25, 97]. CSM é uma técnica criptográfica que permite a um grupo de parceiros computar uma função conjunta sem revelar suas entradas, i.e., as entradas são mantidas privadas enquanto todos têm acesso ao resultado. CSM é aplicada por meio de protocolos que computam sobre frações de dados. Em termos simplísticos, cada protocolo toma como entrada frações de dados e produz como saída frações do resultado.

A grande vantagem de CSM está no fato de que ela oferece garantias de segurança. É possível obter-se um valor provável de segurança ou, em outras palavras, uma medida exata das chances de que as entradas não serão reveladas inadvertidamente. CSM evita a revelação das entradas e é capaz de calcular problemas de otimização para a cadeia logística. Obtêm-se ambos: o plano diretor ótimo sem

que que nenhum dado seja revelado e os (exatos) riscos associados a esta operação. A desvantagem é que a implementação de protocolos de CSM é bastante complicada e lenta.

3.5 Programação Linear com Preservação da Privacidade

A literatura referente a PL com preservação da privacidade aplicada em computação segura multi-parte para resolução de problemas de gerenciamento de cadeias logísticas é bastante escassa. Entretanto, destacam-se duas contribuições para protocolos seguros de PL aplicados ao gerenciamento de CL. A principal diferença entre elas está no modo como o índice do elemento pivô (no método simplex) é escondido, ou escolhido.

Li e Atallah [68] propõem protocolos colaborativos, seguros e com preservação da privacidade para PL, nos quais o índice do elemento pivô é selecionado em claro. A matriz que representa o sistema é permutada por cada um dos parceiros de modo que nenhum deles conhece a permutação final (que é resultado da combinação das permutações individuais de cada um). Sendo assim, o índice do elemento pivô pode ser selecionado publicamente (em claro) porque representa uma escolha aleatória entre os elementos da matriz permutada. A escolha em claro do elemento pivô tem um efeito positivo no que diz respeito ao desempenho, mas existe um custo para isso. Escolher um elemento em claro pode revelar informações desnecessárias e, portanto, um protocolo de "embaralhamento e permutação"é utilizado em cada iteração do método simplex. Além disto, a solução de Li e Atallah somente aplica-se ao caso de dois parceiros (*two-party*) em uma configuração honesta-mas-curiosa. As implicações (ou custos) decorrentes da extensão de tal protocolo para o caso multi-parte ainda não foram formalmente considerada na literatura.

Toft [93], por outro lado, introduziu primitivas para computação segura multi-parte baseadas em variáveis secretas (criptografadas), além de protocolos de alto nível para programação linear segura. O índice dos elementos é mantido como uma variável criptografada e os protocolos foram provados seguros contra adversários passivos. Desta forma, não existe necessidade de protocolos de permutação (ou embaralhamento), mas existe um custo extra associado ao fato da realização de computações com elementos criptografados. Todas as operações são do tamanho do array (ou matriz) o que exige uma intensa troca de informação, i.e., altos custos de comunicação. Toft utiliza o conceito de operações em bloco para reduzir este efeito: todas as operações independentes são realizadas em paralelo, melhorando consideravelmente o desempenho.

3.5.1 Secure and Private Collaborative Linear Programming

O modelo de segurança de Li e Atallah [68] foi provado seguro para o caso honesto-mas-curioso, i.e., os participantes seguem o protocolo corretamente, mas tentam obter mais informações do que o permitido. O comportamento desonesto, no qual os participantes não necessariamente seguem o protocolo também foi considerado em [68]. Li e Atallah introduzem um protocolo seguro e colaborativo para programação linear em um cenário de duas partes para ambos os casos.

Os trabalhos de Li e Atallah assumem que todas as entradas são números inteiros. A razão pela qual isto não é considerado uma limitação baseia-se na linearidade do problema de otimização a ser resolvido, o que permite que as entradas sejam "convertidas"para valores inteiros e as saídas, consequentemente, "revertidas"aos valores corretos.

O problema de programação linear segura e colaborativa é, então, definido como:

$$\begin{array}{l} \textit{minimize}\ \ c^T \cdot x = -z_0 \\ \textit{sujeito a}\ \ A \cdot x = b,\ x \geq 0 \end{array} \qquad \Rightarrow \qquad D = \begin{bmatrix} c^T & -z_0 \\ A & b \end{bmatrix} \begin{bmatrix} x \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

onde A é uma matriz $m \times n$, b is um vetor de dimensão m, c e x são vetores de dimensão n e, a letra T indica a operação de transposição de matrizes. A matriz do sistema D é especificada na forma canônica de um programa linear [36], e tanto ela quanto os elementos A, b, c e z_0, são aditivamente compartilhados entre dois participantes, Alice e Bob.

Um item α é dito ser aditivamente compartilhado entre Alice e Bob se Alice possui α' e Bob possui α'' de modo que $\alpha = \alpha' + \alpha'' \pmod N$, onde o valor de α é desconhecido para ambos os parceiros

[68]. O módulo N é um módulo RSA de um sistema criptográfico homomórfico como descrito em [34]. Em tal esquema, o módulo RSA é compartilhado entre os participantes mas sua fatoração é desconhecida. A geração da chave pode ser feita tanto por meio de uma terceira parte confiável quanto utilizando-se protocolos duas partes como os propostos por [11] ou [49]. Suponha que α é aditivamente compartilhado entre Alice e Bob utilizando-se as propriedades homomórficas do sistema. Então a afirmação $Enc(\alpha') \cdot Enc(\alpha'') = Enc(\alpha' + \alpha'' \; mod \; N) = Enc(\alpha)$ é sempre verdadeira. Subtração é um caso especial onde $Enc(\alpha'')^{-1} = Enc(-1 \cdot \alpha'') = Enc(-\alpha'')$.

D é uma matriz de dimensão $(m+1) \times (n+1)$. O protocolo seguro de PL de [68] é baseado no algoritmo simplex de [36] com a diferença de que, em [68], a matriz D é aditivamente compartilhada entre Alice e Bob, i.e., $D = D' + D''$ e, o protocolo mantém este invariante durante todos os passos de iteração. O protocolo pode ser resumido em uma sequência de passos, a saber: (i) embaralhar e permutar; (ii) encontrar a variável de entrada; (iii) teste de limitação; (iv) encontrar a variável de saída; (v) operação de pivoteamento; e (vi) extração do resultado.

A operação de pivoteamento requer que tanto Alice quanto Bob conheçam o índice do elemento pivô. Entretanto, o conhecimento de tal posição na matriz pode revelar informações a respeito da matriz original desnecessariamente. A fim de evitar que informações privadas a respeito da matriz do sistema sejam reveladas, um protocolo de embaralhamento e permutação é definido. Considere que Alice e Bob aditivamente dividem D e que cada um possui permutações individuais para linhas e colunas. Feito isto, eles permutam D de maneira conjunta de modo que a permutação final seja desconhecida, e a matriz permutada $\overline{D}$ é então, novamente compartilhada entre eles utilizando uma randomização diferente. Agora já não importa mais se algum dos participantes conhece um determinado índice da matriz $\overline{D}$ porque não há como relacionar tal índice com a posição correspondente na matriz do sistema original D. Tal permutação final deve ser calculada em conjunto de forma que o sistema linear resultante seja equivalente ao sistema original D, i.e., o conjunto de soluções deve ser mantido. O protocolo de embaralhamento e permutação é, então, executado antes de cada operação de pivoteamento. Para tal, Alice e Bob executam, cada um, $2(m+1) \times (n+1)$ cifragens homomórficas. Assim, o custo computacional do protocolo embaralhamento e permutação é $\mathcal{O}(mn)$ exponenciações modulares. Para a execução de uma operação de pivoteamento na posição (i, j) da matriz D, para cada $k = 1, \cdots, m+1$ exceto $k = i$ (coluna do elemento pivô), o protocolo substitui a $k$ésima equação multiplicada por d_{ij} e a $i$ésima equação multiplicada por $-d_{kj}$, onde d representa os elementos de D. O sistema resultante é equivalente ao original, i.e., o conjunto solução é mantido, mas não se encontra mais na forma canônica. Portanto, o protocolo de pivoteamento calcula os valores atualizados $\overline{d_{kl}}$ para todo $k = 1, \cdots, m+1$ e $l = 1, \cdots, n+1$, como $d_{kl}d_{ij} - d_{il}d_{kj}$. Depois, cada participante localmente subtrai suas partes de $d_{kl}d_{ij}$ e de $d_{il}d_{kj}$ e o valor de $\overline{d_{kl}}$ é aditivamente compartilhado. O custo do protocolo de pivoteamento depende do protocolo de multiplicação compartilhada de [45] e é portanto, limitado por $\mathcal{O}(mn)$ exponenciações modulares.

Ao passo em que o protocolo de *blind-and-permute* evita que informações sobre os índices da matriz original sejam reveladas, ele faz com que os índices das variáveis da base (responsáveis pela determinação da solução) também sejam permutados. Desta forma, para que a solução do problema seja encontrada se faz necessário que os índices originais dos elementos que estão na base sejam determinados. O protocolo de recuperação dos índices é baseado no próprio protocolo de embaralhamento e permutação, sendo que Alice e Bob permutam um array ordenado, usando suas permutações individuais, a fim de que seja possível extrair a permutação combinada final. Tal protocolo é executado k vezes, onde k é o número de permutações aplicadas, igual ao número de iterações do método simplex. O custo do protocolo de recuperação de índices é limitado por $\mathcal{O}(kn)$ exponenciações modulares.

Encontrar a variável de saída requer procurar pelo índice da menor razão entre os valores do vetor b e da matriz A (para maiores detalhes favor referir-se à Seção 2.1.2.1 e [36]), Sabendo-se que divisão segura de inteiros é uma operação extremamente custosa, Li e Atallah propõem a comparação de pares de inteiros para a determinação da menor razão, a fim de evitar a divisão. Para tal, eles usam um protocolo de circuito mexido que executa $\mathcal{O}(l)$ exponenciações modulares, onde l é o número de bits de precisão por variável (maiores detalhes em [68]).

Ao fim, a solução do problema de PL pode assumir valores racionais que requerem divisão segura de inteiros. Li e Atallah proprõem que os participantes Alice e Bob executem um protocolo seguro de divisão compartilhada de inteiros, como o descrito em [3], para a extração dos resultados.

No total, o custo do protocolo seguro de programação linear de Li e Atallah é limitado por

$\mathcal{O}(kmn + kln + klm + lmn)$ exponenciações modulares. Visto que $n \geq m$, o custo total se torna $\mathcal{O}(k(mn + ln) + lmn)$ [68].

Problemas de cadeias logísticas tipicamente envolvem diversos parceiros de negócios distintos, o que resulta em problemas de PL de tamanho considerável. Sabendo-se que a análise de complexidade do protocolo como um todo depende do número de iterações, o qual está relacionado ao número de participantes, a invocação do protocolo de embaralhamento e permutação em cada uma das iterações se torna uma operação extremamente cara. Adicionalmente, toda a análise conduzida por Li e Atallah (e o próprio protocolo descrito) é válida somente para cenários com dois participantes. Tais cenários são demasiadamente simples para o tipo de problemas de otimização de CL discutidos neste trabalho.

3.5.2 Secure Linear Programming

Thomas Toft [93] propôs um protocolo de programação linear com preservação de privacidade baseado em uma variação do método simplex tradicional proposto em [24]. Esta variação, por sua vez, é baseada no pivoteamento de (números) inteiros e o resultado é um protocolo que revela uma quantidade mínima de informação – o número de iterações executadas (para informações detalhadas referir-se a [86]).

Toft propõe uma funcionalidade genérica e abstrata para computações aritméticas multi-parte que permite a construção simples de protocolos. Os participantes repetidamente fornecem comandos para tal funcionalidade, os quais são então, executados. Quando a funcionalidade considerada o próximo comando a ser executado, ela requer que todos os participantes honestos forneçam o mesmo comando. O processo acontece em rodadas (*rounds*) o que possibilita a atualização do estado dos participantes de maneira iterativa, e consequentemente uma computação adaptativa, i.e., dependente dos valores de saída. Como resultado, a complexidade do protocolo seguro é equivalente àquela do algoritmo base, no sentido de que o número de operações computacionais básicas é aumentado somente por um pequeno fator constante.

O algoritmo simplex com preservação de privacidade assume um problema limitado com m restrições e n variáveis, descrito na forma de um *tableau*. O objetivo é maximizar ou minimizar a função objetivo f sendo que

$$f(x_1, \cdots, x_n) = \sum_{i=1}^{n} f_i \cdot x_i$$

onde x_i estão sujeitos a restrições na forma

$$\sum_{i=1}^{n} c_{j,i} \cdot x_i \leq b_j \text{ for } j \in \{1, \cdots, m\}$$

Variáveis de folga são introduzidas no sistema (*tableau*) o que resulta no aparecimento de inequações nas restrições. O problema é então representado em uma de suas formas canônicas:

$$\begin{array}{ccc|cccccc|c} c_{1,1} & \cdots & c_{1,n} & 1 & 0 & 0 & \cdots & 0 & 0 & b_1 \\ c_{2,1} & \cdots & c_{2,n} & 0 & 1 & 0 & \cdots & 0 & 0 & b_2 \\ & \ddots & & & & & \ddots & & & \vdots \\ c_{m,1} & \cdots & c_{m,n} & 0 & 0 & 0 & \cdots & 0 & 1 & b_m \\ \hline -f_1 & \cdots & -f_n & 0 & 0 & 0 & \cdots & 0 & 0 & z = 0 \end{array}$$

O *tableau* inicial é codificado em variáveis secretas (valores criptografados) e conhecido por todos os participantes. Estes então invocam um comando para executar o algoritmo simplex. Tal comando toma o *tableau* inicial codificado e calcula o *tableau* atualizado e final (juntamente com o último elemento pivô), que por sua vez, contém a solução final do programa linear.

Valores racionais são armazenados na forma de inteiros, dividindo-se cada valor em duas parcelas: numerador e denominador (de maneira similiar à solução empregada por Li e Atallah). O método simplex baseado no pivoteamento de inteiros repetidamente atualiza o *tableau* realizando os seguintes passos:

1. Determinar a variável que entrará na base: encontrar o índice correspondente ao primeiro ele-

mento negativo da função objetivo, i.e., a *coluna pivô* C;

2. Determinar a variável que sairá da base: encontrar o índice da linha R, tal que sua intersecção com C, C_R, seja positiva (restrição) e b_R/C_R seja mínima. A linha R é chamada de *linha pivô* e o elemento C_R é o *elemento pivô*;

3. Multiplicar todas as linhas não-pivô pelo elemento pivô;

4. Subtrair um múltiplo da linha pivô de todas as linhas não pivô de modo que a coluna pivô atualizada consista somente de zeros, excetuando-se o próprio elemento pivô;

5. Dividir as linhas não-pivô pelo elemento pivô da iteração anterior (inicializado como 1);

6. Repetir até que o vetor f consista apenas de elementos negativos;

Adicionalmente, Toft utiliza a regra de Bland para garantir a não existência de ciclos no método simplex. De acordo com a regra de Bland, em caso de empate na escolha da meno razão no passo 2, escolhe-se a razão composta pelos elementos de menor índice.

Toft considera três medidas de complexidade distintas em sua análise. Complexidade computacional é calculada de modo que uma multiplicação segura é igual a duas unidades e uma comparação segura é igual a c_r unidades, dependendo da implementação. Complexidade de comunicação se refere ao total de comunicação realizada durante o protocolo, i.e., o número de bits enviados entre todos os participantes. Sendo assim, a multiplicação de duas variáveis secretas é considerada como a unidade básica de complexidade de comunicação. Comparações seguras também são utilizadas como unidade de comunicação. Complexidade de rodada (*rounds*) é normalmente o foco principal, tanto na teoria quanto na prática. Ela compreende o número total de vezes que mensagens de tamanho arbitrário são transmitidas (entre todos os participantes, i.e., *multicast*). Toft assume ainda que, as primitivas lineares propostas por ele podem ser paralelizadas arbitrariamente, desde que os resultados necessários para o comando seguinte estejam disponíveis, o que propicia uma melhora no desempenho. Além disto, assume-se que entrada, saída e, multiplicação de duas variáveis secretas requerem exatamente uma rodada cada.

Cada passo do protocolo executa no máximo um número constante de multiplicações (seguras) por elemento do *tableau* – algumas vezes escondidas em operações de indexação – assim como no máximo um número constante de comparações (seguras) por variável. Uma análise detalhada revela, para os passos de 1 a 5, uma complexidade total limitada por $8m + n$ comparações e $5mn + 17n + 5m^2 + 22m + 3$ multiplicações – $\mathcal{O}(m + n)$ comparações e $\mathcal{O}(m(m + n))$ multiplicações, em notação big-$\mathcal{O}$. Complexidade computacional, em notação big-$\mathcal{O}$, é dominada pelo passo 2 e, é limitada por $\mathcal{O}(\log\log(m))$. Uma análise mais aprofundada resulta em $(3\log\log(m) + 2) \cdot c_r + 8\log\log(m) + 21$ unidades computacionais. Detalhes sobre a análise podem ser encontrados em [93] Existe ainda a possibilidade de utilizar-se um protocolo constante de rodadas para encontrar o valor máximo de um array (ver [93] para detalhes), o que resulta em uma operação de pivoteamento constante no número de rodadas ao custo de uma maior complexidade de comunicação, i.e., $\mathcal{O}(m^2 + n)$ comparações.

Observe que o número de operações é aproximadamente o mesmo do que no pior caso do algoritmo base calculando com variáveis em claro. O número de multiplicações é sempre linear no tamanho do *tableau*, enquanto o número de comparações é pelo menos linear em m e pode ser maior do que $m + n$. Além disto, as constantes são pequenas o que significa que a computação segura é essencialmente tão eficiente quanto esperado, apesar da possibilidade de pequenas melhorias, que não serão tratadas aqui, por questões de clareza.

O trabalho de Toft otimiza o número de rodadas, i.e., ele une todas as operações que podem ser executadas em paralelo e executa somente uma operação de comunicação. Para tal, assume-se que tantas operações quanto possível podem ser executas em paralelo, ou seja, em um número infinito de processadores e, com isso reduz-se a complexidade de computação ao nível da complexidade de rodada. No entanto, na prática dispõe-se de um número limitado de processadores, o que coloca os resultados de Toft em algum ponto intermediário entre uma implementação sequencial (pior caso) e o caso de paralelismo ilimitado, dependendo da disponibilidade de recursos.

3.6 Programação Linear em Paralelo

Algoritmos de programação linear e não-linear em paralelo já foram considerados pela literatura. Para Métodos de Ponto Interior [58, 96] cita-se a implementação orientada a objetos que executa em paralelo de [53]. A implementação em paralelo da Rotina de Minimização da Função Simplex de [62] propõe uma generalização do método simplex de *Nelder-Mead* [74] para resolução de programas não-lineares utilizando processadores em paralelo (há a possibilidade de transformar um problema de PL em um problema não-linear e então aplicar tal método). Para o caso do simplex de Dantzig [36], o Método Simplex Revisado em Parallelo de [90] traz um esquema de paralelização de colunas que funciona aplicando-se multiplicações matriz-vetor paralelas e reduções mínimas repetidamente, em cada iteração. O trabalho de [73] apresenta uma técnica de parallelização de linhas baseada na estrutura mestre-escravo para o algoritmo simplex de duas fases em paralelo. Em [41], Dong et al. introduzem o algoritmo BSP Simplex em Paralelo que possui custo computacional reduzido.

O que todas estas abordagens tem em comum é o fato de serem projetadas visando o aumento do desempenho sem nenhum requisito de segurança. Visto que estamos interessados em fornecer soluções práticas e seguras para problemas de otimização de cadeias logísticas, é necessário propor versões seguras de tais algoritmos paralelos. Depois de comparar as soluções listadas acima, considerando-se especialmente os requisitos de segurança, escolheu-se o método BSP Simplex em Paralelo de [41]. Dentre as razões para tal escolha encontram-se a independência da estrutura ou tipo de restrições para o modelo de paralelização e, extensão do modelo BSP para cenários de computação segura multi-parte.

3.6.1 BSP-based Parallel Simplex

O método simplex em paralelo baseado no model BSP foi introduzido por Dong et al. in [41] como uma alternativa para melhorar o tempo computacional em problemas de larga escala. O algoritmo paralelo é baseado no modelo *Bulk-Synchronous Parallel* (BSP) proposto por Valiant em [94].

Existem diversas razões para a escolha do método BSP simplex em paralelo dentre as outras possibilidades. O modelo BSP é adequado para CSM genérica (detalhes em como aplicar tal modelo a CSM serão descritos na Seção 6.2.2). A paralelização é independente da estrutura do problema ou tipo de restrições. O único requisito é de que o problema esteja em uma forma canônica (mesmo requisito de Toft e Li e Atallah), o que é aplicável aos problemas de PDCL. Em respeito ao desempenho, este modelo oferece um *speed-up* considerável. Considerável porque outras opções de paralelização como o Método Simplex Revisado em Paralelo de [90], ou o Método de Ponto-Interior em Paralelo de [53], oferecem *speed-ups* maiores, mas tal melhoria depende da estrutura do problema, densidade da matriz, identificação de estruturas-bloco, tipo de restrições e, tais propriedades não podem ser garantidas para qualquer problema de PDCL. A complexidade da solução é aproximadamente a mesma do método tradicional dividida pelo número de processadores, mais um custo de comunicação (que pode ser teoricamente estimado). Por fim, consideram-se os custos para uma implementação segura. O método BSP simplex em paralelo não depende da estrutura da matriz, e.g., o fato de que todos os elementos da matriz são variáveis secretas não invalida a solução como no caso de [90] e [53], por exemplo. Ele não incluí nenhum tipo de operações complexas com matrizes como inversão (e possivelmente decomposição LU). De maneira geral, este método apresenta aproximadamente o mesmo *overhead* que o protocolo de Toft para computação com variáveis secretas.

O modelo BSP argumenta por escalabilidade, portabilidade e previsibilidade, fornecendo novas fundações para o desenvolvimento de sistemas escaláveis de computação paralela [41]. Tal modelo encara uma máquina paralela como um conjunto de três elementos básicos:

1. conjunto de pares processador-memória;
2. rede de comunicação global que entrega mensagens de maneira ponto-a-ponto entre os processadores;
3. um mecanismo para a sincronização global de todos os processadores por meio de uma barreira;

Um programa BSP consiste em uma sequência de super passos que inclui computação simultânea local em cada processador, seguida de comunicação entre eles para a troca de informações e finalmente, uma barreira de sincronização que assegura que as ações de comunicação foram completadas com

sucesso. A Figura 3.1 mostra a estrutura de um super passo em algoritmos paralelos baseados no modelo BSP.

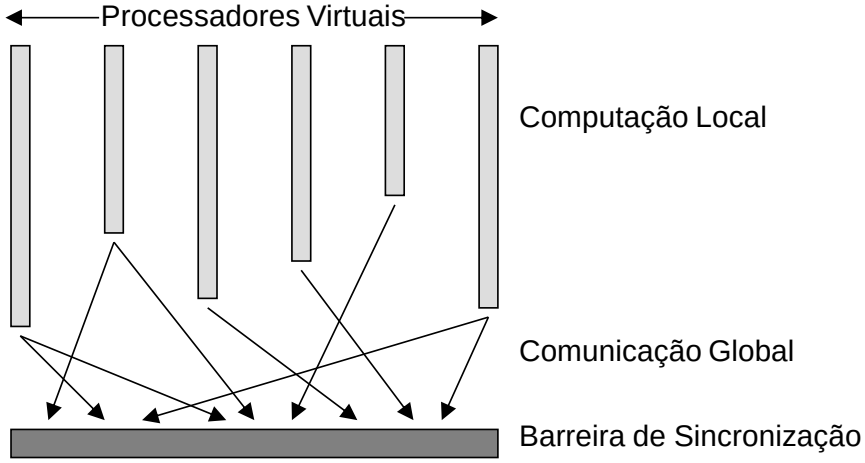


Figura 3.1: Estrutura de um super passo no modelo BSP

Problemas de PDCL (modelados como PL) e o algoritmo simplex tradicional exibem diversas operações que podem ser paralelizadas. O método BSP simplex em paralelo aplica uma técnica de paralelização em nível de tarefas, também chamada de paralelização horizontal ou de linhas, o que significa que cada processador possui um conjunto de linhas do *tableau* inicial. Os processadores são identificados por um identificador único, i.e., no intervalo $[0, (p-1)]$, onde p é o número de processadores disponíveis. Existe ainda uma distinção entre três classes de processadores:

processador 0 : responsável pela seleção da coluna e linha do elemento pivô;

processador min_L : possui a linha que contém o elemento pivô;

processador L : todos os outros processadores;

Dois dos passos do método simplex tradicional são paralelizados entre os processadores: *determinar a variável que sai da base* e a operação de *pivoteamento*.

Assume-se que o problema encontra-se em uma forma canônica e que está representado na forma de um *tableau* [36, 76, 95].

$$\begin{array}{l} minimize \;\; c \cdot x \\ sujeito\ a \;\; A \cdot x \leq b,\; x \geq 0 \end{array} \qquad \Rightarrow \qquad B = \begin{bmatrix} A & I & b \\ c & 0 & 0 \end{bmatrix}$$

A partir da forma padrão da matriz (*tableau*), o algoritmo calcula a matriz final contendo os valores ótimo para a função objetivo mediante a execução de basicamente os mesmos passos definidos pelo método simplex tradicional (referir-se a [41] para a descrição completa do algoritmo).

O custo de computação deste método é limitado por $\mathcal{O}(\frac{km(m+n)}{p} + k\epsilon + k\delta)$ contra $\mathcal{O}(km(m+n))$ do método simplex tradicional, onde m é o número de restrições, n é o número de variáveis, p é o número de processadores, k representa os saltos dos vértices, ϵ é o custo de comunicação de um salto entre vértices adjacentes e δ é o custo de sincronização. Na prática, de acordo com os experimentos de [41], o *speed-up* resultante para o caso de dois processadores é *1.65* e, para o caso de quatro processadores é *2.71*.

3.7 Conclusão

Transações de negócios em rede tem se tornado mais e mais comuns devido à larga utilização da Internet e arquiteturas orientadas a serviço. Aplicações B2B requerem que empresas se envolvam em

transações eletrônicas que, por sua vez, são vitais para o sucesso dos negócios. A indústria como um todo está apostando seu futuro nesta tendência e o gerenciamento de cadeias logísticas é um exemplo disto.

Os parceiros de negócio enquanto tendo acesso a sistemas internos se tornam uma ameaça de segurança. Mesmo sabendo que o compartilhamento de informação durante o gerenciamento de cadeias logísticas reduz custos, tais parceiros se mostram relutantes em compartilhar, principalmente devido ao medo do vazamento de dados. Neste capítulo foram discutidos o valor da informação no gerenciamento de CL assim como a necessidade de privacidade por parte dos parceiros. Entre as medidas de segurança mais comuns, CSM é a única a oferecer garantias prováveis de segurança, as quais podem convencer os parceiros a compartilhar.

Eficiência é a maior preocupação no que diz respeito à protocolos CSM e computação paralela pode ser uma alternativa para a redução de custos. Foram encontradas várias tentativas de paralelização de métodos de programação linear. A abordagem de Dong et al. (baseada em BSP) é a que mais se adequa às necessidades impostas por problemas de gerenciamento de CL e, portanto, foi descrita em detalhes.

Capítulo 4

Otimizando o Número de Permutações

Este Capítulo trata da necessidade de permutações seguras durante o protocolo *Secure and Private Collaborative Linear Programming* de [68]. A ocorrência de permutações em *cada iteração* torna este protocolo bastante caro. Sendo assim, é de extremo interesse a redução do número de permutações requeridas pelo protocolo proposto por Li e Atallah.

4.1 Introdução

A solução de Li e Atallah [68] não requer variáveis secretas (criptografadas) para a representação do problema. A utilização da técnica de compartilhamento aditivo dos dados entre os participantes, i.e., a matriz D, permite que a seleção do índice do elemento pivô seja feita em claro.

Entretanto, o fato de os participantes conhecerem tal índice eventualmente revela informações a respeito do problema original de modo desnecessário. Além disso, cada linha da matriz corresponde a uma restrição e cada uma das colunas corresponde a um certo coeficiente. Assim, se um índice de uma linha ou coluna for selecionado mais de uma vez, informação extra pode ser obtida, o que caracteriza o seu vazamento.

A fim de evitar vazamento de informação, um protocolo de embaralhamento e permutação é executado em *todas as iterações* do método simplex seguro. Isto garante a randomização dos índices antes da escolha do pivô de modo que nenhuma informação pode ser obtida, o que implica na preservação da privacidade dos dados dos participantes. Corretude é assegurada pela execução do protocolo de recuperação dos índices. Tal protocolo é invocado apenas uma vez, ao fim da execução do protocolo, com o intuito de restaurar as posições originais dos índices da matriz, ao custo de re-executar o protocolo de embaralhamento e permutação k vezes, no qual k é o número total de iterações.

Os custos computacionais de protocolos seguros de permutação para o caso de dois participantes já foram discutidos na literatura, em [68], por exemplo. Eles representam uma parcela importante dos custos totais do protocolo seguro de PL. Como exemplo, para o caso do protocolo de Li e Atallah, os custos da permutação segura são iguais aos custos da operação de pivoteamento, que é a operação mais cara de todo o protocolo. Problemas de gerenciamento de cadeias logísticas, e.g., plano diretor da cadeia logística, incluem diversos parceiros, o que requer soluções multi-parte. Permutações seguras para cenários multi-parte foram, pela primeira vez, formalmente consideradas em [61]. Os custos computacionais e de comunicação associados a tal operação dependem do número de participantes e, de maneira geral, são bastante elevados.

Em suma, a solução de Li e Atallah requer que o protocolo de embaralhamento e permutação seja invocado $2k$ vezes. O número de iterações k necessárias para a resolução de problemas de PL não pode ser modificado, i.e., ele depende do problema e varia de acordo com ele. A otimização de cadeias logísticas frequentemente envolve cenários multi-parte nos quais as cadeias possuem diversos nós, o que resulta em problemas de PL de tamanho considerável. Um problema de tamanho médio possui um número de restrições m aproximadamente igual a 2000 [36], por exemplo. A relação entre o número de iterações e o número de permutações, i.e., permutação em *cada* iteração, constitui um grande problema de desempenho. A redução do número de permutação é, portanto, de grande interesse.

Este Capítulo é divido como segue. Uma análise probabilística das razões pelas quais a permutação é necessária é feita na Seção 4.2. A Seção 4.3 introduz uma idéia baseada em probabilidade para a

redução do número de permutações e discute aspectos conceituais. Uma descrição matemática formal é realizada na Seção 4.4. A Seção 4.5 fornece uma avaliação da proposta baseada em experimentos. Os resultados e trabalhos relacionados são discutidos na Seção 4.6 e, por fim, a Seção 4.7 resume as conclusões do Capítulo.

4.2 Distribuição Probabilística da Repetição de Índices de Linhas ou Colunas em Problemas de PL

Existem três fontes principais para vazamento de informação quando se considera protocolos nos quais a escolha do índice do elemento pivô é feita em claro:

1. seleção de dois ou mais elementos pivô na mesma linha;
2. seleção de dois ou mais elementos pivô na mesma coluna;
3. seleção de dois ou mais elementos pivô exatamente na mesma posição;

A solução de problemas de PL utilizando-se simplex eventualmente resulta em um número de iteração maior do que o número de variáveis ou de restrições. Não é possível se garantir que uma linha ou coluna, em particular, não seja escolhida mais de uma vez durante a execução do protocolo (simplex). A seleção consecutiva de dois elementos pivôs na mesma coluna não é permitida pelo protocolo. O algoritmo simplex assegura que depois de uma operação de pivoteamento todos os elementos da coluna pivô são iguais a zero, exceto pelo próprio elemento pivô (detalhes em [35]). Entretanto, linhas podem ser escolhidas mais de uma vez, consecutivamente.

Permutações são então utilizadas para embaralhar os índices da matriz D antes da seleção de cada elemento pivô. Isso garante que tal seleção seja uma escolha randômica na matriz permutada, de modo que o conhecimento de tal índice seja irrelevante, no que diz respeito ao ganho de informação sobre o sistema.

A fim de reduzir o número de permutações se faz necessário, inicialmente, determinar a distribuição de probabilidades de que linhas ou colunas se repitam durante a execução do algoritmo simplex. Note que o terceiro caso relacionado acima, i.e., seleção de dois ou mais elementos pivô exatamente na mesma posição, é uma combinação dos casos 1 e 2.

O procedimento adotado consiste em projetar e implementar um simulador de problemas aleatórios de PL, mais especificamente, problemas de PDCL, nosso caso de uso. Uma cadeia logística randômica é gerada a partir de um número específico de estágios, nós e ligações e o problema é automaticamente modelado (matematicamente), de acordo com a teoria de programação linear apresentada no Capítulo 3. A entrada para o simulador consiste no número de estágios s e no número de nós n, e a saída consiste em uma representação da cadeia logística com n nós aleatoriamente distribuídos em s estágios com as ligações correspondentes. Cada nó representa um participante (parceiro da cadeia), i.e., fornecedor, consumidor, etc., e é matematicamente modelado como uma função objetivo e um conjunto de restrições. O simulador então agrega as funções de todos os nós e gera um único sistema linear, na forma canônica, representado por uma matrix, i.e., a matriz D de Li e Atallah, como saída. Os valores referentes às entradas privadas dos participantes, i.e., as constantes do sistema, são aleatoriamente gerados, respeitando-se as restrições e assegurando-se que o PL resultante esteja na forma canônica.

Uma ferramenta (*software*) que implementa os passos descritos pelo algoritmo simplex de [36] é utilizada para a resolução do PL gerado (tal ferramenta não foi desenvolvida durante os trabalhos, visto existirem diversos exemplos na Internet). O programa linear resultante, na forma canônica, serve de entrada para a ferramenta. A saída corresponde aos valores ótimos para a função objetivo gerada, e.g., minimizar os custos totais do plano diretor da cadeia logística.

Várias configurações diferentes para cadeias logísticas foram simuladas. Assuma configurações definidas em termos do número de estágios e de nós como (s, n). Os valores para o número de estágios são $s = \{3, 4, 5, 6\}$. Os valores para o número de nós seguem a seguinte expressão: $\{(s, n = s), (s, n = 2s), (s, n = 3s), (s, n = 4s), (s, n = 5s)\}$, para todo s. Para $s = 3$, o conjunto de configurações utilizadas é $\{(3, 3), (3, 6), (3, 9), (3, 12), (3, 15)\}$, onde $(3, 3)$ poderia representar (hipoteticamente) uma cadeia com um fornecedor, uma empresa de manufatura e um consumidor, por exemplo.

O simulador não é capaz de gerar cadeias logísticas para as quais a solução (utilizando-se o método simplex) cobre todos os possíveis valores para o número de iterações. Como exemplo, nenhum dos problemas gerados foi resolvido em exatamente 3 iterações. Em virtude disto, uma função linear de interpolação foi utilizada para a geração dos valores intermediários.

A Figura 4.1 traz os resultados da distribuição de probabilidades de repetição de linhas ou colunas como uma média aritmética da execução de 100 problemas para cada uma das configurações para o caso de 3 estágios.

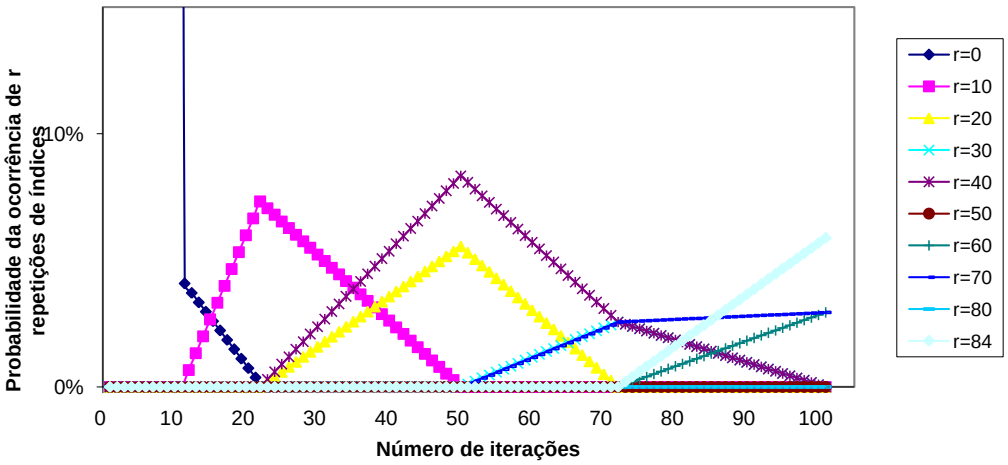


Figura 4.1: Distribuição de probabilidades para a repetição de índices em linhas ou em colunas

O número de repetições r, incluindo linhas e colunas, varia de 0 a 84, para este caso em particular. A frequência da probabilidade de repetição de índices com valores baixos diminui ao passo que o número de iterações aumenta, e.g., $P(r = 0)$. Por outro lado, a frequência da probabilidade de repetição de índices com valores altos aumenta ao passo que o número de iterações também aumenta. Além disso, a taxa na qual o número de repetições de índices (de linhas ou colunas) cresce é muito menor do que a taxa na qual o número de iterações cresce.

A probabilidade de repetição de linhas ou colunas nunca excede 10%, exceto para $r = 0$, que é o caso no qual não há repetição. Baseado nestes fatos, conclui-se que a solução apresentada em [68] utiliza um política de número de permutações bastante restritiva, no que diz respeito ao desempenho. Os custos poderiam ser reduzidos simplesmente permutando antes da primeira iteração e somente quando uma linha ou coluna estivesse para ser repetida. No entanto, esta abordagem revela o número exato de repetições ocorridas durante a execução do algoritmo. Esta informação poderia ser utilizada por algum parceiro malicioso para inferir sobre os valores da matriz original, desrespeitando a privacidade dos dados. Os requisitos de privacidade demandam, portanto, uma solução mais elaborada.

4.3 Conceito

Introduz-se, então, o conceito de uma *moeda* (jogar a moeda) para a preservação da privacidade das entradas e a redução do número de permutações necessárias durante a execução do protocolo seguro de PL.

A moeda é jogada antes de cada iteração, com uma certa probabilidade, e o resultado decide se o protocolo irá ou não permutar a matriz do sistema nesta iteração. Assumem-se duas possíveis razões para permutação: (i) um índice está para ser repetido (requisito do problema de PL); e (ii) a moeda decidiu que uma permutação é necessária (requisito artificialmente introduzido). A distribuição de probabilidades de (i) pode ser extraída de experimentos práticos, como os mostrados na Seção 4.2, e a distribuição de probabilidades de (ii) pode ser matematicamente estimada.

Parceiros que executam um protocolo de PL seguro de maneira colaborativa são capazes de observar duas variáveis: (i) o número de iterações; e (ii) o número de permutações (caso seja diferente de (i)). Se a probabilidade da moeda for estimada de modo que um participante não consiga distinguir a exata razão de uma dada permutação, i.e., se devido a repetição de índices ou devido à decisão da moeda, nenhuma informação adicional poderá ser inferida a respeito do número real de índices

repetidos. Quanto menor for esta probabilidade, menor será o número de permutações executadas, e consequentemente, melhor será o desempenho do protocolo seguro de PL. Note que, mesmo sendo impossível distinguir a razão pela qual o protocolo permuta, uma pequena quantidade de informação ainda pode ser obtida, i.e., o número de permutações, mas muito menor do que se nenhuma moeda fosse usada.

Desta forma, se faz necessário encontrar a menor probabilidade para o lançamento da moeda que revelará aproximadamente a mesma quantidade de informação que pode ser obtida quando se observa um certo número de iterações e permutações, ou quando se permuta em todas as iterações (abordagem proposta por Li e Atallah).

4.4 Análise Matemática

A distribuição de probabilidades referente à repetição dos índices de linhas ou colunas já é conhecida (Seção 4.2). Deseja-se estimar a menor probabilidade para a moeda que garanta a privacidade das entradas, isto é, melhore o desempenho do protocolo seguro de PL revelando o mínimo possível de informação. Para tal, introduz-se a seguinte idéia:

> *Quantificar o número de bits necessários para representar todas as possíveis matrizes obtidas a partir da observação de um certo número de permutações e iterações durante a execução de um protocolo de PL, definido como NoB.*

O valor de NoB deve ser máximo quando se permuta em todas as iterações, ou seja, mínimo de informação revelada, e deve ser mínimo quando se permuta somente onde há repetição de índices (seja por causa de repetição de linhas ou de colunas). Existe um ponto no qual a combinação de ambas as probabilidades, i.e., da moeda e da real repetição de índices, resulta em um valor para NoB que é tão perto quanto possível do valor máximo. Nosso argumento baseia-se no fato de que tal ponto é consideravelmente menor do que o número de iterações, o que resulta em uma redução no número de permutações e, consequentemente, em uma melhoria na performance do protocolo seguro de PL como um todo (para o caso no qual a escolha de índices é feita em claro).

A probabilidade de observarem-se m permutações e l iterações é formalmente representada pela Equação 4.1,

$$P(pr = m, i = l) = P(r = n, i = l) \cdot P(c = m - n, i = l) \tag{4.1}$$

onde $P(r = n, i = l)$ é a probabilidade de observar-se n repetições (devido à repetição de índices de linhas ou de colunas) e l iterações, obtida experimentalmente. A probabilidade de observar-se permutações devido a decisões da moeda e l iterações é determinada por $P(c = m - r, i = l)$. O termo $P(c = m - r, i = l)$ é definido como uma função da probabilidade da moeda, do número de iterações, de repetições e de permutações e pode ser formalmente expresso pela Equação 4.2,

$$P(c = m - n, i = l) = C_{m-n}^{l-n} \cdot p^{m-n} \cdot (1 - p)^{l-m} \tag{4.2}$$

com

$$C_{m-n}^{l-n} = \binom{l-n}{m-n} = \frac{(l-n)!}{(m-n)!(l-m)!} \tag{4.3}$$

onde l é o número de iterações, m é o número de permutações, n é o número de repetições e p denota a probabilidade da moeda.

A Equação 4.2 pode ser decomposta em três termos: (i) a combinação dos possíveis locais restantes onde uma permutação ainda pode ocorrer, excluindo-se os locais onde ela já ocorreu, devido à repetição de índices ou decisão da moeda (Equation 4.3); (ii) a probabilidade da moeda, em si; e (iii), a probabilidade de que a moeda indique que não há necessidade de permutação. De maneira geral, quanto maior for o número de repetições ou permutações, menor serão as chances de ocorrerem permutações devido à decisão da moeda. Também espera-se que à medida que a probabilidade p aumenta, a probabilidade $P(c = m - r, i = l)$ também aumente.

Colocando todas as informações juntas, o número de bits necessário para representar as matrizes obtidas pela observação de um certo número de permutações e iterações é quantificado pela Equação 4.4:

$$NoB = \sum_{m=0}^{l_{max}} \left(\sum_{l=0}^{l_{max}} \left(\sum_{n=0}^{m} P(pr = m, i = l) \cdot \log \left(\sum_{n=0}^{m} N \cdot P(r = n, i = l) \right) \right) \right) \tag{4.4}$$

onde N é o número de bits necessário para representar a matriz do sistema.

Espera-se que o valor de NoB seja máximo quando $p = 1$, o que significa que a moeda decidiu permutar em todas as iterações (exatamente como no caso de Li e Atallah). Por outro lado, espera-se que o valor de NoB seja mínimo exatamente quando $p = 0$, i.e., quando todas as permutações foram causadas pela repetição de um índice, seja ele de uma linha ou de uma coluna.

4.5 Prova do Conceito

Primeiramente, a Equação 4.4 precisa ser validada. Como exemplo, considere dois casos distintos para o número de iterações l: (i) l assume um valor pequeno; e (ii) l assume um valor grande. No caso (i), para um valor pequeno e fixo de l, a probabilidade de ocorrerem repetições – $P(r = n, i = l)$ – decresce ao passo que o número de repetições aumenta. Já no caso (ii), para um valor grande e fixo de l, o valor de $P(r = n, i = l)$ aumenta ao passo que o número de repetições aumenta. Em ambos os casos, a Equação 4.4 se comporta como esperado.

Agora é necessário calcular NoB utilizando valores experimentais para $P(r = n, i = l)$ e variações na probabilidade da moeda p. Assuma $N = 8$ (número arbitrário de bits necessários para representar a matriz do sistema) e o número de iterações l variando no intervalo $0 \geq l \geq l_{max}$, onde $l_{max} = 101$ com um passo de tamanho igual a 1 (dados da Figura 4.1). A probabilidade da moeda varia de acordo com o intervalo $0 \leq p \leq 1$, com um passo de tamanho igual a 0.1. A Figura 4.2 mostra o resultado em termos do vazamento de informação, em valores percentuais, para um dado valor de probabilidade da moeda. Note que o valor de NoB é proporcionalmente inverso ao valor de cada ponto mostrado na Figura 4.2.

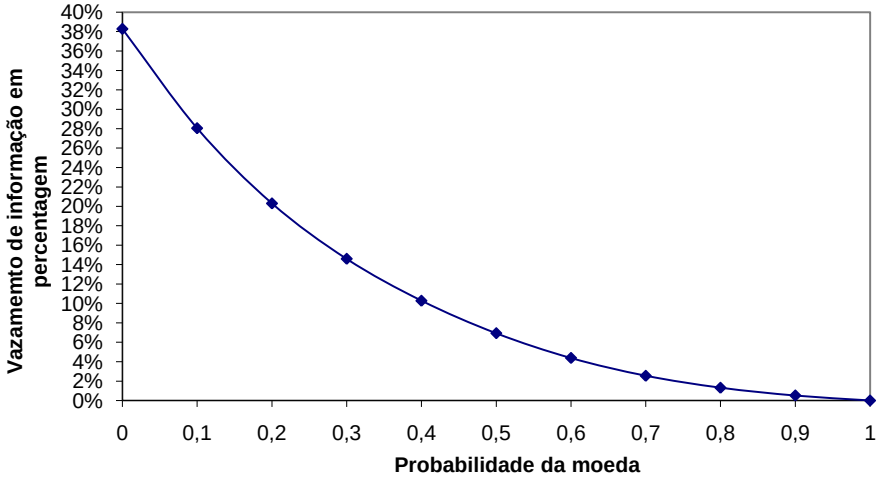


Figura 4.2: Porcentagem de informação que vaza devido à repetição de índices durante a escolha do elemento pivô

O valor de NoB aumenta quando a probabilidade da moeda aumenta. Quanto maior o valor referente à probabilidade da ocorrência de permutação (seja devido à repetição de índices ou à decisão da moeda), mais difícil é de se representar as matrizes obtidas pela observação de um certo número de permutações e iterações, menos informação é revelada (ver Figura 4.2) e, com isso, a privacidade dos dados é preservada. Se $p = 1$, o que indica que a moeda decide pela ocorrência de permutações em todas as iterações (caso equivalente a [68]), nenhuma informação é revelada. Se, por outro lado, $p = 0$, o que indica que todas as pemutações ocorreram devido a repetições de índices, NoB assume seu valor mínimo e aproximadamente 38% do total de bits é revelado.

Os experimentos revelaram que a introdução do esquema da moeda faz com que não seja mais necessário que ocorram permutações em todas as iterações. A introdução do conceito da probabilidade da moeda reduz as constantes do protocolo de permutação ($\sim 40\%$) e, em virtude disto, melhora o desempenho do protocolo como um todo, para casos práticos. O resultado mais importante está no fato de que a probabilidade da moeda é capaz de controlar a relação privacidade *versus* custos computacionais.

A implementação da fórmula para o cálculo do NoB bem como os resultados referentes à distribuição de probabilidades de repetição de índices não foram formalmente validados (prova matemática formal). Como os resultados experimentais dependem da implementação realizada, não se pode garantir a veracidade dos resultados experimentais apresentados, fato que de modo algum os invalida nem mesmo diminui a importância dos trabalhos realizados.

4.6 Resultados e Considerações

Algoritmos de PL com seleção em claro dos índices do elemento pivô requerem protocolos especiais para preservação da privacidade das entradas. A melhor solução conhecida foi proposta por Li e Atallah em [68]. Tal solução aplica o protocolo de embaralhamento e permutação em *cada* iteração do simplex seguro a fim de esconder a posição real dos índices na qual a operação de pivoteamento será realizada.

Os custos para uma permutação (segura), mesmo considerando-se o caso mais simples, i.e., com dois participantes, são bastante altos. De acordo com [68], o protocolo de embaralhamento e permutação tem a mesma complexidade computacional que a operação de pivoteamento, por exemplo. Adicionalmente, o protocolo de recuperação de índices requer que k extra permutações seguras sejam executadas (onde k representa o número de iterações do método simplex seguro). Considerando-se um cenário multi-parte, os custos são ainda maiores. Os trabalhos de [61] consideram algumas abordagens para permutação segura multi-parte. Até então, não existe solução que possa ser executa em um número constante de rodadas, mas se observa uma dependência do número de participantes. Diversas outras contribuições na literatura discutem os custos relacionados a outras operações multi-parte, como, por exemplo, comparação e multiplicação [29, 33, 60]. Mesmo que já existam soluções que executem em um número constante de rodadas (exemplos acima), a mesma dependência do número de participantes (ou de uma parte deles) é sempre observada na complexidade computacional.

Desempenho prático é uma das maiores preocupações quando se fala em protocolos de CSM. Muito esforço tem sido realizado nesta direção de pesquisa e pode-se dizer que a construção de protocolos factíveis na prática ainda é considerada um desafio. Por tais razões, a redução do número de permutações é de extremo interesse, especialmente na prática. Este Capítulo tratou da introdução de um esquema, baseado na probabilidade de uma moeda, para a redução do número de permutações em protocolos seguros de LP nos quais a escolha do índice do elemento pivô é realizada em claro. Não é necessário que ocorram permutações em todas as iterações, mas devido à introdução do esquema proposto, observa-se uma redução no número de permutações, o que definitivamente melhora o desempenho de tais protocolos, em termos práticos.

Em relação à privacidade da solução combinada, i.e., *secure and private collaborative linear programming* de [68] com o esquema da moeda para redução do número de permutações, alguns aspectos devem ser discutidos. Primeiramente, considera-se a existência de uma relação de compromisso (*trade-off*) entre otimização e privacidade da solução. Li e Atallah propõem uma solução na qual nenhum dos participantes revela informações a respeito de suas entradas, exceto aquelas que podem ser deduzidas a partir dos resultados, com uma complexidade computacional proporcional à complexidade de tempo do método simplex tradicional [36]. Tal solução (e análise de complexidades) é somente válida para cenários com dois participantes. Soluções para cenários multi-parte requerem um esforço computacional e de comunicação muito maior. Duas soluções para este tipo de cenário são discutidas em [61] como resultado dos trabalhos descritos no Capítulo 5 desta dissertação.

A introdução do esquema da moeda diminui consideravelmente os esforços computacionais e de comunicação ao passo que reduz o número de permutações. Esta afirmação é suportada pela análise do número de matrizes a serem excluídas. Usa-se uma medida de informação teórica devido à inclusão do protocolo, i.e., 4% dos *bits* da matriz podem ser supostos. Obviamente, a escolha do limiar para

um vazamento aceitável depende do cenário em que o protocolo será utilizado. Dessa maneira, nosso intuito aqui é somente dar um exemplo de como o esquema da moeda poderia ser aplicado. Em última análise, a privacidade da solução pode ser controlada pela probabilidade da moeda, definida *a priori*.

Um segundo aspecto a ser considerado é a posição (iteração) na qual as permutações ocorrem. Qualquer solução que reduza o número de permutações frente ao número de iterações estará ao mesmo tempo revelando as posições onde tais permutações ocorreram. O esquema da moeda proposto aqui poderia ser extendido a fim de que as probabilidades incluam mais esta informação. Entretanto, acredita-se que a curva resultante não será demasiadamente diferente da curva atual (Figura 4.2) e ainda representará uma melhoria considerável no desempenho.

Por último, é importante ressaltar que, durante a execução do protocolo, nenhum dos participantes conhece a probabilidade da moeda. Isto pode ser conseguido porque a moeda é lançada de maneira conjunta, utilizando-se um protocolo de lançamento de moeda padrão, no qual nenhum dos participantes é capaz de determinar o resultado sozinho. Exemplos de protocolos como esse podem ser encontrados em [88].

4.7 Conclusão

Este Capítulo considerou o problema de programação linear colaborativa, segura e com preservação de privacidade para gerenciamento de cadeias logísticas, e.g., o problema de PDCL e os aspectos de desempenho relacionados ao número de permutações. A melhor solução conhecida na qual o índice do elemento pivô é selecionado em claro foi proposta em [68], e possui a desvantagem de exigir a execução do protocolo de embaralhamento e permutação em *todas* as iterações do algoritmo simplex.

Neste Capítulo foram identificadas as razões pelas quais existe a necessidade de permutação, sendo elas a repetição de índice em linhas ou em colunas. Foi realizado um estudo da probabilidade da ocorrência de tais eventos durante a execução do método simplex e conlui-se que não é necessário que se permute em todas as iterações. Sendo assim, introduziu-se um esquema baseado na probabilidade de uma moeda para a redução do número de permutações. O conceito proposto foi descrito e uma análise matemática detalhada foi desenvolvida, juntamente com uma avaliação baseada em resultados experimentais. Fundamentalmente, a probabilidade da moeda é capaz de controlar a relação entre desempenho e privacidade de um dado método simplex seguro, no que diz respeito ao número de permutações. Devido à introdução do esquema proposto, pode-se observar uma redução no número de permutações, o que definitivamente melhora a permformance do protocolo seguro de PL enquanto garante a privacidade dos dados.

Capítulo 5

Otimizando a Complexidade de Protocolos de Permutação Multi-Parte

Li e Atallah [68] propõem um protocolo seguro de PL para o caso de duas partes. Problemas de gerenciamento de cadeias logísticas requerem soluções multi-parte. Este Capítulo discute duas soluções seguras para protocolos de permutação multi-parte que são uma importante sub-operação de protocolos seguros de PL.

5.1 Introdução

Otimização de cadeias logísticas é uma tarefa bastante complexa e requer uma quantidade significativa de recursos mesmo quando executada de maneira não-segura. Protocolos seguros multi-parte como os de Li e Atallah [68] e Toft [93], ou combinações de protocolos para operações elementares como o de Cramer [29], de Damgard [32, 33], de Beaver [7], de Bogetoft Et Al. [9], de Du e Zhan [42], de Ueli [71] são capazes de evitar que as entradas sejam reveladas inadvertidamente e podem, portanto, calcular de maneira segura problemas de otimização de CL.

O desafio de pesquisa mais importante é a construção de soluções seguras de PL que sejam realizáveis na prática. Para tal, dois aspectos precisam ser considerados. Primeiramente, a construção teórica na qual os problemas são decompostos e analisados. Técnicas especiais podem ser utilizadas a fim de otimizar a complexidade assintótica de um dado protocolo em respeito à construção genérica de circuitos.

Existem duas complexidades que podem ser otimizadas: complexidade de computação e complexidade de comunicação. A primeira se refere ao esforço máximo que um participante precisa fazer durante a execução de um protocolo em passos de computação. Complexidade de computação pode ser medida e número de exponenciações modulares, que são as operações mais custosas. Complexidade de comunicação se refere ao número total de unidades de informação trocadas durante a execução de um protocolo. Comunicação, por sua vez, possui um outro aspecto a ser analisado, chamado de complexidade de rodada, i.e., o número de rodadas necessárias para executar um determinado algoritmo em um sistema distribuído síncrono. A complexidade de rodada normalmente domina o tempo absoluto de comunicação.

O segundo aspecto diz respeito à performance, na prática, de um dado protocolo. A análise teórica é verificada e as constantes são expostas. Os protocolos são implementados e os custos reais são determinados por meio de avaliações de desempenho.

O Capítulo 3 descreveu duas soluções para otimização segura de CL e o Capítulo 4 demonstrou a importância da redução dos custos referentes à execução de permutações seguras em soluções nais quais o índice do elemento pivô é selecionado em claro. Otimização de CL usualmente envolve múltiplos participantes com funções objetivo diferentes, cada qual utilizando suas próprias informações privadas. A interação entre os participantes é notavelmente um elemento chave em computações seguras distribuídas. Complexidade de rodada é definida em termos de tais interações e constituí-se em uma das maiores barreiras à construção de protocolos seguros práticos [6]. Além do mais, o cenário de mercado global e operações a nível mundial resultam em cadeias logísticas nas quais os parceiros encontram-se geograficamente dispersos, de modo que aplicações reais incluem atrasos na comunicação.

Complexidade de rodada é diretamente afetada por condições específicas de rede, como atraso (*delay*).

Protocolos para multiplicação e comparação segura multi-parte com custos reduzidos já foram propostos e discutidos pela literatura. Exemplos incluem [29, 33, 46]. No entanto, protocolos práticos e seguros de permutação multi-parte para otimização de cadeias logísticas foram pela primeira vez considerados por Kerschbaum e Deitos em [61] (resultados baseados nos trabalhos descritos neste Capítulo). Este Capítulo tem o objetivo de introduzir dois protocolos seguros de permutação multi-parte. O primeiro consiste em uma extensão direta do protocolo proposto por [68] com complexidade linear de rodada. Visto que CL reais incluem atrasos de comunicação, um segundo protocolo, com complexidade logarítmica de rodada, é introduzido como uma tentativa de reduzir dos efeitos negativos das condições da rede, para aplicações práticas.

Este capítulo está assim dividido: a Seção 5.2 define a notação utilizada para a construção dos protocolos. As Seções 5.3 e 5.4 introduzem as duas soluções propostas para permutação segura multi-parte. A Seção 5.5, por sua vez, realiza uma comparação teórica das soluções, no que diz respeito às complexidades. Implementação e desempenho prático dos protocolos são tratados na Seção 5.6. Discussão sobre os resultados e conclusões são apresentadas na Seção 5.7 e Seção 5.8, respectivamente.

5.2 Notação para a Construção dos Protocolos

Considere um cenário multi-parte no qual p parceiros desejam gerar, de maneira colaborativa, o plano diretor da cadeia logística utilizando programação linear segura. Tal problema é modelado como um PL na forma canônica [35] e representado por uma matriz com $(m+1)$ linhas e $(n+1)$ colunas, de modo similar ao proposto por [68]. Seja $[D]$ a matriz do sistema.

Cada participante i $(1 \leq i \leq p)$ possui uma permutação secreta e individual para linhas e uma para colunas, para cada iteração do PL seguro. Considera-se neste trabalho duas maneira diferentes para se representar permutações: notação de vetores ou matrizes [10].

A maneira mais natural de representar-se permutações é por meio de vetores. Seja x um vetor de tamanho η representado como $x = \{x_0, x_1, \cdots, x_\eta\}$. A permutação π de η elementos é um mapa dos elementos de tal conjunto sobre outros elementos do mesmo conjunto, i.e., $\pi : \{1, \ldots, \eta\} \rightarrow \{1, \ldots, \eta\}$. Em outras palavras, uma permutação descreve as novas posições de tais elementos no conjunto resultante. Como exemplo, considere $x = \{x_0, x_1, x_2, x_3\}$ e $\pi = \{3, 1, 0, 2\}$ de modo que $\pi(x) = \{x_3, x_1, x_0, x_2\}$. Vetores privados de permutação para linhas e colunas pertencentes ao participante i são então denotados por π_r^i e π_c^i, respectivamente.

Alternativamente, matrizes quadradas $(0, 1)$ com exatamente uma entrada igual a 1 em cada linha e em cada coluna, e 0's (zeros) em todas as posições restantes, podem ser utilizadas para representar permutações. Dada a permutação π definida acima, sua matriz de permutação é a matriz P_π, de tamanho $\eta \times \eta$, para a qual as entradas são todas iguais a 0 exceto na linha i, na qual a entrada $\pi(i)$ é igual a 1. Matrizes de permutação são denotadas como segue:

$$P_\pi = \begin{bmatrix} \mathbf{e}_{\pi(1)} \\ \mathbf{e}_{\pi(2)} \\ \vdots \\ \mathbf{e}_{\pi(\eta)} \end{bmatrix}, \text{ e.g., } P_\pi = \begin{bmatrix} 010\cdots0 \\ 100\cdots0 \\ \ddots \\ 000\cdots1 \end{bmatrix}_{\eta\times\eta}$$

onde $\mathbf{e}_j$ é o vetor linha de tamanho η com valor igual a 1 na j-ésima posição e valor igual a 0 em todas as demais.

Permutações são funções bijetivas e o produto de duas permutações corresponde a composição das mesmas, como funções. De maneira semelhante, bijeções possuem inversas e como tal, permutação também o fazem. Ambas as funções $P \circ P^{-1}$ e $P^{-1} \circ P$ são equivalentes à *permutação identidade*, que resulta em todas as posições dos elementos da matriz inalteradas e que, em virtude disto, pode ser utilizada para a construção de protocolos de recuperação de índices.

Um protocolo de permutação multi-parte genérico pode ser definido em termos de suas entradas e saídas. Os participantes fornecem suas permutações secretas como entradas, representadas como vetores ou matrizes. A saída do protocolo é a matriz do sistema permutada de acordo com a combinação de todas as permutações individuais (e secretas) dos participantes. A execução do prototolo é tal que

nenhum participante é capaz de inferir informações a respeito das permutações secretas de outros, exceto pelo que pode ser deduzido a partir dos resultados.

Considere uma fase de inicialização onde cada participante recebe (ou gera utilizando um esquema de GDC, como introduzido no Capítulo 2) uma chave pública pk e um segredo compartilhado sk_i (para o participante i) de um sistema criptográfico homomórfico com limiar. Para este trabalho utilizou-se uma implementação do sistema criptográfico com limiar de Paillier [77] já descrito na Seção 2.3, baseada nos resultados de [57]. O limiar é denotado por t, de modo que são necessárias pelo menos $t+1$ partes do segredo (ou participantes) para decifrar um dado texto criptografado [57]. Para o caso semi-honesto, o valor do limiar é definido como $t < p-1$, onde p é o número de participantes.

Seja $[\cdot]_{pk}$ a representação de uma cifragem aleatória de um elemento qualquer utilizando-se a chave pública pk. Com $[A]_{pk}(i)$ denota-se a cifragem aleatória do i-ésimo elemento do vetor A, e com $[B]_{pk}(i,j)$ denota-se a cifragem aleatória do elemento na i-ésima linha e j-ésima coluna da matriz $[B]$. Para facilidade de explicação, de agora em diante, a referência à chave pública será omitida e simplesmente se escreverá $[B]$, por exemplo.

Como matrizes de permutação são secretas (privadas), introduz-se a noção de matriz de permutação criptografada, que nada mais é do que a forma criptografada de uma matriz de permutação (definida acima). Isto implica que cada participante precisa gerar seu próprio par de matrizes de permutação criptografadas para linhas e colunas, i.e., o participante inicialmente gera um matriz de permutação em claro e posteriormente, utilizando a chave pública pk distribuída durante a fase de inicialização, cifra todos os elementos da matriz. A partir de agora, por clareza e simplicidade, o termo matriz de permutação será utilizado para denotar matriz de permutação criptografada.

O tamanho das mensagens trocadas é definido em função do tamanho do texto criptografado, que depende do sistema criptografico utilizado. Para este trabalho, considera-se um espaço de textos criptografados $c \in \mathbb{Z}_{mod^{s+1}}$ onde mod é o módulo e s é o parâmetro de segurança do sistema criptográfico usado. Por simplicidade, os resultados teóricos apresentados aqui não consideram o *overhead* resultante da escolha de uma determinada tecnologia de comunicação.

5.3 Permutação Multi-Parte: Algoritmo Linear de Rodada

O algoritmo linear de rodada é uma extensão direta para o cenário multi-parte do protocolo de embaralhamento e permutação de [68]. O Algoritmo 1 contém uma descrição detalhada do procedimento utilizado. Tal algoritmo é invocado em todas as iterações do protocolo seguro de PL.

Assuma que cada participante i, para $1 \geq i \geq p$, possui suas permutações secretas para linhas e colunas (π_r^i e π_c^i), respectivamente. Os participantes então embaralham e permutam a matriz do sistema, um após o outro, primeiro linhas e depois colunas e enviam a matriz permutada para o próximo participante. O último participante realiza um *multicast* da matriz resultante para todos os outros participantes, como demonstrado na Figura 5.1.

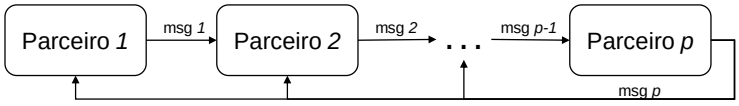


Figura 5.1: Mensagens trocadas durante a execução do Algoritmo 1

A matriz resultante é permutada pela combinação das permutações individuais e secretas de todos os participantes, i.e., $\pi_c^p(\pi_r^p(...(\pi_c^1(\pi_r^1(D)))))$. Observe que a ordem em que as permutações são aplicadas é importante e precisa ser respeitada, e.g., $\pi_c^i \circ \pi_r^i(\cdot)$, o que implica em permutar primeiro linhas e depois colunas. As primeiras $(p-1)$ mensagens são enviadas em sequência, de participante para participante. O último participante (de índice p) então envia uma mensagem para cada um dos outros contendo a matriz resultante.

O processo de embaralhamento corresponde a somar a todos os elementos da matriz o valor referente ao resultado da cifragem homomórfica aleatória do número zero, o que permite que a permutação seja aplicada em claro. Portanto, os próximos participantes não são capazes de inferir sobre as permutações anteriores ou sobre as posições dos elementos da matriz, o que implica na preservação da privacidade. O processo de embaralhamento é realizado utilizando-se o protocolo MixAndPermute

Algorithm 1: LinearRoundPerm

Input: The encrypted system matrix $[D]_{pk}$ and a matrix $[I]_{pk}$ with same size as D
Output: Both input matrices permuted on rows and columns according to this party's private permutations

1 $\pi_r^i \leftarrow$ generate new random permutation with size $m+1$;
2 $\pi_c^i \leftarrow$ generate new random permutation with size $n+1$;
3 $[D']_{pk} \leftarrow$ MixAndPermute$([D]_{pk}, \pi_r^i, \pi_c^i)$;
4 $[I']_{pk} \leftarrow$ MixAndPermute$([I]_{pk}, \pi_r^i, \pi_c^i)$;
5 **if** *not last party* **then**
6 | **send** $[D']_{pk}$ and $[I']_{pk}$ to the next party;
7 **else**
8 | **multicast** $[D']_{pk}$ and $[I']_{pk}$;
9 **end**

(descrito pelo Algoritmo 2) – o símbolo $\oplus$ representa uma adição homomórfica. As operações das linhas 5-10 incluem duas transposições de matrizes, que são omitidas por efeito de clareza, visto que não afetam a complexidade total do algoritmo.

Algorithm 2: MixAndPermute$([A]_{pk}, \pi_r, \pi_c)$

Input: Encrypted matrix $[A]_{pk}$ and two permutation arrays π_r and π_c for rows and columns, respectively
Output: The input matrix mixed and permuted according to the given permutations

1 **forall** *elements in matrix* $[A]_{pk}$ **do**
2 | $[0]_{pk} \leftarrow Enc_{pk}(0)$; `random homomorphic encryption of zero`
3 | $[A']_{pk}(i,j) \leftarrow [A]_{pk}(i,j) \oplus [0]_{pk}$; `homomorphic addition`
4 **end**
5 **for** $0 \geq r \geq m$ **do** *all rows in* $[A']_{pk}$
6 | $[A''](r, \bullet) \leftarrow \pi_r([A'](r, \bullet))$;
7 **end**
8 **for** $0 \geq c \geq n$ **do** *all columns in* $[A'']_{pk}$
9 | $[A'''](\bullet, c) \leftarrow \pi_c([A''](\bullet, c))$;
10 **end**
11 **return** $[A''']_{pk}$;

Um efeito colateral da permutação dos elementos da matriz é que os índices das variáveis básicas também são permutados. Em outras palavras, os participantes precisam encontrar as variáveis correspondentes na matriz original. A matriz I é uma matriz auxiliar utilizada para o protocolo de recuperação dos índices. Assuma que o primeiro participante receba tal matriz como a matriz identidade criptografada com a chave pública pk. Os participantes procedem de maneira exatamente igual, embaralhando e permutando a matriz $[I]$ com as mesmas permutações secretas utilizadas para a matriz do sistema, e na mesma ordem. Ao final, quando a solução ótima é encontrada (assumindo-se que o problema é limitado e tal solução existe), os participantes calculam o valor de tal solução baseado nas variáveis básicas da matriz resultante.

A fim de recuperar as posições originais dos índices da matriz do sistema, os participantes executam o Algoritmo 3. Isto é feito somente uma vez, depois que a solução ótima foi encontrada. A matriz $[I]$, entrada do algoritmo, contém as cominações (de linhas e colunas) que mapeiam a matriz original na matriz permutada $[D]$, resultante da última iteração do protocolo seguro de PL.

A primeira linha do Algoritmo 3 corresponde a uma invocação do Algoritmo 1 a fim de permutar uma última vez a matriz do sistema. Visto que a solução já foi encontrada, não se faz necessário selecionar índices ou realizar-se novas operações de pivoteamento. A invocação do Algoritmo 1 é utilizada para manter secretas as combinações de todas as permutações individuais dos participantes. Nenhum participante é capaz de inferir informações sobre os índices da matriz $[I]$ e, portanto, nenhuma

Algorithm 3: Index Recovering Protocol for the Linear Round Permutation

Input: Two encrypted matrices $[D]_{pk}$ and $[I]_{pk}$ (with same size), where $[I]_{pk}$ contains the combination of all permutations applied to the initial system matrix
Output: The final matrix with indexes restored to the original position

1 $[D']_{pk}, [I']_{pk} \leftarrow$ LinearRoundPerm$([D]_{pk}, [I]_{pk})$;
2 $I' \leftarrow$ DecryptMatrix$([I']_{pk})$;
3 **extract** Π_r and Π_c from I';
4 $[D'']_{pk} \leftarrow \Pi_c(\Pi_r([D']_{pk}))$;
5 **return** $[D'']_{pk}$;

informação extra pode ser obtida.

O resultado da linha 1 inclui a matriz criptografada $[I']_{pk}$ e sendo assim, a combinação de todas as pemutações. Durante o próximo passo, os parceiros colaborativamente decifram os elementos da matriz permitindo a extração da combinação (resultante) das permutações para linhas e colunas, em claro. O processo de decifragem é conduzido de acordo com o Algoritmo 4, que por sua vez, implementa o algoritmo de decifragem com limiar descrito por [57] para o sistema criptográfico de Paillier. Mesmo que as permutações sejam reveladas depois do processo de decifragem, nenhuma informação extra pode ser obtida, o que garante a privacidade das entradas.

Algorithm 4: DecryptMatrix$([A]_{pk})$

Input: An encrypted matrix $[A]_{pk}$ where $[A]_{pk}(i,j) \in \mathbb{Z}_{pk}$
Output: The plain text corresponding to the given input

1 **forall** i, j *from* A **do**
2 | $A(i,j) \leftarrow$ Decrypt$([A]_{pk}(i,j))$; `algorithm from [57]`
3 **end**
4 **return** A;

Os participantes calculam a função inversa da matriz de permutação na linha 3 do Algoritmo 3. As posições originais dos índices são recuperadas aplicando-se tais permutações novamente sobre a matriz do sistema. O resultado do Algoritmo 3 é a matriz do sistema contendo os valores referentes à última iteração do PL seguro nas suas posições originais, o que permite a extração da solução ótima.

5.3.1 Análise

A análise de complexidade dos protocolo será dividida em três partes: complexidade de comunicação, rodada e computacção.

Para a análise da complexidade de comunicação do Algoritmo 1, assume-se que o tamanho das mensagens das linhas 6 e 8 é igual a $2(nm)\mathcal{O}(c)$ bits, onde m é o número de linhas e n o número de colunas de ambas as matrizes $[D]$ e $[I]$. Cada participante envia uma mensagem durante a fase sequencial e o último participante envia $(p-1)$ mensagens relativas à operação de *multicast* da linha 8. A complexidade de comunicação é portanto, limitada por $2(p-1)(mn)\mathcal{O}(c) = \mathcal{O}(p(mn)c)$ bits.

A complexidade de rodada é calculada como a soma das $(p-1)$ mensagens em sequência mais a última mensagem de *multicast* do participante p. No total, o algoritmo leva p rodadas para aplicar a combinação das permutações, i.e., $\mathcal{O}(p)$.

A complexidade computacional é dominada pelo protocolo MixAndPermute. A fim de embaralhar e permutar uma matriz, tal protocolo executa (mn) exponenciações modulares mais (mn) multiplicações modulares. O custo da aplicação das permutações sobre as matrizes é irrelevante. A complexidade computacional total do Algoritmo 1, assumindo-se que ambas as invocações do protocolo MixAndPermute podem ser realizadas em paralelo, é então limitada por $\mathcal{O}(mn)$ exponenciações modulares.

As complexidades referentes ao protocolo de recuperação de índices (Algoritmo 3) serão analisadas a seguir. A complexidade de comunicação é dependente dos resultados do Algoritmo 1 e do protocolo de decifragem Decrypt (proposto por [57]). Para decifrar uma matriz de tamanho mxn

é necessário executar-se (mn) decifragens individuais. Para decifrar um elemento são necessárias 2 mensagens de *multicast* por participante, com tamanho $\mathcal{O}(c)$ bits, i.e., $2(p-1)\mathcal{O}(c)$ bits. Sendo assim, a complexidade de comunicação é igual a $2p(p-1)(mn)\mathcal{O}(c)$ mais $(2p(p-1)(mn)\mathcal{O}(c))$ bits, o que pode ser re-escrito como $\mathcal{O}(p^2(mn)c)$ bits.

A complexidade de rodada é dominada pela complexidade do Algoritmo 1, visto que o protocolo de decifragem é executado em um número constante de rodadas. O Algoritmo 3 leva $(p+2)$ rodadas para ser executado, i.e., $\mathcal{O}(p)$ rodadas.

Finalmente, tanto o Algoritmo 1 quanto o protocolo DECRYPT são responsáveis pela complexidade computacional do protocolo de recuperação de índices. Para decifrar um único elemento, o protocolo executa uma exponenciação modular (para o processo de decifragem compartilhada), mais $(t+1)$ exponenciações modulares, mais $(2t^2+3t+1)$ multiplicações modulares, mais (t^2+t) inversões modulares (para o processo de combinação das partes do segredo). Assume-se que operações independentes são executadas em paralelo. O custo total para decifrar um elemento é então, igual a 2 exponenciações modulares, mais $(t+1)$ inversões modulares, mais $(2t+1)$ multiplicações modulares. De maneira similar, as (mn) decifragens (referentes à matriz $[I]$) são executadas em paralelo. Assim, a complexidade computacional do Algoritmo 3 é igual a $\mathcal{O}(mn)$ exponenciações modulares e $\mathcal{O}(mn)$ multiplicações modulares para a permutação, mais $\mathcal{O}(1)$ exponenciações modulares e $\mathcal{O}(t)$ multiplicações e inversões modulares para decifragem. Considere o modelo de segurança semi-honesto no qual o limiar para compartilhamento de segredo é definido como $t < p-1$. O protocolo de recuperação de índices executa portanto, $\mathcal{O}(mn)$ exponenciações, $\mathcal{O}(p+mn)$ multiplicações e $\mathcal{O}(p)$ inversões modulares.

5.4 Permutação Multi-Parte: Algoritmo Logarítmico de Rodada

Cenários práticos na área de cadeias logísticas incluem atraso na comunicação. Entre os efeitos negativos de tais condições pode-se citar a degradação no tempo total de execução e uma possível alternativa é a utilização de algoritmos com complexidade de rodada reduzida.

O Algoritmo 5 descreve um protocolo seguro de permutação multi-parte com complexidade logarítmica de rodada, em contraste ao protocolo proposto na Seção 5.3, com complexidade linear de rodada. Ao invés de utilizar-se permutações representadas como vetores, o Algoritmo 5 usa a notação matricial definida na Seção 5.2. Uma vantagem do uso de matrizes para representar permutações reside no fato de que a combinação de todas as permutações individuais dos participantes pode ser calculada separadamente, como um única matriz para linhas e uma única matriz para colunas. Desta forma, todas as computações seguras inclusas no protocolo LOGROUNDPERM fazem uso de matrizes criptografadas de permutação (como definido na Seção 5.2).

Todos os participantes executam o Algoritmo 5 em paralelo e retornam o mesmo resultado: a matriz $[D]$ permutada em linhas e colunas de acordo com a combinação das permutações individuais (e secretas) de cada um dos participantes. Tal processo ocorre de modo que nenhum dos participantes é capaz de obter informações a respeito das permutações dos outros, exceto o que pode ser inferido a partir dos resultados. Todas as operações independentes, como multiplicações seguras, são executadas em paralelo.

O Algoritmo 5 é pode ser dividido em quatro passos principais:

1. Troca de matrizes de permutação secretas: cada participante realiza um *mulaticast* de suas matrizes (criptografadas) e recebe as matrizes de todos os outros (linhas 2-5);

2. Multiplicação segura das matrizes de permutação em um número logarítmico de rodadas (linhas 6 e 7);

3. Atualização da combinação de permutações secretas da iteração atual (linhas 8 e 9). Este passo otimiza o protocolo de recuperação de índices;

4. Multiplicação segura final das matrizes de permutação (linha e coluna) e da matriz $[D]$, produzindo o resultado desejado (linhas 10-12);

A vantagem desta abordagem em relação ao algoritmo linear é basicamente representada pelas operações nas linhas 6 e 7. O Algoritmo 6 é invocado (LOGROUNDMULT) a fim de combinar as permutações secretas dos participantes em um número logarítmico de rodadas.

Algorithm 5: LogRoundPerm

Input: The encrypted system matrix $[D]_{pk}$

Output: The encrypted system matrix permuted according the composition of the permutations of all parties

$k \leftarrow$ current iteration number;
${}^{k}[R]^{i}_{pk} \leftarrow$ generate random permutation matrix for rows on party i for iteration k;
${}^{k}[C]^{i}_{pk} \leftarrow$ generate random permutation matrix for cols. on party i for iteration k;
multicast ${}^{k}[R]^{i}_{pk}$ and ${}^{k}[C]^{i}_{pk}$;
wait until receiving ${}^{k}[R]^{i}_{pk}$ and ${}^{k}[C]^{i}_{pk}$ for $1 \geq i \geq p$;
$[R]_{pk} \leftarrow$ LogRoundMult$({}^{k}[R]^{1}_{pk}, \cdots, {}^{k}[R]^{p}_{pk})$;
$[C]_{pk} \leftarrow$ LogRoundMult$({}^{k}[C]^{1}_{pk}, \cdots, {}^{k}[C]^{p}_{pk})$;
${}^{k}[R]_{pk} \leftarrow$ MultiplyMatrices$([R]_{pk}, {}^{k-1}[R]_{pk})$;
${}^{k}[C]_{pk} \leftarrow$ MultiplyMatrices$([C]_{pk}, {}^{k-1}[C]_{pk})$;
$[D']_{pk} \leftarrow$ MultiplyMatrices$([R]_{pk}, [D]_{pk})$;
$[D'']_{pk} \leftarrow$ MultiplyMatrices$([D']_{pk}, [C^T]_{pk})$;
return $[D'']_{pk}$;

Algorithm 6: LogRoundMult$(\{P\}_{pk})$

Input: An array of encrypted permutation matrices of size p, where p is a power of two: $\{P\}_{pk} = \{[P]^{0}_{pk}, \cdots, [P]^{p-1}_{pk}\}$

Output: An encrypted permutation matrix equal to the multiplication of all the matrices in the input array

if $p = 1$ **then**
  return $[P]^{0}_{pk}$;
else
  $\{B\}_{pk} \leftarrow \{[B]^{0}_{pk}, \cdots, [B]^{(\frac{p}{2}-1)}_{pk}\}$; `create an array of matrices with half the size of the input array`
  for $i = 0, \cdots, (\frac{p}{2} - 1)$ **do**
    $[B]^{i}_{pk} \leftarrow$ MultiplyMatrices$([P]^{2i}_{pk}, [P]^{2i+1}_{pk})$;
  end
end
return LogRoundMult$(\{B\}_{pk})$;

A entrada para o Algoritmo 6 é um vetor de matrizes criptografadas. Assume-se que o tamanho de tal vetor é potência de dois, por simplicidade. O vetor de entrada é então encarado como uma árvore binária e as multiplicações de matrizes são executadas em pares (ver Figura 5.2). O protocolo de multiplicação MultiplyMatrices é invocado de maneira recursiva, de modo que a saída é um vetor de tamanho igual à metade do tamanho do vetor de entrada, até que exista somente uma matriz no vetor de saída. Tal matriz é igual à combinação de todas as matrizes do vetor de entrada, e portanto, representa a combinação segura de todas as permutações secretas dos participantes.

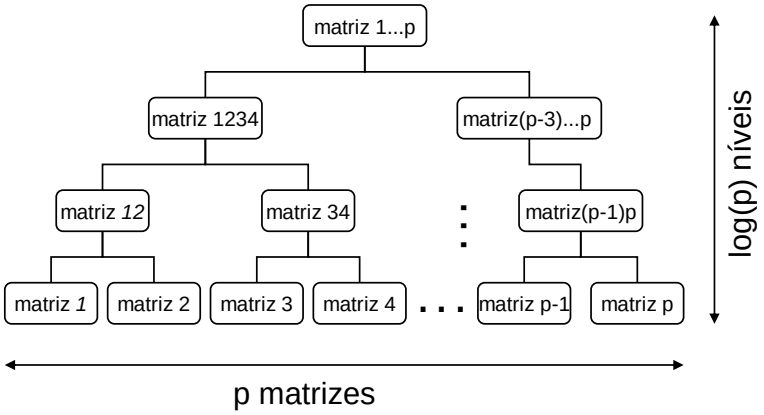


Figura 5.2: Exemplo de uma estrutura de árvore binária utilizada pela solução logarítmica

O produto de duas matrizes é realizado de acordo com a operação tradicional de multiplicação de matrizes. O Algoritmo 7 descreve como tal processo é conduzido. Nota-se que é necessário um esforço extra quando se trata de matrizes criptografadas. Cada multiplicação segura de dois elementos é realizada invocando-se o protocolo de multiplicação segura de Cramer e Damgard, introduzido em [29] (ver Seção 2.4.4 para detalhes), que executa em um número constante de rodadas.

Algorithm 7: MultiplyMatrices($[A]_{pk}, [B]_{pk}$)

Input: Two encrypted matrices $[A]_{pk}$ and $[B]_{pk}$ where $[A]_{pk}(i,j), [B]_{pk}(i,j) \in \mathbb{Z}_{pk}\ \forall i,j$
Output: An encrypted matrix that is the result of the multiplication of the input matrices

```
1  initialize [C]_pk;
2  if number of columns of A = number of rows of B then
3  |  for i = 0 to number of rows of A do
4  |  |  for j = 0 to number of columns of B do
5  |  |  |  for k = 0 to number of columns of A do
6  |  |  |  |  [C]_pk(i,j) ← [C]_pk(i,j) ⊕ SecureMult([A]_pk(i,k), [B]_pk(k,j));
7  |  |  |  end
8  |  |  end
9  |  end
10 end
11 return [C]_pk;
```

O protocolo de recuperação de índices é bastante simples (ver Algoritmo 8). A razão para tal simplicidade está nas multiplicações de matrizes das linhas 8 e 9 do Algoritmo 6. O protocolo LogRoundPerm é invocado em cada iteração. Devido às propriedade bijetivas das permutações e das propriedades de multiplicação de matrizes, a combinação de permutações secretas é atualizada a cada iteração. A saída do protocolo LogRoundMult é a combinação das permutações secretas de todos os participantes para a iteração atual i. O produto deste elemento e da combinação resultante da última iteração resulta em uma nova matriz de permutação que representa a combinação corrente e atualizada das permutações secretas da iteração 1 até a iteração i (atual). Depois da última iteração k do método simplex, todos os participantes conhecem a combinação das permutações como $[R] = {}^{k}[R] * {}^{k-1}[R] * \cdots * {}^{0}[R]$ para linhas, por exemplo. O mesmo aplica-se para colunas, i.e., matriz $[C]$. Tais matrizes, $[R]$ and $[C]$, são suficientes para o cálculo da permutação inversa, que por sua vez

recupera as posições originais dos índices da matriz do sistema.

Algorithm 8: Index Recovering Protocol for the Logarithmic Round Permutation

Input: Three encrypted matrices $[D]_{pk}$, ${}^{f}[R]_{pk}$ and ${}^{f}[C]_{pk}$ (with same size), where D is the final system matrix, ${}^{f}R$ and ${}^{f}C$ are the resulting permutation matrices for rows and columns, respectively, from the last iteration $k = f$

Output: The final matrix with indexes restored to the original position

1 $[D']_{pk} \leftarrow$ MultiplyMatrices$({}^{f}[R^T]_{pk}, [D]_{pk})$;
2 $[D'']_{pk} \leftarrow$ MultiplyMatrices$([D']_{pk}, {}^{f}[C]_{pk})$;
3 **return** $[D'']_{pk}$;

Utilizando-se álgebra matricial, a matriz de permutação para linhas é igual a $[R] * [D] = [D']$, o que implica em $[D] = [R^{-1}] * [D']$, por exemplo. O cálculo da matriz inversa de matrizes criptografadas é uma operação extremamente complexa. No entanto, matrizes de permutação são matrizes ortogonais, por definição. Isso significa que a seguinte afirmação é verdadeira: $R * R^T = I$ (a matriz identidade). Sendo assim, a matriz inversa existe e pode ser escrita/calculada como $R^{-1} = R^T$. Tal propriedade possibilita a utilização de uma solução simplificada para multiplicação de matrizes como apresentada no Algoritmo 8.

5.4.1 Análise

De maneira similar à Seção 5.3.1, a análise de complexidade será dividida em rodada, comunicação e computação, inicialmente para o protocolo de permutação e em seguida para o protocolo de recuperação de índices.

A complexidade de rodada do Algoritmo 5 é dividida em três partes distintas:

1. Troca das matrizes de permutação (linhas 1-5);
2. Multiplicação em número logarítmico de rodadas (linhas 6 e 7);
3. Multiplicação final de matrizes (linhas 8-11);

A parte (i) não é constante no número de rodadas, mas depende da complexidade do protocolo LogRoundMult. O Algoritmo 6 claramente precisa de $\mathcal{O}(\log(p))$ rodadas para ser executado. O protocolo de multiplicação segura de [29] executa em exatas 5 rodadas. Assume-se que todas as operações independentes são executadas em paralelo, i.e., geração de matrizes de permutação e todas as multiplicações seguras referentes às invocações dos protocolos LogRoundMult e MultiplyMatrices. Assim, o número total de rodadas necessárias para a execução do Algoritmo 5 é igual a $(1+5\log(p)+5)$, o que implica em $\mathcal{O}(\log(p))$ rodadas.

Complexidade de comunicação considera todas as mensagens de *multicast* nais quais uma mensagem de *multicast* corresponde a $(p-1)$ mensagens. Considere que mensagens possuem tamanhos diferentes dependendo do seu tipo e que ambas as matrizes de permutação são de dimensões mxn, com $m = n = a$.

Na linha 4, cada participante envia mensagens de *multicast* contendo ambas as matrizes de permutação, o que implica em uma complexidade de comunicação igual a $(p-1)2a^2\mathcal{O}(c)$ bits, por participante. As linhas 6 e 7 correspondem exatamente à mesma operação: para um array de tamanho p, o protocolo LogRoundMult executa $(p-1)$ multiplicações de matrizes. O produto de duas matrizes requer a invocação do protocolo MultiplyMatrices e, para tal, a^3 multiplicações seguras são realizadas. Para cada multiplicação segura quatro mensagens de *multicast* de tamanho igual a $\mathcal{O}(c)$ bits, mais uma mensagem de *multicast* contendo um inteiro em claro (32 bits, i.e., constante) são enviadas. A complexidade de comunicação das operações das linhas 6 e 7 é limitada por $2(p-1)\{a^3[(p-1)(4\mathcal{O}(c)+\mathcal{O}(1))]\}$ bits, por participante, cada. As linhas 8-11 correspondem a quatro invocações independentes do protocolo MultiplyMatrices que resulta na troca de $a^3[(p-1)(4\mathcal{O}(c)+\mathcal{O}(1))]$ bits, por participante. O número total de unidades de informação trocadas durante a execução do Algoritmo 5 é igual a $\{(p-1)2a^2\mathcal{O}(c)\} + 2(p-1)\{a^3[(p-1)(4\mathcal{O}(c)+\mathcal{O}(1))]\} + \{4a^3[(p-1)(4\mathcal{O}(c)+\mathcal{O}(1))]\}$, por participante, o que pode ser representado por $\mathcal{O}(p^3a^3c)$ bits.

Para a complexidade computacional considera-se três operações principais: (i) geração das matrizes, (ii) multiplicação segura de elementos (em pares), e (iii) decifragem de um elemento.

(i) requer $2a^2$ exponenciações modulares;

(ii) requer três exponenciações modulares (uma cifragem homomórfica e duas multiplicações por constantes), mais $(2p + 3)$ multiplicações modulares (referentes às adições homomórficas), mais 2 decifragens;

(iii) requer $(t + 2)$ exponenciações modulares, mais $(t^2 + t)$ inversões modulares, mais $(2t^2 + 3t + 1)$ multiplicações modulares (como descrito anteriormente);

Sendo assim, o protocolo SecureMult domina a complexidade computacional. Assume-se que todas as operações independentes são executadas em paralelo. Ambas as invocações do protocolo LogRoundMult (linhas 6 e 7) podem ser paralelizadas, e cada uma delas executa $(p - 1)$ multiplicações de matrizes. As 4 multiplicações de matrizes que se seguem são executadas em duas rodadas. Assume-se matrizes quadradas de dimensões axa. O cálculo do produto de duas matrizes resulta em a^3 multiplicações seguras de pares de elementos (invocações do protocolo SecureMult), que são executadas em paralelo. O custo de uma única multiplicação segura é limitado por 7 exponenciações, $2(t+1)$ inversões e $4(t + 1)$ multiplicações modulares. O valor do limiar é definido pelo modelo de segurança utilizado. Neste caso, para o modelo semi-honesto, assume-se $t < p - 1$. A complexidade computacional total é, então, limitada por $\mathcal{O}(pa^3)$ exponenciações, mais $\mathcal{O}(p^2a^3)$ multiplicações e inversões modulares.

O protocolo de recuperação de índices é constante no número de rodadas. Ele depende de duas invocações do protocolo MultiplyMatrices tendo vetores de tamanho igual a 2 como entrada. Cada invocação é equivalente a a^3 multiplicações seguras, executadas em um número constante de rodadas, que podem ser paralelizadas.

A complexidade de comunicação do protocolo de recuperação de índices depende da complexidade de comunicação do protocolo SecureMult. Para cada uma das $2a^3$ multiplicações seguras são enviadas 5 mensagens de *multicast*, por participante. Portanto, a complexidade de comunicação é limitada por $\mathcal{O}(p^2a^3)$ bits.

A complexidade computacional é dominada pelo protocolo SecureMult, que por sua vez depende do protocolo Decrypt. Assumindo-se operações realizadas em paralelo e o modelo de segurança semi-honesto, os custos computacionais do protocolo de recuperação de índices são limitados por $\mathcal{O}(a^3)$ exponenciações modulares.

5.5 Comparação Teórica

As Tabelas 5.1 e 5.2 trazem uma comparação teórica dos custos de ambos os protocolos seguros de permutação multi-parte e protocolos de recuperação de índices, respectivamente. A solução logarítmica possui complexidade de rodada inferior para os protocolos de permutação e recuperação de índices com um custo extra de comunicação e computação.

Solução	Comunicação	Rodada	Computação
Linear	$\mathcal{O}(pa^2c)$	$\mathcal{O}(p)$	$\mathcal{O}(a^2)$
Logarítmica	$\mathcal{O}(p^3a^3c)$	$\mathcal{O}(\log(p))$	$\mathcal{O}(pa^3)$

Tabela 5.1: Comparação de complexidades dos protocolos seguros de permutação multi-parte

O custo extra é relativo ao processo de multiplicação de matrizes no qual a^3 multiplicações são requeridas para matrizes de dimensões $a \times a$, em ambos os protocolos baseados na abordagem logarítmica. Tal solução também requer $(p - 1)$ multiplicações de matrizes *por participante*, para o protocolo de permutação, o que implica em um fator extra igual a p^2 nos custos de comunicação.

Análises baseadas na notação *Big-*$\mathcal{O}$ frequentemente escondem constantes que podem influenciar as soluções na prática. Sendo assim, uma avaliação prática das soluções propostas é de extremo interesse. A próxima seção fornece uma estimativa de como a combinação de complexidades e condições da rede impactam o desempenho dos protocolos na prática.

Solução	Comunicação	Rodada	Computação
Linear	$\mathcal{O}(p^2a^2c)$	$\mathcal{O}(p)$	$\mathcal{O}(a^2)$
Logarítmica	$\mathcal{O}(p^2a^3c)$	$\mathcal{O}(1)$	$\mathcal{O}(a^3)$

Tabela 5.2: Comparação de complexidades dos protocolos de recuperação de índices

5.6 Implementação e Resultados

A linguagem de programação Java (*Sun*, versão 6[1]) foi utilizada na implementação de ambos os protocolos, linear e logarítmico, e RMI[2] foi utilizado como *middleware* para comunicação, provendo abstração de alto nível para chamadas de procedimentos remotos. Foram utilizadas um total de 16 máquinas equipadas com processadores *dual core* de 2.4GHz e 3GB de memória RAM, com o sistema operacional GNU Linux, conectadas por uma rede *gigabit*. Uma implementação da versão com limiar do sistema criptográfico de Paillier foi responsável pela geração das chaves, i.e., chave pública e partes secretas da chave privada [57, 77]. O tamanho das chaves foi definido como 1024 bits e $s = 2$, o que significa que o tamanho do texto em claro é $\mathbb{Z}_{mod^2}$ enquanto textos criptografados possuem tamanho de $\mathbb{Z}_{mod^3}$, onde mod denota o módulo do sistema criptográfico. Assume-se o modelo de segurança semi-honesto que requer $t < p - 1$ (requisito de entrega de mensagens). Note que apesar de os participantes eventualmente receberem todas as partes (mensagens), por questões de desempenho utiliza-se $t < p/2$ para as computação (visto que o valor de t implica diretamente no número de exponenciações durante o processo de combinação das partes do segredo).

O custo de um algoritmo é medido como a combinação de exponenciações (ME), inversões (MI), multiplicações (MM) – em matemática modular – além dos custos de communicação (com). A primeira bateria de testes mediu os custos (em milissegundos, ms) de cada uma das operações listadas acima como a média aritmética referente à 100 execuções. A Tabela 5.3 contém os resultados.

ME	MI	MM	com
280	3	$<< 1$	≈ 1

Tabela 5.3: Custo de operações básicas (em ms)

Cenários *práticos* de cadeias logísticas incluem **atraso** na comunicação e programação **paralela**, i.e., operações independentes podem ser executadas em paralelo. Assume-se que cada participante possui um total de ρ processadores disponíveis. O tempo de execução absoluto de cada protocolo é então definido como:

$$
\begin{aligned}
RT_{lin} &= \frac{2a^2p}{\rho}ME + (p)com \\
RT_{log} &= \frac{(5\log(p)+5)a^3 + 2a^2}{\rho}ME + \frac{(\log(p)+1)a^3}{\rho}MI + \\
&\quad \left(\frac{(3\log(p)+3)a^3}{\rho} + 1\right)com \\
RT_{ir_{lin}} &= \left(\frac{2a^2p}{\rho} + 2\right)ME + 2MI + (p)com \\
RT_{ir_{log}} &= \frac{10a^3}{\rho}ME + \frac{2a^3}{\rho}MI + \frac{6a^3}{\rho}com
\end{aligned}
$$

onde RT denota o tempo de execução absoluto para os protocolos de permutação e RT_{ir} denota o tempo de execução absoluto para os protocolos de recuperação de índices. Note a dependência do tamanho do problema a e do número de participantes p.

Quanto maior o valor do atraso da rede, maior é a influência da parcela referente à comunicação no tempo de execução absoluto. Considere, por exemplo, o impacto em se ter os custos de comunicação

[1] http://java.sun.com/javase/

[2] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

da mesma ordem, ou de ordem maior do que os custos de uma exponenciação. No que diz respeito aos protocolos de permutação, os custos de computação e comunicação para a solução logarítmica dependem do termo $\log(p)$, e podem ser executados em paralelo. Por outro lado, os custos de comunicação da solução linear não podem ser executados em paralelo e, além disto, possuem taxa de crescimento linear. O mesmo se aplica para os protocolos de recuperação de índices nos quais os custos de comunicação da solução linear não podem ser paralelizados.

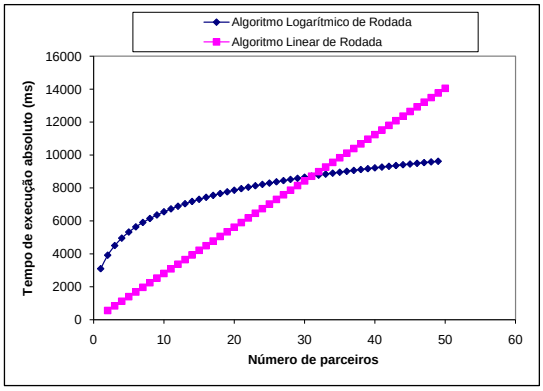
Quanto maior o nível de paralelismo, melhor os resultados. As $2a^2$ exponenciações referentes aos processos de embaralhamento e decifragem da solução linear são independentes e portanto, podem ser executados simultaneamente. Assim, os custos podem ser re-escritos como segue.

$$RT_{lin} = (p)ME + (p)com$$
$$RT_{ir_{lin}} = (p+1)ME + (2)MI + (p)com.$$

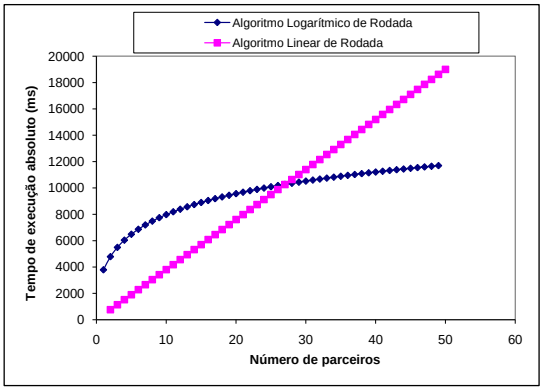
De maneira análoga, o fator a^3 presente na solução logarítmica é reduzido a uma constante devido ao paralelismo das operações e os custos atualizados podem ser escritos como

$$RT_{log} = (5\log(p) + 6)ME + (\log(p) + 1)MI + (3\log(p) + 4)com$$
$$RT_{ir_{log}} = 10ME + 2MI + 6com.$$

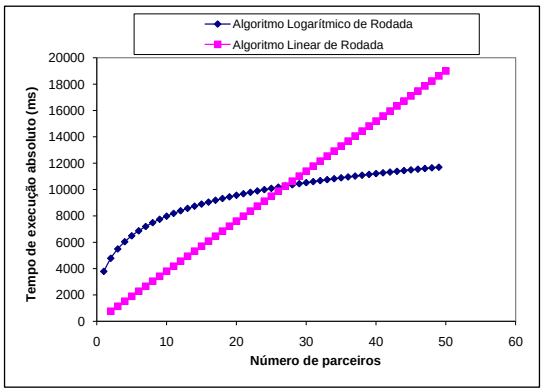
Os próximos experimentos tem o intuito de comparar o desempenho dos protocolos de permutação e recuperação de índices para ambas as soluções, linear e logarítmica, utilizando-se os custos reduzidos mostrados acima. Para tal, foram considerados quatro cenários com atraso na rede diferentes: 0, 100, 200, e 300 milissegundos de atraso. Uma variação no número de participantes também foi considerada. O objetivo inicial é descobrir o ponto de intersecção entre as curvas de ambas as soluções, a partir do qual a solução logarítmica trará melhores resultados.



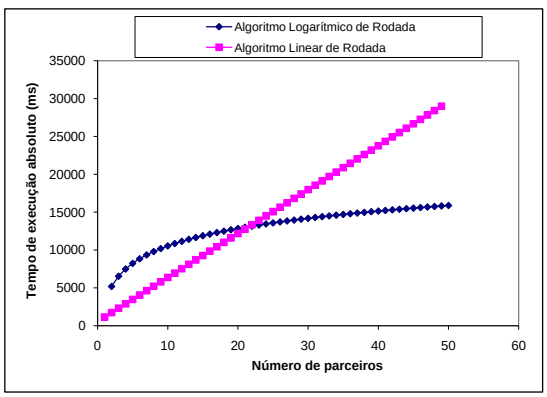
(a) Sem atraso



(b) 100ms de atraso



(c) 200ms de atraso



(d) 300ms de atraso

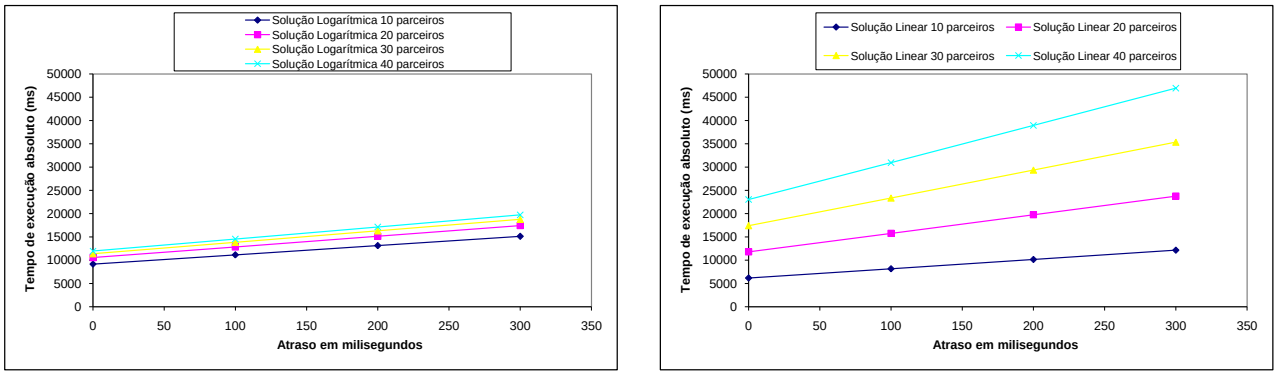
Figura 5.3: Comparação entre protocolos de permutação

Primeiramente analisaremos os protocolos de permutação. Com um atraso de comunicação

pequeno (aproximadamente igual a zero), uma cadeia logística com 31 participantes é suficiente para mostrar melhorias no desempenho quando se utiliza a solução logarítmica (ver Figure 5.3). Cenários realísticos incluem atrasos maiores. Os casos (c) e (d) da Figura 5.3 mostram os resultados para exemplos nos quais os atrasos assumem tais valores. Para estes cenários, o uso da solução logarítmica para CL com mais de 23 participantes já seria suficiente para melhores resultados. Com o mercado globalizado as CL estão se tornando cada vez maiores envolvendo parceiros de todos os lugares do mundo. As melhorias no desempenho observadas pelo uso da solução logarítmica se tornam mais expressivas quando os problemas envolvem grandes CLs, e um grande número de participantes, devido à diferença na taxa de crescimento das funções, como fica evidente na Figura 5.3.

A escolha de uma solução para permutação segura multi-parte determina o tipo de solução para o protocolo de recuperação de índices. Assume-se o mesmo cenário prático, i.e., atraso na comunicação e custos referentes a um paralelismo ilimitado. O protocolo de recuperação de índices da solução linear depende de uma invocação do protocolo de permutação, que depende linearmente do número de participantes e não oferece a possibilidade de paralelismo (pelo menos no que diz respeito à parcela de comunicação). Por outro lado, a solução logarítmica se baseia unicamente em duas multiplicações de matrizes, que podem ser executadas em um número constante de rodadas e podem ser paralelizadas. Na prática, cenários com mais de 8 participantes pode ser eficientemente resolvidos utilizando-se a solução logarítmica.

A solução como um todo é resultado da combinação dos protocolos de permutação e recuperação de índices. Os experimentos a seguir analisaram como o atraso na comunicação afeta tais soluções. O procedimento é bastante similar, com variações no atraso e no número de participantes. A Figura 5.4 apresenta os resultados.



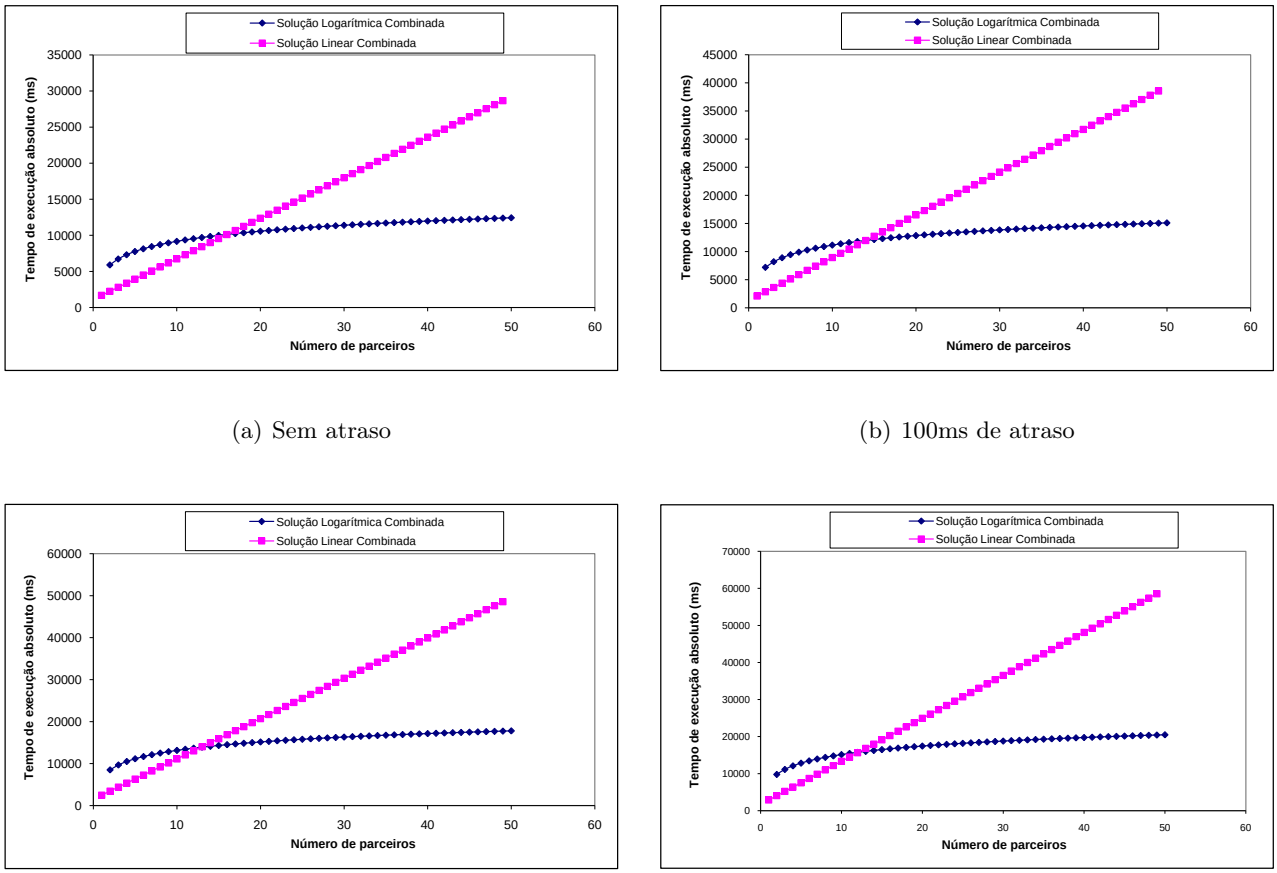
(a) Log Alg (b) Linear Alg

Figura 5.4: Efeito do atraso no tempo de execução absoluto

Considere o caso (a). As curvas permanecem aproximadamente paralelas enquanto o atraso de comunicação aumenta, i.e., note que a diferença entre as curvas aumenta de maneira lenta com a introdução de atraso. Agora considere o caso (b). A influência do atraso é visivelmente mais expressiva, i.e., com pequenos aumentos no atraso as curvas tendem a se distanciar rapidamente, devido à influência da parcela de comunicação, que cresce a uma taxa linear no número de participantes.

O tempo de execução para as soluções combinadas é apresentado na Figura 5.5. Por um lado, o protocolo de permutação da solução logarítmica cresce a uma taxa logarítmica enquanto o protocolo de recuperação de índices é constante. Por outro lado, tanto o protocolo de permutação quanto o protocolo de recuperação de índices da solução linear crescem a uma taxa linear no número de participantes. Como consequência, observa-se um ponto de menor valor para a intersecção das curvas (ver Figura 5.5), quando se compara aos resultados da Figura 5.3. Por exemplo, utilizando-se a solução logarítmica para CLs com apenas 14 participantes (Figure 5.5), contra os anteriores 23 (Figura 5.3), já é suficiente para obter-se melhores resultados, em relação ao tempo de execução. Observa-se que os resultados da solução logarítmica são consideravelmente melhores devido, especialmente, ao tempo constante de execução do protocolo de recuperação de índices.

Ressalta-se que os resultados experimentais apresentados aqui dependem da implementação dos algoritmos. Visto que não foram apresentados métodos formais para validação dessa implementação,



(a) Sem atraso

(b) 100ms de atraso

(c) 200ms de atraso

(d) 300ms de atraso

Figura 5.5: Comparação entre soluções combinadas em respeito ao número de participantes

não se pode garantir, pelo menos por meio de prova formal, a veracidade dos resultados experimentais apresentados. Naturalmente isso deve ser levado em conta na análise do que foi apresentado, mas de modo algum invalida os resultados nem mesmo diminui a importância dos trabalhos realizados.

5.7 Considerações

Diferentes abordagens para protocolos de permutação segura em problemas de otimização de cadeias logísticas ainda não haviam sido formalmente considerados. Esta sub-operação (do método simplex) é comumente considerada na sua forma mais simples, com um número linear de rodadas, sendo que os participantes permutam um dado conjunto de informações baseados em suas permutações secretas individuais, de maneira sequencial. A fim de se preservar a privacidade de tais informações, mecanismos adicionais são introduzidos, como o processo de embaralhamento. Além da solução de [68], que considera somente cenários com dois participantes, a literatura contém outros exemplos de protocolos multi-parte de embaralhamento. Os trabalhos de [8], em 1988, já introduziam idéias de como se realizar o embaralhamento de vetores de maneira privada, o que pode ser utilizado para protocolos de permutação. Mais recentemente, Silaghi descreveu como *mix-nets* podem ser utilizadas para embaralhar um vetor de elementos criptografados e posteriormente recuperar as posições originais, utilizando-se permutações inversas [91]. Silaghi forneceu adicionalmente, provas de conhecimento nulo (*zero-knowledge proofs*) para corretude dos protocolos de embaralhamento (e de recuperação de posições). Mesmo que novos conceitos tenham sido introduzidos, a mesma idéia de permutação linear continua a ser utilizada. O foco tem sido em soluções com segurança provável para os diferentes modelos de segurança, e não em aspectos práticos de aplicações como otimização de CL.

Os trabalhos descritos neste capítulo apontam na direção de fornecer melhorias na complexidade de rodada de protocolos de permutação, como uma tentativa de reduzir os efeitos advindos da existência

de atrasos de comunicação, em cenários práticos. A literatura é bastante escassa em implementações de tais protocolos e mais ainda, em avaliações práticas. Os resultados apresentados em [61] são o primeiro passo nesta direção, o que reflete os esforços do projeto *SecureSCM*[3], financiado pela União Européia, ambos baseados nos trabalhos apresentados no presente capítulo.

Comparações baseadas em análise de complexidade utilizando-se a notação *big-*$\mathcal{O}$ eventualmente escondem constantes que torna difícil a avaliação do comportamento dos protocolos em aplicações práticas. Na prática, por uma lado observa-se que é possível otimizar a complexidade de rodada de um dado protocolo utilizando-se paralelismo, i.e., operações independentes são executas em paralelo, assumindo-se a existência de mais de um processador e, por outro lado, observa-se aspectos negativos da existência de atraso na comunicação. A vantagem da utilização do protocolo logarítmico está na sua estrutura de operações independente. Não somente as multiplicações seguras de matrizes podem ser paralelizadas, mas também a comunicação. O protocolo de permutação linear não oferece esta possibilidade. Os resultados mostram que, por exemplo, não é difícil encontrar custos de comunicação de magnitude similar aos custos de exponenciações (operação mais cara). Tais atrasos são comumente observados em cenários práticos e portanto, pode-se afirmar que a parcela de comunicação possui um importante papel no desempenho do protocolo como um todo.

Em última análise, a fonte das melhorias observadas no protocolo de permutação da solução logarítmica é o poder de paralelismo computacional. O poder de computação atual (nas empresas em que tais protocolos seriam aplicados) pode não ser capaz de efetuar o tipo de otimização considerada nas análises, e.g., execução de a^3 operações em paralelo. Em uma implementação sequencial ou com um baixo poder de paralelismo, a solução linear apresentaria melhores resultados.

Identifica-se, portanto, uma relação de compromisso entre complexidade de rodada, número de participantes, condições da rede de comunicação e poder computacional. Para a maioria dos casos práticos, esta relação coloca os resultados em algum ponto intermediário entre os extremos: implementação sequencial e paralelismo ilimitado.

Todavia, prediz-se cenários futuros em que soluções com complexidade reduzida de rodada farão grande diferença (melhorias expressivas). Existem várias evidências que apontam na direção de um mercado cada vez mais globalizado dentro de poucos anos. Não existem mais barreiras físicas nos negócios e empresas de todas as partes do mundo interagem entre si a fim de fornecer melhores serviços e produtos. O plano diretor da cadeia logística é um exemplo concreto, no qual empresas geograficamente dispersas planejam, de maneira colaborativa, produção, armazenamento e transporte. O tamanho de tais cadeias tende a crescer, eventualmente incluindo centenas de parceiros. Aliado a tudo isso, durante os últimos anos, tem se observado um tremendo aumento no poder computacional, acompanhado pela queda dos preços. Isto possibilita inclusive, às empresas de pequeno e médio porte a compra de melhores equipamentos e o ganho em competitividade no mercado. O crescimento massivo do poder de computação torna os requisitos de paralelismo possíveis. Sendo assim, a tendência é favorável à soluções com complexidade reduzida de rodada, como o protocolo logarítmico apresentado neste capítulo.

5.8 Conclusão

Problemas de otimização de cadeias logísticas frequentemente incluem vários participantes e atraso na comunicação entre eles. Este capítulo considerou o problema de protocolos multi-parte seguros de permutação em condições reais de rede de comunicação, mais especificamente, na presença de atraso. Duas abordagens foram introduzidas. Primeiro, um extensão direta do protocolo proposto por Li e Atallah. Segundo, um protocolo com complexidade logarítmica de rodada como uma tentativa de reduzir os efeitos negativos do atraso na comunicação. Ambas as soluções foram comparadas por intermédio de uma análise teórica detalhada incluindo complexidade de rodada, computação e comunicação. Posteriormente, os protocolos foram analisados experimentalmente, dada uma implementação e avaliação de desempenho. Em particular, foram estimados os efeitos do número de participantes e diferentes condições de comunicação no desempenho das soluções.

Dado o poder de computação atual, a solução linear possivelmente apresenta melhores resultados, visto que a solução logarítmica depende de um alto nível de paralelismo. Na prática, observa-se a

[3] www.securescm.org/

existência de uma relação de compromisso entre o número de participantes e o nível de paralelismo, para os protocolos de permutação, o que põe os resultados em algum ponto intermediário entre uma implementação sequencial e paralelismo ilimitado. Entretanto, as estimativas para cenários futuros são promissoras. As cadeias logísticas ficarão cada vez maiores, consequencia do mercado globalizado, contendo centenas de particantes dispersos pelo mundo todo. Adicionalmente, o poder computacional tende a crescer massivamente, o que fornece os meios para níveis elevados de paralelismo. A tendência é portanto, a favor da solução logarítmica, i.e., com complexidade reduzida de rodada, de modo que em um futuro próximo tal solução será ainda mais rápida quando comparada a soluções lineares.

Capítulo 6

Otimizando o Método Simplex Paralelo com Preservação da Privacidade

Este capítulo marca o início da segunda parte desta dissertação. Serão consideradas aqui, soluções multi-parte com preservação da privacidade para programação linear nas quais a matriz do sistema é representada por variáveis secretas, i.e., criptografadas. O *overhead* associado às operações de indexação com variáveis secretas, é reduzido por meio do uso de paralelismo.

6.1 Introdução

A literatura em programação linear com preservação da privacidade inclui soluções nas quais os índices do elemento pivô são selecionados em claro, como em [68]. Os Capítulos 4 e 5 discutiram algumas desvantagens de tal abordagem e propuseram esquemas e protocolos para melhorar o desempenho em aplicações práticas. No entanto, também é possível resolver problemas de otimização de cadeias logísticas nos quais as variáveis do sistema são representadas utilizando-se elementos criptografados. O trabalho de Toft [93] introduz primitivas para computações seguras multi-parte além de um protocolo de programação linear com preservação da privacidade baseado em operações com varáveis secretas. Este protocolo é provado ser seguro contra adversários passivos.

Soluções baseadas em varáveis secretas não requerem protocolos de embaralhamento ou permutação. Privacidade é assegurada pela utilização de valores criptografados, i.e., as propriedades do sistema criptográfico garantem que nenhum participante obterá informações extra sobre os elementos da matriz. Protocolos seguros e com custos reduzidos (número constante de rodadas) para multiplicação e comparação utilizando variáveis criptografadas já foram propostos pela literatura (ver Seção 2.4.4). Entre os exemplos encontram-se o protocolo para multiplicação de [29] e o protocolo de comparação de inteiros de [46].

É extremamente difícil realizar operações de indexação com variáveis secretas. Cada operação precisa ser do tamanho do vetor ou matriz e, por isso, envolve comunicação intensa. Em [93], uma notação especial é utilizada. Considere um vetor de variáveis secretas $[A]$. Um índice secreto i é armazenado essencialmente como um contador unário um vetor consistindo somente de 0's com exceção da posição i, que contém um elemento de valor igual a 1. Para executar uma operação de indexação sobre variáveis secretas realiza-se uma soma de produtos como segue

$$[A]([i]) = \sum_{j=1}^{[A].len} [A](j) \cdot [i](j)$$

Este procedimento é correto, preserva a privacidade das entradas e requer $[A].len$ multiplicações seguras. Atribuição de valores utilizando-se variáveis secretas também é possível. A afirmação $[A]([i]) \leftarrow [x]$ é equivalente à atualização de *todas* as entradas de $[A]$ como

$$[A](j) \leftarrow [i](j) \ ? \ [x] : [A](j)$$

Como a operação de indexação, esta operação requer exatamente $[A].len$ multiplicações seguras.

A notação para vetores é também utilizada para para representar matrizes, como apresentado na Seção 5.2.

Toft otimiza complexidade de rodada por meio da utilização de operações *em pacotes*, i.e., todas as operações independentes são executadas em paralelo. Assume-se que todas as $[A].len$ multiplicações são então, executadas em uma única rodada, em paralelo, o que requer o uso de recursos ilimitados, ou paralelização ilimitada. Em cenários reais, paralelização ilimitada não é possível simplesmente devido à limitações de recursos, o que implica que somente um número limitado de processadores está disponível para computação. Visto que o interesse desta dissertação é fornecer soluções práticas para programação linear segura utilizando-se computação paralela (máquinas *multi-core/multi-processor*), se faz necessária uma análise mais precisa do protocolo proposto por [93].

Se por um lado Toft propõe a execução de tantas operações em paralelo quanto possível, por outro lado, a literatura contém diversas tentativas de métodos de paralelização de programas lineares, i.e., algoritmo paralelo por projeto. Gondzio e Sarkissian [53] propõem uma implementação em paralelo do método do Ponto-Interior de [58, 96]. Os trabalhos de Lee e Wiswall [62] introduzem uma generalização do método simplex de *Nelder-Mead* [74] para processadores em paralelo. No que diz respeito ao método simplex tradicional proposto por Dantzig [36], cita-se as abordagens de Shu e Wu [90], Natraj e Thompson [73] além de Dong et al. [41]. Esta última, em particular, possui diversos recursos que podem facilitar uma implementação com preservação da privacidade. Entre eles relaciona-se a independência da estrutura ou tipo de constantes em relação ao método de paralellização e extensão do modelo BSP para computação segura multi-parte.

Uma vantagem de algoritmos paralelos em relação à algoritmos sequenciais é o ganho em desempenho, conhecido como *speed-up*. No entanto, todas as abordagens citadas acima compartilham uma característica em comum: projeto orientado ao desempenho sem nenhuma preocupação com segurança de modo que nenhuma versão segura de tais algoritmo foi proposta até então. Sendo assim, introduz-se neste Capítulo, uma versão segura de algoritmos paralelos para PL que se adequa às necessidades de privacidade na gestão da cadeia logística.

Este Capitulo está dividido em três partes. Primeiramente, a Seção 6.2 realiza uma comparação teórica entre duas abordagens: paralelização de um algoritmo seguro e versão segura de um algoritmo paralelo. Primeiro, o protocolo de [93] é revisado, evidenciando os custos exatos por processador. Depois, um algoritmo paralelo e seguro, baseado no modelo BSP, é proposto e analizado. Na segunda parte, a Seção 6.3 descreve um implementação da escolha mais simples: paralelização de um algoritmo seguro. Métodos distintos de sincronização são comparados sob diferentes condições de rede. Dada a melhor implementação, o desempenho é praticamente independente de tais condições, mas o paralelismo, i.e., número de *threads* por processador ou core, precisa ser adaptado. Na terceira parte, a Seção 6.4 introduz e avalia um algoritmo de agendamento adaptativo para seleção dinâmica do número de *threads*, de modo que não é necessário determinar-se estaticamente e *a priori* tal número para *speed-up* ótimo. O algoritmo pode também lidar com variações nas condições da rede de comunicação. Finalmente, a Seção 6.5 discute algumas considerações finais e a Seção 6.6 concluí o capítulo.

6.2 Comparação de Algoritmos Seguros Paralelos

O Capítulo 3 descreveu o protocolo seguro para programação linear de [93] e o método simplex paralelo baseado no modelo BSP de [41]. O objetivo desta seção é de comparar teoricamente duas abordagens: paralelização de um algoritmo seguro e versão segura de um algoritmo paralelo, evidenciando complexidades de comunicação, computação e rodada, assim como custos exatos por processador/core. Para que seja possível comparar tais abordagens, faz-se necessário, por um lado, paralelizar o protocolo seguro de [93] e, por outro lado, propor uma versão segura do método simplex paralelo de [41]. As próximas seções descrevem como tal procedimento foi realizado, bem como custos detalhados, com análises de complexidade.

Serão considerados dois tipos de complexidade:

- Complexidade de Comunicação: medida em termos de multiplicações seguras (`SM`) e comparações seguras (`SC`), i.e., as quais se referem ao número total de *bits* trocados entre os participantes;
- Complexidade Computacional (`CC`): cada multiplicação segura é considerada uma unidade e uma

comparação segura leva c_r unidades;

Para o restante deste trabalho assume-se, sem perda de generalidade, que as entradas são números inteiros, pela mesma razão descrita na Seção 3.5.1. Assume-se também que o tableau inicial foi criptografado utilizando-se a chave pública *pk* e que as partes da chave secreta (privada) foram distribuídas entre os participantes. A este procedimento chama-se *passo 0*: uma fase de configuração para a distribuição das chaves e geração do tableau inicial. As variáveis criptografadas, e.g., vetores, matrizes, índices, são descritas de acordo com a notação introduzida na Seção 5.2, sendo representadas com $[\cdot]$. Por simplicidade, durante o restante do capítulo, se referirá às matrizes e vetores criptografados somente como matrizes e vetores.

6.2.1 Programação Linear Segura em Paralelo

Toft já havia considerado uma noção de paralelismo em suas análises. De acordo com [93], um número arbitrário de primitivas lineares pode ser executado de maneira paralela em uma única rodada desde que qualquer resultado de comandos anteriores, exigido pelo comando atual, esteja disponível.

Em se tratando da funcionalidade genérica de CSM, assume-se que entrada, multiplicação de duas varáveis secretas e saída requerem exatamente uma rodada cada. O número de operações realizadas em paralelo é portanto, ilimitado, mas existe uma clara distinção entre complexidade de rodada e complexidade de comunicação. As rodadas, na funcionalidade genérica de CSM, não são diretamente equivalentes às rodadas de comunicação. Uma rodada na funcionalidade descreve o que pode ser executado em paralelo, mas não infere, de modo algum, nas mensagens sendo trocadas durante a execução do protocolo em questão [93].

Para que um protocolo seja considerado de rodada constante, requere-se, naturalmente, primitivas de rodada constante, o que é o caso (em notação *big-*$\mathcal{O}$) dos trabalhos de Toft.

A complexidade de rodada é posteriormente dividida em *pre-processamento* e *computação online*. Operações que são independentes das entradas podem ser executadas antecipadamente, como consequência da maneira iterativa em que o protocolo é construído. O ganho em complexidade é imediato, i.e., um número ilimitado de operações executadas em paralelo em uma única rodada mais pre-processamento contra uma execução sequencial. Entretanto, visto que a análise de Toft assume paralelismo arbitrário, uma investigação mais detalhada é requerida para aplicações práticas.

Os protocolos de Toft são então, re-escritos com uma construção mais precisa, a fim de claramente mostrar as operações que podem ser realizadas em paralelo e os custos exatos da implementação de cada uma destras operações em máquinas *multi-core/multi-processor*.

O paralelismo é realizado em nível de participante, sendo que cada um destes possui um conjunto de processadores, denotado pela variável p. Os algoritmos cobrem exatamente os mesmo passos descritos na Seção 3.5.2 e, para tal, utilizam a mesma notação de [93]. A matriz do sistema é representada na forma de tableau, o que permite que os elementos sejam referenciados de maneira simples. Tal matriz possui dimensões $m+1$ linhas e $m+n+1$ colunas. A função objetivo é representada por um vetor de tamanho $(m+n)$, e o conjunto de inequações, por um vetor de tamanho igual a $m+1$, ambos inclusos no tableau. Assume-se ainda, um vetor de tamanho igual a m, para os índices das variáveis básicas do sistema.

O controle de fluxo de variáveis criptografadas é necessário e a seguinte idéia (algoritmo) é utilizada:

Algorithm 9: Control-Flow based on Secret Variables

Input: Three secret variables $[a]$, $[b]$ and $[b']$ to enforce
`if ([`a`] = 1) then [`b`] else [`b'`]`
Output: The boolean result selecting either $[b]$ or $[b']$

1 **return** $[a] \otimes ([b] - [b']) + [b']$;

Cada um dos passos e correspondentes análises de complexidades são agora considerados separadamente. Note que a complexidade de comunicação é a mesma do protocolo original de Toft [93].

6.2.1.1 Seleção da Coluna Pivô

O primeiro passo consiste na seleção da variável que entrará na base, i.e., a coluna pivô. A saída deste passo é o índice da variável de entrada (o primeiro elemento negativo contido no vetor que representa a função objetivo) e uma cópia da coluna pivô (ver Algoritmo 10 para detalhes).

Algorithm 10: Select the pivot column

Input: The tableau $[T]$
Output: The secret variable $[c]$ such that the value indexed by $[T]([c])$ is the first negative value, and $[C]$ representing the pivot column

1 **forall** *processor* p **do in parallel**
2 **for** $i = 1, \cdots, (m+n)$ **do**
3 $[D](i) \leftarrow [F](i) \overset{?}{<} 0$; | CC: $\frac{(m+n)}{p} c_r$
4 **end**
5 **end**
6 $[D'] \leftarrow prev_{\vee}([D])$; | CC:12 from [93]
7 $[c](1) \leftarrow [D'](1)$;
8 **for** $i = 2, \cdots, (m+n)$ **do**
9 $[c](i) \leftarrow [D'](i) - [D](i-1)$;
10 **end**
11 **forall** *processor* p **do in parallel**
12 **for** $i = 1, \cdots, (m+1)$ **do**
13 **for** $j = 1, \cdots, (m+n)$ **do**
14 $[C](i) \leftarrow [C](i) + \left([T](i,j) \otimes [c](j) \right)$; | CC: $\frac{(m+1)(m+n)}{p}$
15 **end**
16 **end**
17 **end**
18 **return** $[c], [C]$;

Indexação baseada em varáveis secretas é requerida, e.g., $[C]([c])$, e para tal utiliza-se o esquema de indexação secreta apresentado na Seção 6.1.

A comparação de todos os elementos (valores) da função objetivo com 0 tem um custo computacional de $(m+n)c_r/p$ unidades. A invocação da primitiva $prev_{\vee}(\cdot)$ custa exatamente 12 unidades de computação (ver [93] para maiores detalhes). A última operação de indexação, que tem o objetivo de obter uma cópia da coluna pivô, é realizada a um custo de $(m+1)(m+n)/p$ unidades de computação. No total, o custo computacional da seleção da coluna pivô é igual a $(m+n)c_r/p + (m+1)(m+n)/p + 12$ unidades e a complexidade de computação é igual a $(m+14)(m+n)SM + (m+n)SC$.

6.2.1.2 Seleção da Linha Pivô

O objetivo é selecionar uma variável para entrar na base, i.e., a linha pivô. Em linhas gerais, deseja-se encontrar o índice da razão mínima (como definido em [76]), utilizando-se a regra de *Bland* em caso de empate. A saída é o índice da variável que deixará a base e uma cópia da linha que contém o elemento pivô. O Algoritmo 11 descreve esta operação.

Comparação entre os valores contidos no vetor de inequações requer uma transformação inicial na qual todas as restrições que não são aplicáveis, como as que contém valores negativos ou iguais a zero, se tornem aplicáveis. Decisão baseada em variáveis secretas é utilizada ao custo de $\mathcal{O}(m)$ comparações e multiplicações e $m(c_r+1)/p$ unidades de computação.

Visto que todas as restrições se tornam aplicáveis (depois de aplicada a transformação), a saída desejada pode ser calculada utilizando-se comparações de inteiros. O protocolo com complexidade log log de rodada de [93] é aplicado a um custo computacional de $\log\log(m) \cdot 2(3mc_r/p+2) + mc_r/p + 4$ unidades e o índice da razão mínima (o índice da linha pivô) é encontrado. Feito isto, uma outra

Algorithm 11: Select the pivot row

Input: The tableau $[T]$, $[c], [C], [B]$
Output: The secret variables $[r], [R]$

forall *processor* p **do in parallel**
  for $i = 1, \cdots, m$ **do**
    $[C'](i) \leftarrow [C](i) \overset{?}{>} 0 \ ? \ [C](i) \ : \ 1;$
    $[B'](i) \leftarrow [C](i) \overset{?}{>} 0 \ ? \ [B](i) \ : \ \infty \ ;$ $\quad \left| \texttt{CC}: \frac{m(c_r+1)}{p} \right.$
  end
end
$[X] \leftarrow \Big(([C'](1), [B'](1), [S](1)), \cdots, ([C'](m), [B'](m), [S](m)) \Big);$
forall *processor* p **do in parallel**
  $[r] \leftarrow \min_{\mathcal{O}(\log\log(\cdot))}([X]) \ ;$ $\quad \left| \texttt{CC}: \log\log(m) \left(2(\frac{3mc_r}{p} + 2) + \frac{mc_r}{p} + 4\right) \right.$
  for $i = 1, \cdots, (m + n + 3)$ **do**
    for $j = 1, \cdots, m$ **do**
      $[R](i) \leftarrow [R](i) + \Big([T](i, j) \otimes [r](j) \Big) \ ;$ $\quad \left| \texttt{CC}: \frac{m(m+n+3)}{p} \right.$
    end
  end
end
return $[r], [R]$;

operação de indexação é necessária para copiar os elementos da linha pivô e, para tal, são gastas $m(m + n + 3)/p$ unidades de computação.

Ao todo, a seleção da linha pivô executa $m(c_r + 1)/p + \log\log(m) \cdot 2(3mc_r/p + 2) + mc_r/p + 4 + m(m + n + 3)/p$ unidades computacionais e possui complexidade de comunicação igual a $(13m + m^2 + mn)SM + (7m)SC$.

6.2.1.3 Multiplicação de Todas as Linhas Não-Pivô pelo Elemento Pivô

A posição do próximo elemento pivô é conhecida por todos os participantes, i.e., índice da linha e da coluna pivô. O próximo passo consiste em multiplicar todos os elementos da matriz, exceto os que pertencem à linha do elemento pivô, pelo próprio elemento pivô. O resultado deste passo é o tableau atualizado sendo que todos os elementos, exceto os pertencentes a linha pivô, estão multiplicados pelo elemento pivô. O Algoritmo 12 descreve o procedimento em detalhes.

Uma matriz auxiliar é utilizada para manter os elementos da linha pivô intocados, a um custo computacional de m/p unidades. Posteriormente, todos os elementos das matrizes auxiliar e do sistema são multiplicados, preservando-se corretude e privacidade. A complexidade computacional do procedimento é igual a $(m + 1)(m + n + 1)/p$ unidades.

Ao todo, este passo possui complexidade computacional igual a $m/p + (m + 1)(m + n + 1)/p$ unidades e complexidade de comunicação igual a $[m + (m + 1)(m + n + 1)]SM$.

6.2.1.4 Subtração de um Múltiplo da Linha Pivô de Todas as Linas Nâo-Pivô

Os custos computacionais para a subtração de um múltiplo da linha pivô de todas as linhas não-pivô é análogo ao descrito na Seção 6.2.1.3. A entrada deste processo é o tableau resultante do passo anterior e a saída é um tableau atualizado de modo que todos os elementos da coluna pivô, com exeção do elemento na linha pivô, são iguais a zero. O Algoritmo 13 contém o pseudo-código utilizado neste procedimento.

A complexidade computacional é igual a $m/p + (m + 1)(m + n + 1)/p$ unidades e a complexidade de comunicação é igual a $[m + (m + 1)(m + n + 1)]SM$, de modo similar ao passo anterior.

Algorithm 12: Multiplication of all non-pivot rows by the pivot element

Input: The tableau $[T]$, the index of the pivot row $[r]$, and the pivot element $[p]$
Output: The updated tableau $[T']$

```
1  for i = 1, ··· , (m + 1) do
2  |  [M](i) ← [p];
3  end
4  forall processor p do in parallel
5  |  for i = 1, ··· , m do
6  |  |  [M](i) ← [r](i) ? 1 : [M](i) ;                          | CC: m/p
7  |  end
8  |  for i = 1, ··· , (m + 1) do
9  |  |  for j = 1, ··· , (m + n + 1) do
10 |  |  |  [T'](i, j) ← [M](i) ⊗ [T](i, j) ;                    | CC: (m+1)(m+n+1)/p
11 |  |  end
12 |  end
13 end
14 return [T'];
```

Algorithm 13: Subtraction of multiple of the pivot row from all non-pivot rows

Input: The tableau $[T]$, the index of the pivot row $[r]$, the pivot row $[R]$, and the original pivot column $[C]$
Output: The updated tableau $[T']$

```
1  forall processor p do in parallel
2  |  for i = 1, ··· , m do
3  |  |  [C](i) ← [r](i) ? 0 : [C](i) ;                          | CC: m/p
4  |  end
5  |  for i = 1, ··· , (m + 1) do
6  |  |  for j = 1, ··· , (m + n + 1) do
7  |  |  |  [T'](i, j) ← [T](i, j) − ( [C](i) ⊗ [R](j) ) ;      | CC: (m+1)(m+n+1)/p
8  |  |  end
9  |  end
10 end
11 return [T'];
```

6.2.1.5 Divisão das Linhas Não-Pivô pelo Elemento Pivô da Iteração Anterior

Este é o último passo de uma iteração do método simplex seguro em paralelo: todas as linhas não-pivô são divididas pelo elemento pivô da iteração anterior. Naturalmente este processo consiste em uma divisão segura de inteiros, que é conhecido por ser um procedimento extremamente caro: não existe protocolo realmente eficiente para divisão genérica de inteiros. No entanto, este não é um caso genérico e simplificações podem ser feitas. É garantido que o elemento pivô da iteração anterior divide (de maneira exata) todos os elementos do tableau, exceto os que estão na linha pivô (para uma prova detalhada desta propriedade, favor referir-se a [86]).

Esta simplificação possibilita a utilização de um protocolo para o cálculo da inversão de elementos ($1/x$, para um x qualquer da matriz), como o proposto em [5]. Assim, multiplica-se todos os elementos das linhas não-pivô pela inversa do elemento pivô. Esta operação é equivalente à operação descrita na Seção 6.2.1.3 e, portanto, o mesmo algoritmo pode ser utilizado. Visto que a atualização do elemento pivô anterior pelo elemento atual possui custo desprezível e a inversão é eficiente, a complexidade do protocolo de divisão é equivalente à descrita na Seção 6.2.1.3.

6.2.1.6 Terminação

O método simplex seguro em paralelo termina se não houverem mais variáveis para entrar na base ou, em outras palavras, se não houverem mais elementos negativos no vetor correspondente à função objetivo. Isto indica que uma solução ótima para o problema de programação linear foi encontrada.

A entrada para o processo de terminação é um tableau no qual o vetor equivalente à função objetivo contém apenas valores positivos e a saída é calculada em função dos elementos que fazem parte da base atual. Sendo assim, é possível que a solução final do PL assuma valores racionais (divisões podem ocorrer). Se for desejado ou necessária uma solução exata, será preciso executar uma divisão segura de inteiros, ao fim do protocolo, que apesar de ser computacionalmente cara, não pode ser evitada.

6.2.2 Simplex Paralelo e Seguro Baseado no Modelo BSP

Diversos exemplos de como paralelizar métodos simplex entre um certo número de processadores foram apresentado na Seção 6.1. Uma estratégia óbvia seria simplesmente considerar cada participante do problema de PDCL como um *processador* do algoritmo paralelo. No entanto, tal estratégia não cumpre todos os requisitos de segurança impostos pela teoria de computação segura. É necessário não somente proteger as informações trocadas entre os processadores, mas também as informação reveladas a um determinado processador, de modo que as informações de todos os participantes permaneçam *oblivious*.

A escolha do método simplex paralelo baseado no modelo BSP se deve principalmente à técnica de paralelismo utilizada e aos custos de implementação. O primeiro é independente da estrutura do problema ou tipo de restrições, o que possibilita a resolução de qualquer problema de PDCL, desde que na forma canônica (mesma hipótese de Toft e Li e Atallah). Os custos extra devido a uma implementação segura, i.e., custos associados a computações com valores criptografados, são os mesmos que os da solução proposta por Toft.

6.2.2.1 Modelo de CSM Baseado no Modelo BSP

Propõe-se uma nova estratégia de paralelismo com preservação de privacidade para algoritmos paralelos. Ela é baseada no modelo BSP com as seguintes hipóteses:

- o problema de PL é colaborativamente resolvido entre ρ participantes;
- cada participante possui um conjunto de p processadores reais η disponíveis para computação (por efeito de simplicidade, assume-se que todos os participantes possuem pelo menos p), onde $1 \geq \eta \geq p$;
- cada participante conhece a chave pública pk e possui uma parte da chave secreta sk_L, de um esquema criptográfico homomórfico baseado em limiar, onde $1 \geq L \geq \rho$, ambos distribuídos *a priori* (considerada fase de inicialiação, ou *passo 0*);

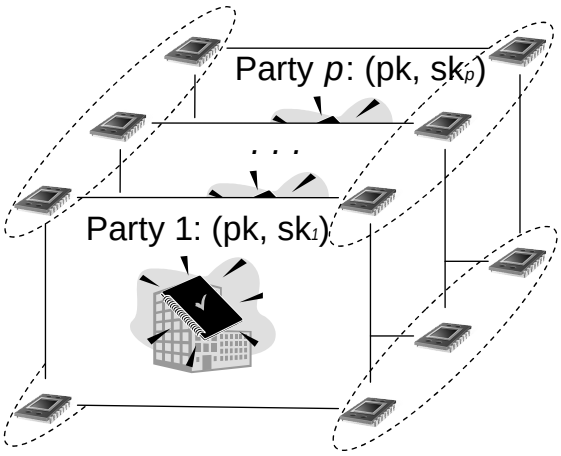


Figura 6.1: Modelo abstrato de processamento paralelo

- as interações entre processadores são conduzidas de acordo com o modelo BSP proposto por [94] descrito na Seção 3.6.1;

O modelo BSP fornece propriedades como escalabilidade, portabilidade e previsibilidade ao passo que considera máquinas paralelas como a união de três elementos básicos: (i) um conjunto de processadores (capazes de executar operações em paralelo), (ii) uma rede de comunicação entre eles, que entrega mensagens de maneira ponto-a-ponto, e (iii) um mecanismo de sincronização [94].

A Figura 6.1 representa a proposta de estratégia de paralelismo. Considere o conjunto de participantes $A = \{L : 1 \geq L \geq \rho\}$ e o conjunto de processadores reais $B = \{\eta : 1 \geq \eta \geq p\}$ (para cada participante). Seja $C = \{L_\eta\ , \forall L \in A, \eta \in B\}$ o conjunto de todos os processadores reais, tal que $|C| = |A| \times |B| = \rho \times p$. Finalmente considere um processador abstrato D_η como um conjunto de processadores reais, um de cada participante, tal que $D_\eta = \{L_i\ , \forall L \in A, i = \eta, \eta \in B\}$, com $|D_\eta| = \rho$. As linhas pontilhadas na Figura 6.1 representam um processador abstrato. O método simplex paralelo e seguro é executado pelos p processadores abstratos.

Existe uma relação entre o modelo BSP original (descrito na Seção 3.6.1) e o modelo seguro introduzido pela na Figura 6.1. Os processadores são representados pelas linhas pontilhadas (processadores abstratos) e a rede de comunicação global pode ser implementada utilizando-se o conceito de partes secretas de CSM.

"Programas" de CSM são *oblivious* (pelo menos em teoria), i.e., seu tempo total de execução não depende das entradas. A possível diferença no tempo de execução dos algoritmos não oferece, portanto, perigo de vazamento de informação e, em virtude disto, pode-se implementar os mecanismos de sincronização sem a necessidade de protocolos espeíficos de segurança.

As operações seguras requerem que cada processador real conheça a chave pública pk e a parte compartilhada da chave secreta sk_L do seu respectivo participante. A construção apresentada para os chamados processadores abstratos possibilita que as computações sejam executadas de maneira paralela pelos processadores reais distribuídos entre os participantes, de modo que a privacidade da computação é garantida.

Considera-se dois tipos de interações: (i) interações que ocorrem dentro de um determinado processador abstrato, e (ii) interações que acontecem entre diferentes processadores abstratos. No primeiro caso as interações involvem todos os participantes e seguem o paradigma de CSM. Isto significa, neste caso e na maioria dos casos, em um padrão malha de comunicação entre todos os participantes. No segundo caso, as interações entre processadores abstratos seguem o algoritmo paralelo. Elas envolvem a comunicação segura de segredos de um processador (remetente) para um outro processador (destinatário) em um mesmo processador abstrato. Visto que ambos os processadores, remetente e destinatário, encontram-se no mesmo participante, esta comunicação é local, i.e., sem a necessidade de uma rede que conecta os participantes.

A versão segura do método simplex paralelo baseado no modelo BSP possui uma estrutura vertical de paralelismo, baseada em linhas, o que implica dizer que o problema é subdividido entre os processadores abstratos de modo que cada um destes recebe um número de linhas da matriz do

problema. Devido à técnica de paralelismo utilizada, dois passos extra são necessários: (i) compartilhamento do tableau, executado somente uma vez, antes do início da execução dos passos do método simplex, e (ii) a atualização do tableau, executada ao fim do protocolo, depois que a solução ótima foi encontrada.

As próximas seções descrevem em detalhes, com análise de complexidades, os algoritmos utilizados para cada um dos passos do simplex. Apesar de o problema ser dividido entre os p processadores abstratos, o que aparentemente resultaria em uma complexidade de comunicação dividida por p, a análise é feita com base na quantidade de informação trocada *por participante*, de maneira similar à notação utilizada na Seção 6.2.1. Para efeito de simplicidade, refere-se aos processadores abstratos como processadores, de agora em diante.

6.2.2.2 Compartilhamento do Tableau

O protocolo simplex seguro e paralelo inicia com o compartilhamento do tableau entre os participantes. O tableau é inicialmente criptografado e representado pela matriz $[T]$, de dimensões $m+1$ linhas e $m+n+1$ colunas. O vetor $[S]$, de tamanho m, contém os índices das variáveis da base.

Esta operação é equivalente à descrita em [41]: o processador 0 inicialmente copia a linha correspondente à função objetivo, da matriz $[T]$ para um vetor privado $[Q]$ e depois, todos os processadores, em paralelo, compartilham as linhas da matriz $[T]$, de acordo com seus identificadores. Ao fim, cada processador possui uma matriz $[C]$ e um vetor $[B]$ que correspondem à suas partes da matriz $[T]$. A partir deste ponto, a matriz completa do sistema não é mais considerada.

Este passo seguro possui custo computacional constante e igual a 2 unidades devido à execução paralela, e visto que operações de indexação tem custo desprezível, o custo total é limitado por $\mathcal{O}(1)$. Todos os participantes conhecem a matriz inicial $[T]$, de modo que não ocorre troca de mensagens durante este passo.

6.2.2.3 Seleção da Coluna Pivô

De maneira similar à Seção 6.2.1.1, a coluna pivô é escolhida como o índice do primeiro elemento negativo do vetor $[Q]$ (função objetivo).

O Algoritmo 14 mostra como este índice é selecionado. Uma idéia semelhante à idéia utilizada por Toft [93] é aplicada aqui, com apenas uma diferença: somente o processador 0 está ativo durante este passo visto ser o único processador que conhece a função objetivo.

Algorithm 14: Select the pivot column

Input: The matrix $[Q]$
Output: The secret variable $[col]$ such that the value indexed by $[Q]([col])$ is the first negative value

do *on abstract processor ZERO*
  for $i = 1, \cdots, (m+n)$ **do**
    $[D](i) \leftarrow [Q](i) \overset{?}{<} 0$; | CC: $(m+n)c_r$
  end
end
$[D'] \leftarrow prev_{\vee}([D])$; | CC: 12
$[col](1) \leftarrow [D'](1)$;
for $i = 2, \cdots, (m+n)$ **do**
  $[col](i) \leftarrow [D'](i) - [D](i-1)$;
end
return $[col]$;

A saída do Algoritmo 14 é o índice da coluna pivô a um custo computacional de $(m+n)c_r + 12$ unidades e complexidade de computação igual a $12(m+n)SM + (m+n)SC$.

6.2.2.4 Seleção da Linha Pivô

O processo de seleção da variável que sai da base pode ser divido em três partes. Primeiramente, após terem recebido informação a respeito do índice da coluna pivô $[col]$, os processadores independentemente procuram pela linha pivô nas suas matrizes individuais (linhas 1-18 do Algoritmo 15). Esta operação é realizada em paralelo de modo que cada processador procura pelo candidato a linha pivô no seu conjunto de linhas. Ao fim da primeira parte, cada processador informa ao procesador 0 sobre a linha pré-selecionada para deixar a base. A segunda parte consiste no processador 0 procurando, entre as candidatas recebidas dos outros participantes, pela variável que sairá da base (linhas 19-21 do Algoritmo 15). E finalmente, na última parte, o processador que possui a linha pivô (identificada pelo procesador 0) é informado da decisão e realiza um *multicast* da cópia desta linha para todo os participantes (linhas 22-28 do Algoritmo 15).

Algorithm 15: Select the pivot row

Input: The secret index $[col]$, the matrix $[C]$ and the vector $[B]$
Output: The secret variables $[R_{ow}]$

1 **foreach** *abstract processor* **do in parallel**
2 **for** $i = 1, \cdots, (m/p)$ **do**
3 **for** $j = 1, \cdots, (m+n)$ **do**
4 $[C'](i) \leftarrow [C'](i) + \left([C](i,j) \otimes [col](j) \right)$; ; | CC: $\frac{m(m+n)}{p}$
5 **end**
6 **end**
7 **for** $i = 1, \cdots, m/p$ **do in parallel**
8 $[C''](i) \leftarrow [C'](i) \overset{?}{>} 0 \ ? \ [C'](i) \ : \ 1$; | CC: $\frac{m(c_r+1)}{p}$
9 $[B''](i) \leftarrow [C'](i) \overset{?}{>} 0 \ ? \ [B](i) \ : \ \infty$;
10 **end**
11 $[X] \leftarrow \left(([C''](1), [B''](1), [S](1)), \cdots, \left([C''](\frac{m}{p}), [B''](\frac{m}{p}), [S](\frac{m}{p}) \right) \right)$;
12 $[row] \leftarrow \min_{\mathcal{O}(\log\log(\cdot))}([X])$; | CC: $\log\log(\frac{m}{p}) \left(2(\frac{3mc_r}{p} + 2) + \frac{mc_r}{p} + 4 \right)$ from [93]
13 **for** $i = 1, \cdots, (m/p)$ **do in parallel**
14 $[C^*](i) \leftarrow [row](i) \overset{?}{=} 1 \ ? \ 1 \ : \ [C'](i)$; | CC: $\frac{m}{p}$
15 $[B^*](i) \leftarrow [row](i) \overset{?}{=} 1 \ ? \ 1 \ : \ [B](i)$;
16 **end**
17 $[ratio] \leftarrow ([B^*], [C^*])$
18 **end**
19 **do** *on abstract processor ZERO*
20 $min_L \leftarrow \min_{\mathcal{O}(\log\log(\cdot))}([ratio]_1, \cdots, [ratio]_L)$; | CC: $\log\log(p)(3c_r + 4)$ from [93]
21 **end**
22 **do** *on abstract processor* MIN_L
23 **for** $i = 1, \cdots, (m/p)$ **do**
24 **for** $j = 1, \cdots, (m+n)$ **do**
25 $[R_{ow}](i) \leftarrow [R_{ow}](i) + \left([C](i,j) \otimes [row](j) \right)$; | CC: $\frac{m(m+n)}{p}$
26 **end**
27 **end**
28 **end**
29 **return** $[R_{ow}]$;

As linhas 1-6 correspondem a uma operação de indexação, i.e., $[C]([col])$, e as linhas 7-10 aplicam

a mesma transformação de restrições utilizada na Seção 6.2.1.2 e definida em [93]. Para a comparação das triplas, o protocolo log log de rodada introduzido em [93] é utilizado.

Nas linhas 13-17 do Algoritmo 15, a razão mínima é extraída das matrizes como sendo $[C]([col]) = [C'] \Rightarrow [C']([row]) = [C]([row], [col]) = [C^*]$ and $[B]([row]) = [B^*]$.

O processador 0 então executa o protocolo com complexidade logarítmica de rodada (como introduzido em [93]) a fim de encontrar a razão mínima entre os valores pré-selecionados (pares de valores), recebidos dos processadores.

Por fim, nas linhas 22-29 do Algoritmo 15, o processador que possui a linha pivô executa uma operação de indexação e obtém uma cópia da mesma. Como resultado, todos os processadores conhecem a coluna e a linha pivô e são capazes de identificar $[p]$, o elemento pivô.

O custo computacional para a seleção da linha pivô é igual a $m/p[2(m+n)+(c_r+1)+1] + \log\log(m/p)[7mc_r/p+8] + \log\log(p)(3c_r+4)$ unidades, enquanto que o custo de comunicação é igual a $[2m^2+2mn+11m+9p]SM + [8m+6p]SC$.

6.2.2.5 Multiplicação de Todas as Linhas Não-Pivô pelo Elemento Pivô

Para a multiplicação das linhas não-pivô pelo elemento pivô precisa-se fazer uma distinção entre os processadores. A linha pivô é encontrada somente nas matrizes $[C]$ e $[B]$ do processador MIN_L, identificado durante o passo anterior.

Como implicação direta, o processador MIN_L utiliza uma matriz auxiliar (linhas 13-15 do Algoritmo 16) para manter os elementos da linha pivô intactos (linhas 16-18 do Algoritmo 16) e então, executa a multiplicação com todos os elementos restantes (linhas 19-24 do Algoritmo 16).

Por outro lado, os outros processadores prosseguem com uma multiplicação direta de todas as entradas de suas respectivas matrizes pelo elemento pivô, como mostrado na linhas 1-11 do Algoritmo 16).

A complexidade computacional do Algoritmo 16 é igual a $m/p[1+(m+n+1)]+(m+n+1)$ e a complexidade de comunicação associada é igual a $[m(m+n+3)+(m+n+1)]SM + [m]SC$.

6.2.2.6 Subtração de um Múltiplo da Linha Pivô de todas as Linhas Não-Privô

O Algoritmo 17 descreve o processo pelo qual todas as linhas não-pivô são subtraídas de um múltiplo da linha pivô de modo que a coluna pivô resultará, ao fim do processo, inteiramente de zeros, exceto pelo elemento na linha pivô.

A computação é equivalente à do passo anterior. Primeiramente define-se o múltiplo a ser subtraído da linha pivô de modo a resultar em zero e depois subtrai-se os respectivos múltiplos de cada uma das outras linhas das matrizes.

De maneira similar, faz-se uma distinção entre processadores, i.e., processador MIN_L e o resto. A razão para tal distinção é que o processador MIN_L é o único processador que conhece as matrizes $[C]$ e $[B]$ que contém a linha pivô.

Da mesma forma, a complexidade deste passo é similar à do passo anterior: $m/p[1+(m+n+1)]+(m+n+1)$ unidades de computação e $[m(m+n+2)+(m+n+1)]SM$ para complexidade de comunicação.

6.2.2.7 Divisão das Linhas Não-Pivô pelo Elemento Pivô da Iteração Anterior

Utiliza-se a mesma idéia descrita na Seção 6.2.1.5. Divisão segura de inteiros é evitada multiplicando-se as linhas não pivô pela inversa do elemento pivô da iteração anterior.

Depois da inversão, a computação é equivalente à descrita na Seção 6.2.2.5 e como tal, o Algoritmo 16 é utilizado. Em relação à complexidade, considera-se a mema descrita para o Algoritmo 16.

6.2.2.8 Atualização do Tableau

Finalmente, a última operação consiste na atualização do tableau com a solução ótima. Neste ponto, cada um dos processadores possui uma matriz $[C]$ e um vetor $[B]$ e o processador 0 possui adicionalmente o vetor $[Q]$, que representa a função objetivo.

Algorithm 16: Multiplication of all non-pivot rows by the pivot element

Input: The matrix $[C]$, the vectors $[B]$ and $[Q]$ (if processor 0), and the pivot element $[p]$
Output: The updated matrix $[C']$, vectors $[B']$ and $[Q']$ (if processor 0) with all non-pivot rows multiplied by $[p]$

1 **foreach** *abstract processor* $\neq MIN_L$ **do in parallel**
2 **for** $i = 1, \cdots, (m/p)$ **do**
3 $[B'](i) \leftarrow [p] \otimes [B](i)$; $\left|\texttt{CC}: \frac{m}{p}\right.$
4 **for** $j = 1, \cdots, (m+n+1)$ **do**
5 $[C'](i,j) \leftarrow [p] \otimes [C](i,j)$; $\left|\texttt{CC}: \frac{(m)(m+n+1)}{p}\right.$
6 **if** *abstract processor 0 and* $i = 0$ **then**
7 $[Q'](j) \leftarrow [p] \otimes [Q](j)$; $\left|\texttt{CC}: (m+n+1)\right.$
8 **end**
9 **end**
10 **end**
11 **end**
12 **do** *on abstract processor* MIN_L
13 **for** $i = 1, \cdots, (m/p)$ **do**
14 $[M](i) \leftarrow [p]$;
15 **end**
16 **for** $i = 1, \cdots, (m/p)$ **do**
17 $[M](i) \leftarrow [row](i) \stackrel{?}{=} 1 \; ? \; 1 \; : \; [M](i)$; $\left|\texttt{CC}: \frac{m}{p}\right.$
18 **end**
19 **for** $i = 1, \cdots, (m/p)$ **do**
20 $[B'](i) \leftarrow [M](i) \otimes [B](i)$; $\left|\texttt{CC}: \frac{m}{p}\right.$
21 **for** $j = 1, \cdots, (m+n+1)$ **do**
22 $[C'](i,j) \leftarrow [M](i) \otimes [C](i,j)$; $\left|\texttt{CC}: \frac{(m)(m+n+1)}{p}\right.$
23 **if** *abstract processor 0 and* $i = 0$ **then**
24 $[Q'](j) \leftarrow [p] \otimes [Q](j)$; $\left|\texttt{CC}: (m+n+1)\right.$
25 **end**
26 **end**
27 **end**
28 **end**
29 **return** $[C']$, $[B']$ and $[Q']$;

Algorithm 17: Subtraction of multiple of the pivot row from all non-pivot rows

Input: The matrix $[C]$, the index of the pivot row $[row]$, the pivot row $[R_{ow}]$, and the original pivot column $[C_{ol}]$
Output: The updated tableau $[T']$

1 **foreach** *abstract processor* $\neq MIN_L$ **do in parallel**
2 **for** $i = 1, \cdots, (m/p)$ **do**
3 $[B'](i) \leftarrow [B](i) - \left([C_{ol}](i) \otimes [B]([row])\right)$; | CC: $\frac{m}{p}$
4 **for** $j = 1, \cdots, (m+n+1)$ **do**
5 $[C'](i,j) \leftarrow [C](i,j) - \left([C_{ol}](i) \otimes [R_{ow}](j)\right)$; | CC: $\frac{m(m+n+1)}{p}$
6 **if** *abstract processor 0 and* $i = 0$ **then**
7 $[Q'](j) \leftarrow [Q](j) - \left([C_{ol}](i) \otimes [R_{ow}](j)\right)$; | CC: $(m+n+1)$
8 **end**
9 **end**
10 **end**
11 **end**
12 **do** *on abstract processor* MIN_L
13 **for** $i = 1, \cdots, (m+1)$ **do**
14 $[C_{ol}]([row]) \leftarrow 0$;
15 **end**
16 **for** $i = 1, \cdots, (m/p)$ **do**
17 $[B'](i) \leftarrow [B](i) - \left([C_{ol}](i) \otimes [B]([row])\right)$; | CC: $\frac{m}{p}$
18 **for** $j = 1, \cdots, (m+n+1)$ **do**
19 $[C'](i,j) \leftarrow [C](i,j) - \left([C_{ol}](i) \otimes [R_{ow}](j)\right)$; | CC: $\frac{m(m+n+1)}{p}$
20 **if** *abstract processor 0 and* $i = 0$ **then**
21 $[Q'](j) \leftarrow [Q](j) - \left([C_{ol}](i) \otimes [R_{ow}](j)\right)$; | CC: $(m+n+1)$
22 **end**
23 **end**
24 **end**
25 **end**
26 **return** $[C']$, $[B']$ and $[Q']$;

O procedimento compreende inicialmente a atualização do tableau com os valores da função objetivo pelo processador 0 e, somente depois, todos os outros processadores, em paralelo, atualizam o tableau de acordo com seus identificadores.

O resultado do processo é um tableau atualizado no qual a n-ésima linha contém somente valores positivos. Isto permite que os processadores deêm inicio ao procedimento de extração da solução ótima. Assim como para o protocolo de Programação Linear Seguro e Paralelo, tal solução pode assumir valores racionais, i.e., como resultado de uma divisão dos elementos do vetor de inequações pelos elementos correspondendo às variáveis básicas. Tal processo requer divisão segura de inteiros e que não pode ser evitado.

6.2.3 Comparação dos Resultados

O primeiro aspecto da comparação será a faixa de computação ou como o problema de PL é dividido entre os participantes/processadores.

A abordagem de Toft assume que cada participante conhece o problema todo (matriz) e as operações são executadas nesta faixa, i.e., $(m+1)$ linhas e $(m+n)$ colunas. Cada participante possui múltiplos processadores e é capaz de dividir a lista de operações da rodada atual entre todos os processadores. No algoritmo baseado no modelo BSP, o problema em si é dividido entre os processadores abstratos, consequência da técnica de paralelismo. Todas as operações são executadas em uma faixa reduzida e tal redução é proporcional ao número de processadores abstratos, que é igual ao número de processadores reais *em cada um dos participantes*.

Ao todo, o protocolo seguro de PL em paralelo possui complexidade computacional igual a

$$\frac{1}{p}\left[5m^2+5mn+(6c_r\log\log(m)+3c_r+4)m+(c_r+4)n+3\right]+ \\ +4\log\log(m)+16$$

contra

$$\frac{1}{p}\left[5m^2+5mn+(7c_r\log\log(\tfrac{m}{p})+c_r+8)m\right]+(c_r+3)(m+n)+ \\ +8\log\log(\tfrac{m}{p})+\log\log(p)(3c_r+4)+17$$

do protocolo simplex baseado no modelo BSP em paralelo e seguro. Ambas as fórmulas de complexidade possuem o termo dominante de ordem semelhante, i.e., dividido pelo número de processadores p, o que revela uma complexidade computacional, em notação big-$\mathcal{O}$, limitada por $\mathcal{O}(m(m+n)/p)$.

A análise realizada por Toft resulta em uma complexidade computacional melhorada, i.e., limitada por $\mathcal{O}(\log\log(m))$, devido à suposição de que $p \approx m^2 \approx mn$. Considerando-se tal hipótese, a análise computacional atualizada do protocolo seguro de PL paralelo é limitada por $\mathcal{O}(\log\log(m))$, como esperado, visto que os termos de maior ordem são cancelados (devido às operações em paralelo) e consequentemente, o termo sub-logaritmico domina. Sob a mesma hipótese, a complexidade computacional do protocolo paralelo e seguro baseado no modelo BSP é também reduzida e limitada por $\mathcal{O}(\log\log(p))$, onde $p \approx m^2$. Nota-se que este resultado depende estritamente da complexidade das primitivas utilizadas para seleção do elemento mínimo no processador 0 (linha 20 do Algoritmo 15). Alternativamente, um algoritmo diferente de busca poderia ser empregado, com custo computacional constante, a um custo extra de comunicação, i.e., $\mathcal{O}(m^2+n)$ (para detalhes sobre o algoritmo favor referir-se a [93]).

Os processadores abstratos são constituídos de vários processadores reais (igual ao número de participantes). Operações seguras requerem a participação de um certo número de participantes ou processadores reais (definido pelo valor do limiar do sistema criptográfico), visto que cada participante possui um parte da chave secreta e, portanto, um processador abstrato pode executar somente uma operação segura por unidade computacional (passo). Ainda que cada processador abstrato seja responsável pela resolução de apenas uma parte do problema, tal divisão não traz melhorias reais em relação à complexidade computacional.

A complexidade de comunicação do protocolo seguro de PL em paralelo é igual a

$$\left[5m^2 + 5mn + 36m + 17n + 3\right] SM + \left[8m + n\right] SC$$

enquanto que a complexidade de comunicação do protocolo simplex paralelo e seguro baseado no modelo BSP é igual a

$$\left[4m^2 + 4mn + 30m + 14n + 9p + 2\right] SM + \left[10m + n + 6p\right] SC.$$

A análise da complexidade de comunicação revela que não existe diferença relevante entre as abordagens. Em notação big-$\mathcal{O}$, ambas as soluções são limitadas por $\mathcal{O}(m(m+n))$ multiplicações e $\mathcal{O}(m+n)$ comparações seguras. Novamente, nenhum ganho significante é observado ao se adicionar segurança a um algoritmo simplex paralelo.

Existem diferentes abordagens para algoritmos de simplex paralelos como o método revisado de [90], o método de duas fases em paralelo de [73] e ainda, o método de ponto interior em paralelo de [53]. No entanto, todos estes protocolos possuem fatores em comum, no que diz respeito a um solução segura, que dificultam ou descartam a sua utilização. Entre eles, relacionam-se:

Modelo de Paralelismo: a maioria dos métodos simplex em paralelo são baseados no modelo *mestre-escravo* no qual um processador é designado como mestre e sendo assim, é responsável por certas operações. No caso do simplex, a tarefa de armazenar a função objetivo é usualmente atribuída ao processador mestre que por sua vez, realiza sozinho a busca pela coluna pivô. Tal atribuição reduz o desempenho da solução visto que um total de $(m+n)$ comparações para a seleção da coluna pivô precisam ser executadas em sequência em um único processador. O mesmo efeito pode ser observado durante a atualização dos índices depois da operação de pivoteamento. Disconsiderando-se este modelo hierárquico e assumindo-se um simples modelo igualitário no qual cada processador realiza exatamente as mesmas operações em paralelo, sabendo-se que operações seguras necessitam de um limiar de participantes ou processadores reais, o *speed-up* advindo do uso de um algoritmo paralelo com divisão do problema não resulta em melhorias reais na complexidade computacional da solução.

Dependência da Estrutura: diversos métodos dependem da densidade da matriz e/ou de sua estrutura em blocos para ganhar em *speed-up*. Este tipo de vantagem não pode ser obtido em modelos seguros devido ao simples fato de que todas as entradas da matriz são representadas como variáveis seguras ou seja, valores criptografados. Encontrar quais são os valores iguais a zero, mesmo sabendo que a maioria dos valores o são, é uma tarefa muito complicada. Todos os elementos da matriz são criptografados e portanto, é impossível distinguir, pelo menos de uma maneira barata, se um dado elemento é igual a zero ou não, de modo que todos precisam ser considerados para a computação. Tal propriedade afeta, ou melhor, invalida qualquer tentativa de paralelismo que baseie seus resultados na estrutura do problema.

Complexidade das Operações: os métodos simplex revisado e ponto interior incluem operações de matrizes bastante complexas. Inversão de matriz, decomposição LU, entre outras, são operações bastante caras quando se tratando de elementos em claro e, mais ainda, quando se tratam de variáveis secretas, i.e., criptografadas.

6.3 Implementação

A análise teórica detalhada revelou que as abordagens consideradas são bastante similares no que diz respeito a complexidade de computação e comunicação.

A escolha por implementar o protocolo de *Programação Linear Segura em Paralelo* deve-se principalmente à sua simplicidade e intuitividade. A forma como as operações são paralelizadas é intuitiva: depois de ter-se evidenciado as operações que podem ser executadas em paralelo, o processo de distribuição de tarefas entre os processadores reais é bastante simples. A abordagem baseada no modelo BSP necessitaria o gerenciamento dos processadores abstratos, o que incluiria a criação e controle do modelo de abstração e hierarquização, e.g., distinção entre processador 0 e os demais, identificação do processador MIN_L, etc.

6.3.1 Implementação em Paralelo do PL Seguro

Esta seção descreve a implementação do núcleo do método simplex seguro e paralelo.

Um ponto de sincronização é definido como o momento durante o qual os participantes comunicam a fim de alcançar o próximo passo da computação. Considera-se dois modelos diferentes de sincronização:

Party Sync.: a sincronização ocorre a nível de participante. O participante espera que todas as *threads* terminem um passo de computação e somente depois disto, uma única mensagem é enviada, de participante para participante, o que permite que o próximo passo seja iniciado (ver Figura 6.2).

Thread Sync.: a sincronização acontece em nível de *thread*. As mensagens de sincronização (que determinam quando um determinado passo de computação termina e outro se inicia) são enviadas diretamente entre *threads*, sem a existência de um passo de sincronização a nível de participante (detalhes na Figura 6.3).

A escolha do modelo de sincronização influencia diretamente o desempenho da solução como um todo. Portanto, a implementação, testes e resultados focam na determinação de qual modelo de sincronização possui o melhor desempenho.

A implementação é baseada na linguagem de programação Java (Sun, versão 6) e RMI (*Remote Method Invocation*) é utilizado como *middleware* de comunicação, provendo abstração de alto nível para chamadas de procedimentos remotos.

A Figura 6.2 descreve todos os passos envolvidos no modelo de sincronização a nível de participante. De maneira resumida, depois da criação e inicialização dos participantes (distribuição das chaves), a lista de operações é sequencialmente processada. Um participante associa operações individuais a processadores enquanto existam processadores disponíveis.

Para o modelo de sincronização a nível de participante são criadas *threads* na máquina virtual Java e o sistema operacional é responsável por agendar tais *threads* a cada um dos processadores. É criado um número fixo de *threads* e associa-se a cada uma delas uma operação da lista de operações (multiplicações seguras) a serem executadas. Cada *thread* computa (ocupa o processador) até que seja necessário comunicar-se, quando então, ela sinaliza ao objeto de comunicação do participante. Este objeto coleta as entradas de todas as *threads* e realiza um *multicast* de uma única mensagem para os outros participantes. Cada uma das *threads* espera até que o objeto de comunicação retorne e somente então, continua com a computação. Após ter finalizado uma operação de multiplicação, uma *thread* recebe outra operação da lista e repete o procedimento. O programa chega ao fim quando a lista de operações estiver vazia, i.e., todas as operações tiverem sido executadas.

O segundo modelo de sincronização, a nível de *thread*, é descrito na Figura 6.3. Novamente, é criado um número fixo de *threads*, mas desta vez, cada uma delas mantém sua própria lista de operações a serem executadas. Cada *thread* seleciona uma operação da sua lista e inicia a execução até que seja necessário comunicar-se. Neste caso, cada *thread* possui seu próprio objeto de comunicação que por sua vez, é responsável pelo envio de mensagens à *thread* correspondente pertencente ao outro participante. A *thread* aguarda até que o objeto de comunicação retorne e então seleciona uma nova operação da sua lista. O programa termina quando todas as *threads* de todos os participantes tiverem executado todas as operações das suas listas.

A principal diferença entre os modelos de sincronização é com quem a comunicação é realizada. Na sincronização a nível de participante, cada participante é representado por um cliente e um servidor RMI, recebendo e enviando mensagens de/para outros participantes. As *threads* se comunicam localmente por intermédio de um objeto de comunicação, por participante. No caso da sincronização a nível de *thread*, a comunicação é distribuída entre as *threads*. Cada uma delas implementa um cliente e um servidor RMI e, sendo assim, é capaz de diretamente invocar serviços RMI de outros participantes. Um participante também implementa um cliente e um servidor RMI a fim de possibilitar a execução da fase de inicialização (distribuição de chaves) e comunicação entre participantes, quando necessário.

Foram analizados quatro casos de testes para cada modelo de sincronização. Variou-se o número de *cores* disponíveis em cada participantes dentro da faixa de 4 a 16, em passos de 4, ou seja, $4, 8, 12, 16$.

Operações de multiplicação segura foram implementadas de acordo com o protocolo introduzido por Cramer e Damgard em [29]. Note que uma multiplicação segura inclui não somente comunicação multi-parte mas também uma decifragem. Um protocolo especial para decifragem homomórfica com limiar foi implementado, baseado nas idéias de Damgard, Jurik e Nielsen propostas em [30], já descritas

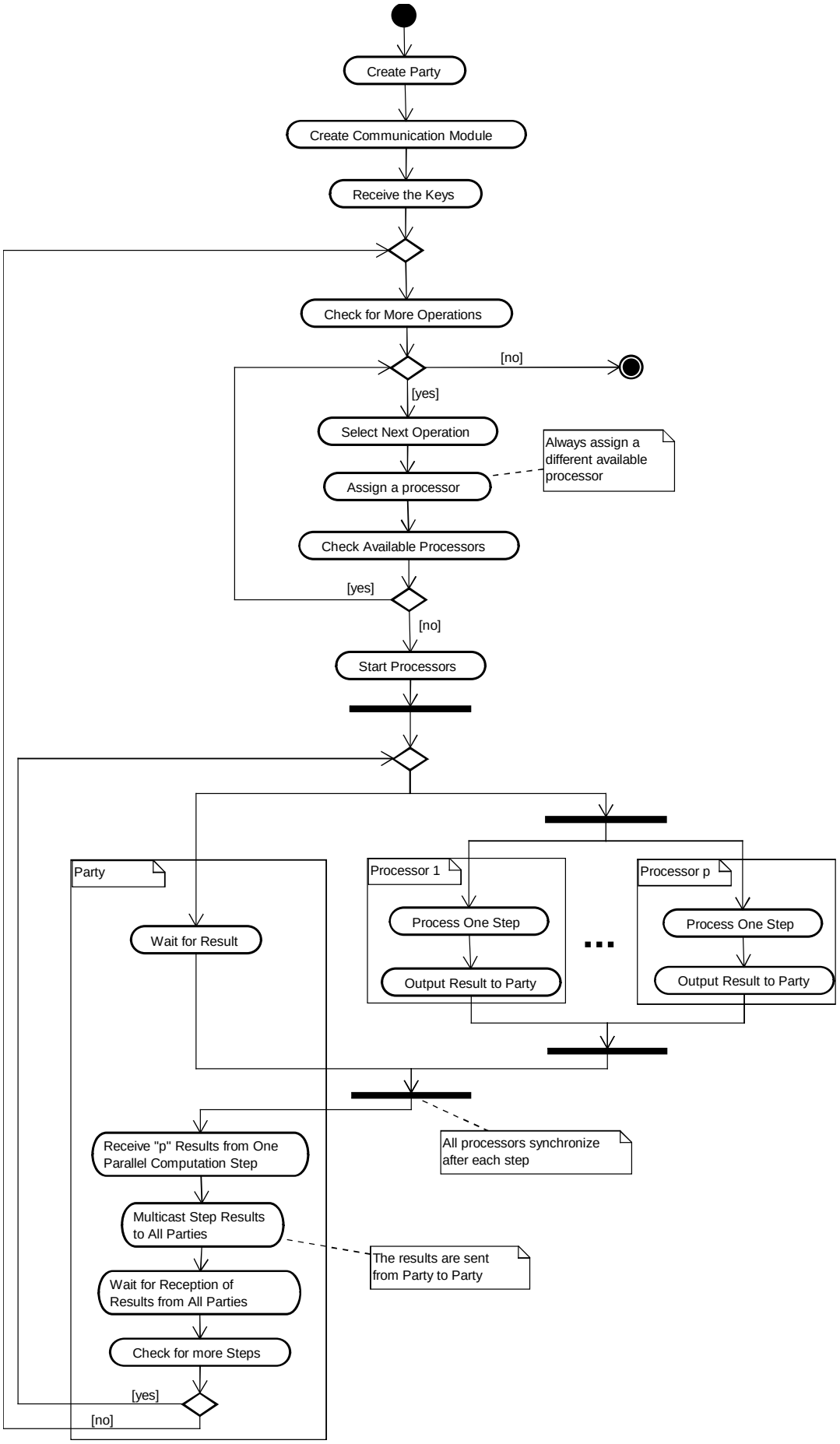


Figura 6.2: Diagrama UML de atividade para o modelo de sincronização a nível de participante

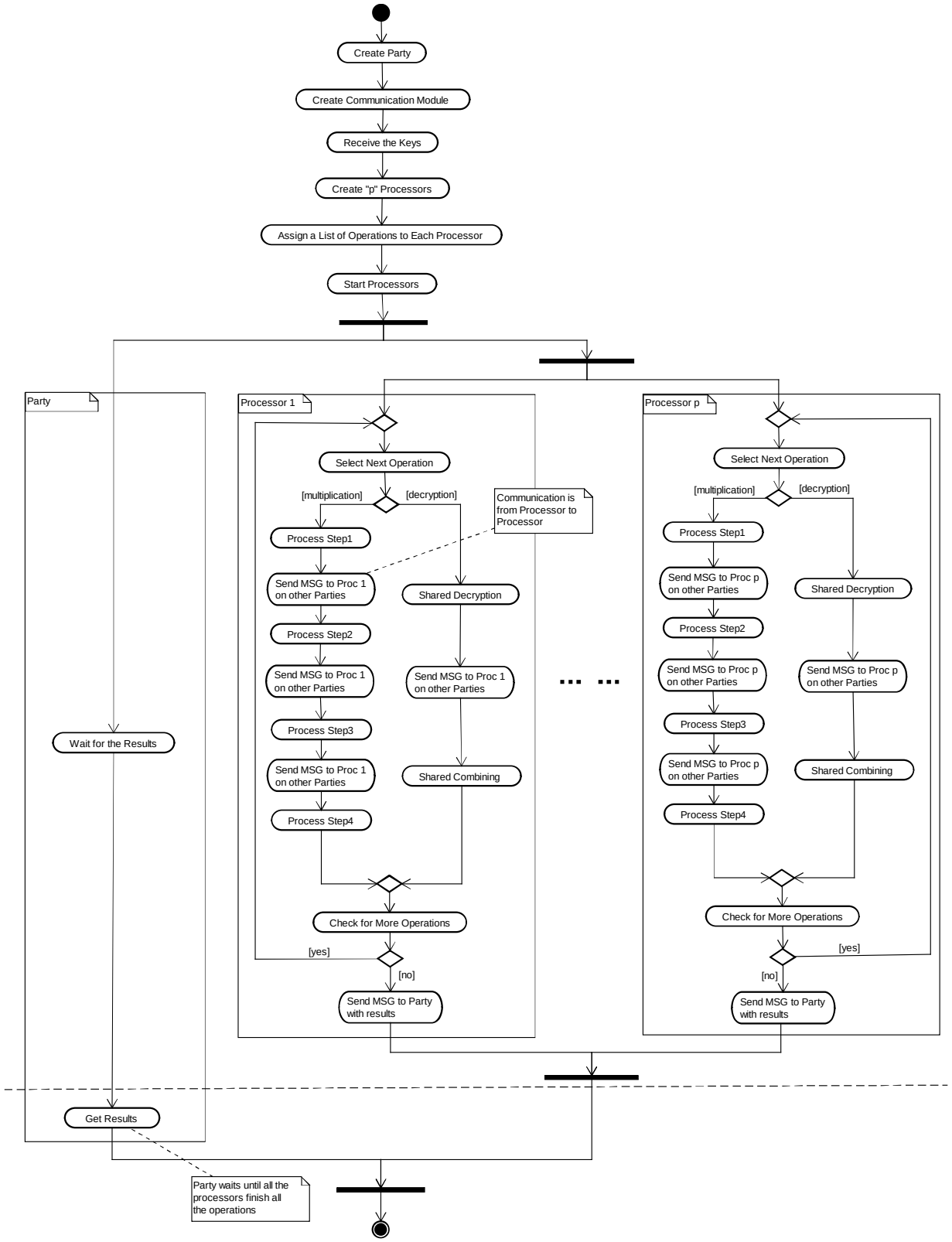


Figura 6.3: Diagrama UML de atividade para o modelo de sincronização a nível de *thread*

em detalhes na Seção 2.3. Qualquer participante pode criptografar dados (utilizando a chave pública distribuída durante a fase de inicialização), mas somente um número de participantes maior que o limiar é capaz de decifrar um dado texto secreto, i.e., pelo menos $t + 1$ participantes (partes da chave privada) são necessários. Este dois algoritmos foram escolhidos porque possuem complexidade constante de rodada e são capazes de realizar computações seguras com n-partes.

6.3.1.1 Resultados Experimentais

Duas máquinas dedicadas equipadas com 16 cores (Intel Xeon E7330 @ 2.40GHz) cada foram utilizadas para os experimentos. Ambas as máquinas possuem 64GB de memória RAM e são conectadas por uma rede 100Mb/s. O sistema operacional utilizado foi o SUSE Linux Enterprise Server 10 SP2, kernel "2.6.16.60-0.21-smp" x86_64 (instruções em 64 bits). Para efeito de simplicidade e, especialmente, disponibilidade de recursos, somente o cenário com dois participantes foi implementado para os testes.

A fim de simular os quatro casos de teste com número distintos de *cores*, fez-se uso de funções especiais do kernel GNU/Linux para abilitar e desabilitar *cores*. Cada um dos *cores* possui uma variável que pode ser utilizada para definir seu estado, i.e., 0 para *offline* e 1 para *online*.

A configuração básica dos casos de teste é a seguinte: cada máquina representa um participante, tendo conhecimento da chave pública e de uma das partes da chave secreta (considerando-se um sistema criptográfico homomórfico e baseado em limiar), ambas distribuídas durante a fase de inicialização. Cada participante conhece o número de operações bem como a sequência na qual elas devem ser executadas, a qual é exatamente a mesma para todos os participantes. Sendo assim, os participantes instanciam um número específico de *threads* para serem executadas em paralelo. Tal número depende do caso de teste.

O objetivo inicial é descobrir qual dos modelos de sincronização propostos apresenta melhor desempenho para resolução de operações seguras. A hipótese de trabalho afirma que é melhor ocupar cada um dos processadores com uma única *thread* por vez.

Procedimento Consideram-se 4 casos para o número de *threads* para cada um dos 4 casos de testes para o número de *cores* para cada um dos modelos de sincronização.

Cada caso de teste possui um número fixo de processadores reais (p) e uma variação no número de *threads* é aplicada. Tal variação depende do número de processadores disponível e pode ser representado na seguinte forma:

$$\text{variação} = [(t = p - 4), (t = p), (t = p + 4), (t = p + 8)]$$

Considere, por exemplo, o caso com 12 processadores reais. A faixa de variação do número de *threads* é $[8, 12, 16, 20]$. Note que para o caso em que o número de processadores é igual a 4, é claramente impossível utilizar-se $t = p - 4 = 0$ *threads*. Para tal, utiliza-se o valor $t = 2$.

Ainda, considera-se somente um passo inferior a $(t = p)$ visto que os resultados são diretos. Assuma um número p de processadores. A utilização de um número de *threads* inferior ao número de processadores/cores, i.e., $t < p$, significa que haverão processadores desocupados (em ambos os modelos de sincronização) e portanto, a execução do protocolo levará um tempo superior do que os outros casos. Os casos de interesse estão na faixa para a qual $t \geq p$.

Resultados Os resultados são referentes à média aritmética entre 50 experimentos para cada um dos casos de teste. A Tabela 6.1 contém os resultados de todos os casos de teste: tempo de execução absoluto para cada um dos números de *threads* em cada um dos casos para o número de processadores.

Note que o número de multiplicações seguras é definido como o mínimo múltiplo comum entre os valores para o número de *threads*. Com tal suposição garante-se que o número de operações é sempre igual a um múltiplo do número de *threads* e assim, não será necessário nenhum passo extra ao fim do protocolo. De fato, isto pode não acontecer em todos os casos reais, i.e., número de operações múltiplo do número de processadores, mas é necessário para que se faça uma comparação justa entre as medições.

O modelo de sincronização a nível de *thread* foi mais rápido em 14 dos 16 casos de teste aplicados. A razão para tal melhoria está relacionada à maneira com que as *threads* são sincronizadas. A diferença deve-se ao fato de que no caso da sincronização a nível de participante, depois de terminado um passo de computação, cada uma das *threads* sinaliza o participante com o seu resultado. Este por sua vez, precisa

Tabela 6.1: Resultados referentes ao tempo de execução (RT) médio para os modelos de sincronização (ms)

Processadores	Operações	Threads	RT Party Sync	RT Thread Sync
4	24	2	12486.5	12542
		4	6732	6721
		8	6911	7762
		12	7215	6966.5
8	48	4	12738.5	12309.5
		8	9463	6232.5
		12	8626	7113.5
		16	8681	7396
12	240	8	32824	28310
		12	31713.5	21405
		16	28735.5	22496
		20	28475	25772.5
16	240	12	21597.5	19128
		16	19805.5	14924
		20	21763.5	17832
		24	22232	18576.5

esperar até que todas as *threads* tenham sinalizado para somente então, comunicar. A sincronização é realizada localmente por intermédio do acesso a um único objeto. Tal operação acontece em cada uma das multiplicações seguras, que neste caso é igual a $5*a/\rho$, onde a é o número de multiplicações seguras e ρ corresponde ao número de processadores reais. Por outro lado, no modelo de sincronização a nível de *thread* tal operação ocorre somente uma vez, ao final, depois que todas as *threads* terminaram sua lista de operações.

Durante a execução das operações de multiplicação, cada *thread* precisa também sincronizar com sua respectiva *thread*, pertencente ao outro participante, por meio de chamadas RMI. Visto que cada *thread* se comunica com outro participante, o tempo de exeucação desta operação é limitado pelo atraso na comunicação. Este último aspecto é o tema da próxima rodada de experimentos.

6.3.2 Otimizando o Número de *Threads*

Os resultados da comparação de diferentes abordagens para PL seguro em paralelo revelaram que não existe diferença relevante entre elas, no que diz respeito a complexidades de computação e comunicação. A implementação da melhor opção e a avaliação dos modelos de sincronização demonstraram que sincronização a nivel de *thread* propicia melhores resultados, i.e., menor tempo de execução absoluto. Os resultados apresentados na seção anterior também chamam a atenção para o fato da possibilidade da existência de um número ótimo de *threads* para um dado número de processadores. Sendo assim, é de interesse descobrir como determinar tal número de *threads* a fim de maximizar o desempenho de uma dada aplicação paralela.

6.3.3 Resultados Experimentais

Os primeiros resultados mostrados na Seção 6.3.1.1, Tabela 6.1 já indicam uma primeira hipótese para o número ótimo de *threads*: o menor tempo de execução é obtido quando o número de *threads* é igual ao número de processadores. As Figuras 6.4 e 6.5 contém resultados que suportam esta hipótese.

Nota-se a forma dos gráficos para o caso de sincronização a nivel de *thread* na qual o melhor tempo de execução foi obtido quando $t = p$ para todos os casos. Em tais casos, exatamente uma *thread* é associada a cada processador de modo que não existe *overhead* de agendamento, o que rende melhores resultados.

Em um cenário real de PDCL, a comunicação entre os participantes não necessariamente acontece em uma LAN com nos experimentos anteriores. Tais problemas envolvem múltiplos participantes que estão normalmente geograficamente dispersos e portanto, as condições da rede exercem grande influência sobre o desempenho dos protocolos. Particularmente, em tais cenários, o atraso na rede

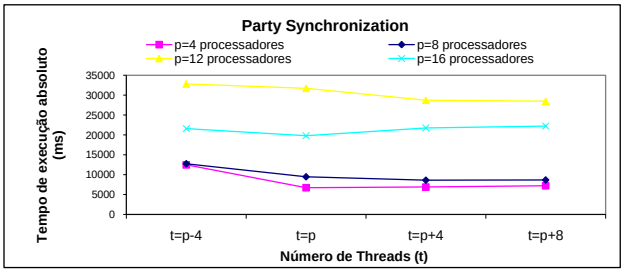


Figura 6.4: Tempo de execução para o caso party sync. (média em ms)

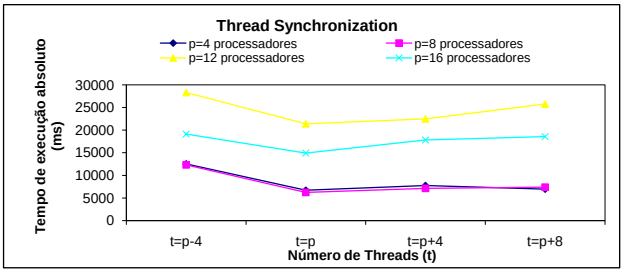


Figura 6.5: Tempo de execução para o caso thread sync. (média em ms)

possui um papel muito importante.

O próximo experimento tem o objetivo de medir a dependência do atraso de comunicação no tempo de execução de um protocolo. Obviamente, um atraso maior na rede faz com que as *threads* esperem mais tempo para o retorno do objeto de comunicação. É possível diminuir o efeito desta espera simplesmente agendando mais *threads* que podem então ser executadas nos processadores desocupados. Um dos objetivos dos experimentos é determinar o número ótimo de *threads* como função do atraso da rede.

Uma configuração especial é necessária para que se possa artificialmente introduzir um atraso controlado na comunicação entre os participantes. Para tal, as máquinas foram organizadas de maneira diferente, de modo que cada uma delas recebeu um segundo endereço IP da classe *192.168.**L**.2* e nova configuração de *gateway* apontando para o endereço *192.168.**L**.1*. Uma terceira máquina executando o sistema operacional FreeBSD[1] foi introduzida no modelo para servir de *gateway* padrão para todas as sub-redes (ver Figura 6.6). FreeBSD foi escolhido porque suporta implementação nativa do *software Dummynet* [85] e *ipfw*[2] que fornecem as funcionalidades desejadas para este tipo de simulação.

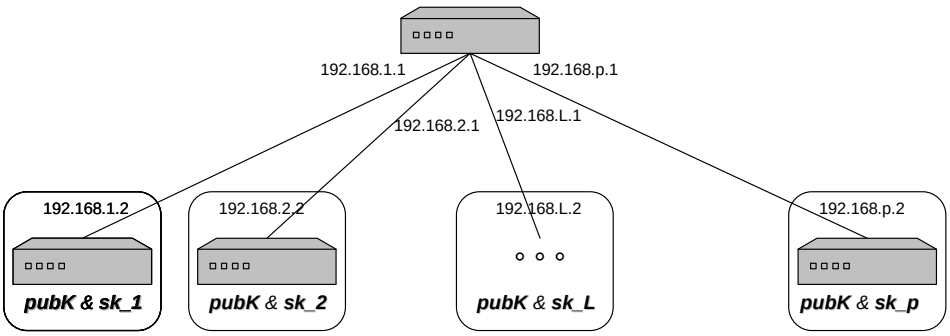


Figura 6.6: Topologia da rede para os testes com atraso

A comunicação entre os participantes/processadores é então realizada via um *gateway* físico que é capaz de redirecionar todas as menagens e artificialmente introduzir atraso na rede. Todas as chamadas RMI são forçadas a passar pelo *gateway* antes de chegar ao seu destino. Visto ser possível definir o valor exato do atraso introduzido na rede, esta configuração permite a avaliação da influência da comunicação no tempo total de execução dos protocolos propostos. Nota-se que quando o caso de atraso nulo é simulado, a presença do *gateway* não afeta os resultados de maneira significativa.

Os casos de teste são os mesmo anteriormente descritos e utilizados na Seção 6.3.1, i.e., 4,8,12,16 processadores para os modelos de sincronização a nível de participante e de *thread*. A Figura 6.7 traz os resultados referentes à média entre 50 execuções para o tempo total de execução necessário para realizar um única multiplicação segura em cada um dos três diferentes cenários: *0, 100 e 200ms*. Adicionalmente, a Figura 6.7 mostra como os diferentes modelos de sincronização são afetados pela presença de atraso na rede.

Os resultados revelam que ambos os modelos de sincronização são igualmente afetados e apesar de o modelo de sincronização a nível de *threads* requerer um número maior de chamadas RMI, i.e., cada

[1] www.freebsd.org/

[2] http://www.freebsd.org/doc/en/books/handbook/firewalls-ipfw.html

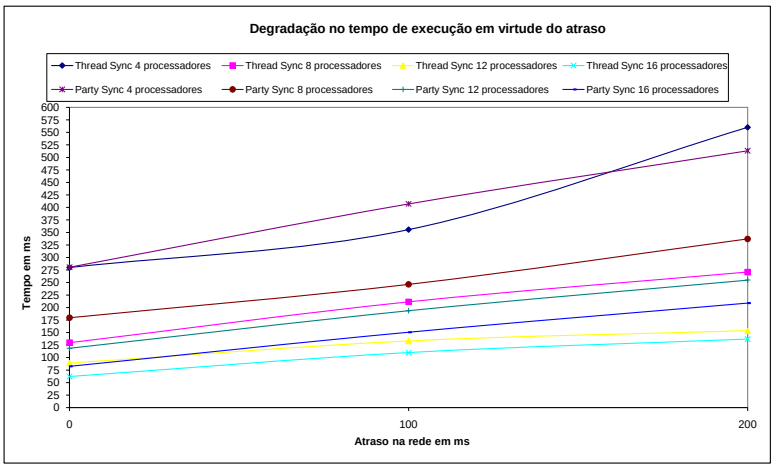


Figura 6.7: Degradação no tempo de execução (por operação segura) em relação ao atraso na rede para diferentes casos de número de processadores e sincronização

thread é representada por um servidor e um cliente RMI que realizam chamadas a outros servidores, a influência do atraso não é maior do que no caso da sincronização a nível de participante.

Quando compara-se o número de mensagens enviadas em ambos os modelos de sincronização, a diferença é notável. Por um lado, no modelo de sincronização a nível de participantes são enviadas $p(p-1)$ mensagens. Por outro lado, no modelo de sincronização a nível de *threads* são enviadas $tp(p-1)$ mensagens (onde t corresponde ao número de *threads*). No entanto, tais mensagens são independentes e podem ser realizadas em paralelo. Assuma que todas as *threads* comunicam exatamente no mesmo instante. Isto é equivalente a uma única mensagem sendo transmitida, processo semelhante ao que acontece no caso da sincronização a nível de participante. Até mesmo o tamanho das mensagens é reduzido: enquanto no caso de sincronização a nível de participantes uma mensagem contém dados de todas as *threads* em execução, no caso de sincronização a nível de *thread*, a soma de todas as mensagens é equivalente, em tamanho, àquela única mensagem. Por exemplo, o modelo de sincronização a nível de *thread* é mais rápido em todos os resultados para os casos de 8, 12 e 16 processadores quando existe atraso de comunicação (ver Figura 6.7).

Aparte da comparação dos modelos de sincronização, foi também avaliada a influência do atraso no tempo absoluto de execução em relação ao número de *threads* utilizado. A Figura 6.8 mostra os resultados para a deterioração relativa dos tempos de execução em função de um número crescente de *threads*.

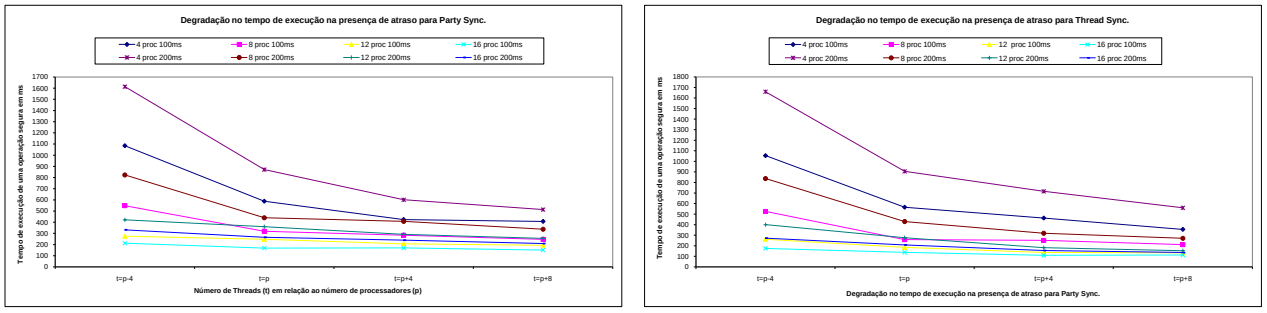


Figura 6.8: Desempenho dos modelos de sincronização em uma rede com atraso

Com o aumento do número de *threads* é possível se reduzir os efeitos negativos do atraso na rede. O tempo de espera adicional (das *threads*) é utilizado pelas *threads* inativas mas prontas para executar, de modo que a utilização total dos processadores aumenta. Desta maneira, é possível utilizar a capacidade máxima dos processadores, mesmo na presença de atraso.

A complexidade de comunicação é definida pelo tamanho total das mensagens enviadas durante a execução do protocolo. Atraso na rede afeta diretamente a comunicação, i.e., quanto maior o atraso mais lenta é a comunicação e pior é a performance do protocolo. O aumento do número de *threads*

que são executadas simultaneamente permite que um número maior de mensagens sejam enviadas em paralelo, em um único passo, o que por sua vez, reduz o efeito do atraso. Em outras palavras, o efeito do atraso é o mesmo quando se enviam x mensagens em paralelo ou quando se enviam y mensagens em paralelo, com $x \gg y$, desde que exista largura de banda suficiente.

Assumindo-se que chamadas RMI são executadas em paralelo, o tempo gasto para realizar *multi-thread* em um único processador se torna menor do que o tempo gasto em comunicação, devido ao atraso na rede. Escolhendo-se corretamente o número de *threads* por processador, em uma rede com atraso, é possível conseguir-se um desempenho aproximadamente tão bom quanto quando considera-se o mesmo modelo em uma rede sem atrasos.

6.4 Agendamento Adaptativo

O modelo de sincronização a nível de *thread* é mais rápido do que o modelo a nível de participantes e ambos os modelos são similarmente afetados pela presença de atraso na rede. Mostrou-se haver um número ótimo de *threads* (ou um intervalo ótimo) para o qual o desempenho do sistema atinge seu máximo, i.e., quando $t = p$ para uma rede sem atraso. Adicionalmente, é possível reduzir-se o efeito do atraso aumentando-se o número de *threads* por processador.

Todos os testes executados até agora fizeram uso de uma configuração estática, na qual o número de *threads* era definido *a priori* e então, permanecia fixo. No entanto, situações práticas requerem uma configuração mais dinâmica, na qual o número ótimo de *threads* é escolhido de maneira a adaptar-se às condições da rede. Em aplicações reais, em que os participantes interagem utilizando a Internet como infraestrutura de comunicação, não é possível assumir-se um valor fixo para o atraso durante toda a execução de um protocolo simplex seguro e paralelo.

Desta forma, propõe-se um algoritmo que é capaz de procurar dinâmicamente o número ótimo de *threads*, identificando possíveis atrasos na rede pela observação do *throughput* ou outro indicador de desempenho e agir adaptando a escolha deste número a fim de reduzir os efeitos advindos da presença de atrasos.

6.4.1 Algoritmo

A idéia básica do algoritmo proposto é procurar por um possível número ótimo de *threads* por meio da observação de um indicador de desempenho.

Ele inicia descobrindo quantos procesadores reais estão disponíveis. Tal valor é utilizado como uma primeira tentativa para o número ótimo: resultado baseado nas discussões apresentadas na Seção 6.3.1, e nas Figuras 6.4 e 6.5. O número de *threads* nunca é menor do que este valor: suposição baseada nas discussões da Seção 6.3.1.1.

A tendência natural é melhorar o desempenho aumentando-se o número de *threads* e observando-se o indicador. O algoritmo procura pelo número máximo de *threads* por fatia de tempo, ou tempo de execução. Assim, o indicador de desempenho é definido como a razão entre o número de *threads* atual e o tempo de execução absoluto para um certo número de operações seguras (multiplicações seguras, neste caso).

Formalmente, assuma $\alpha = \frac{NoT}{RT}$, onde α é o indicador selecionado, NoT é o número de *threads* e RT corresponde ao tempo de execução medido. Consequentemente, quanto maior o valor de α, melhor. Considerando-se uma rede com atraso, o tempo de execução é diretamente afetado, i.e., assume valores mais altos, o que reflete em valores de α menores. Se o algoritmo for capaz de simultaneamente aumentar o número de *threads* até um certo ponto, o valor da razão α eventualmente aumentará e possivelmente alcançará um valor aproximadamente tão bom quanto antes (cenário sem atrasos).

A Figura 6.9 mostra exatamente como o algoritmo de agendamento adaptativo funciona. Os algoritmos 18, 19 e 20 contém o pseudo código da implementação do algoritmo proposto. Os pontos principais do funcionamento do algoritmo são descritos em seguida.

O primeiro passo é definir uma direção de busca e a primeira hipótese é aumentar o número de *threads*. Após cada iteração, o algoritmo compara a razção atual (α_i) com a melhor ração conhecida (α_b). Se o resultado for positivo, o algoritmo mantém a mesma direção de busca, i.e., continua aumentando o número de *threads* – e repete a última ação. Se, por outro lado, o resultado da comparação for negativo, consideram-se duas possíveis razões: (i) o algoritmo já atingiu a região ótima e precisa refinar

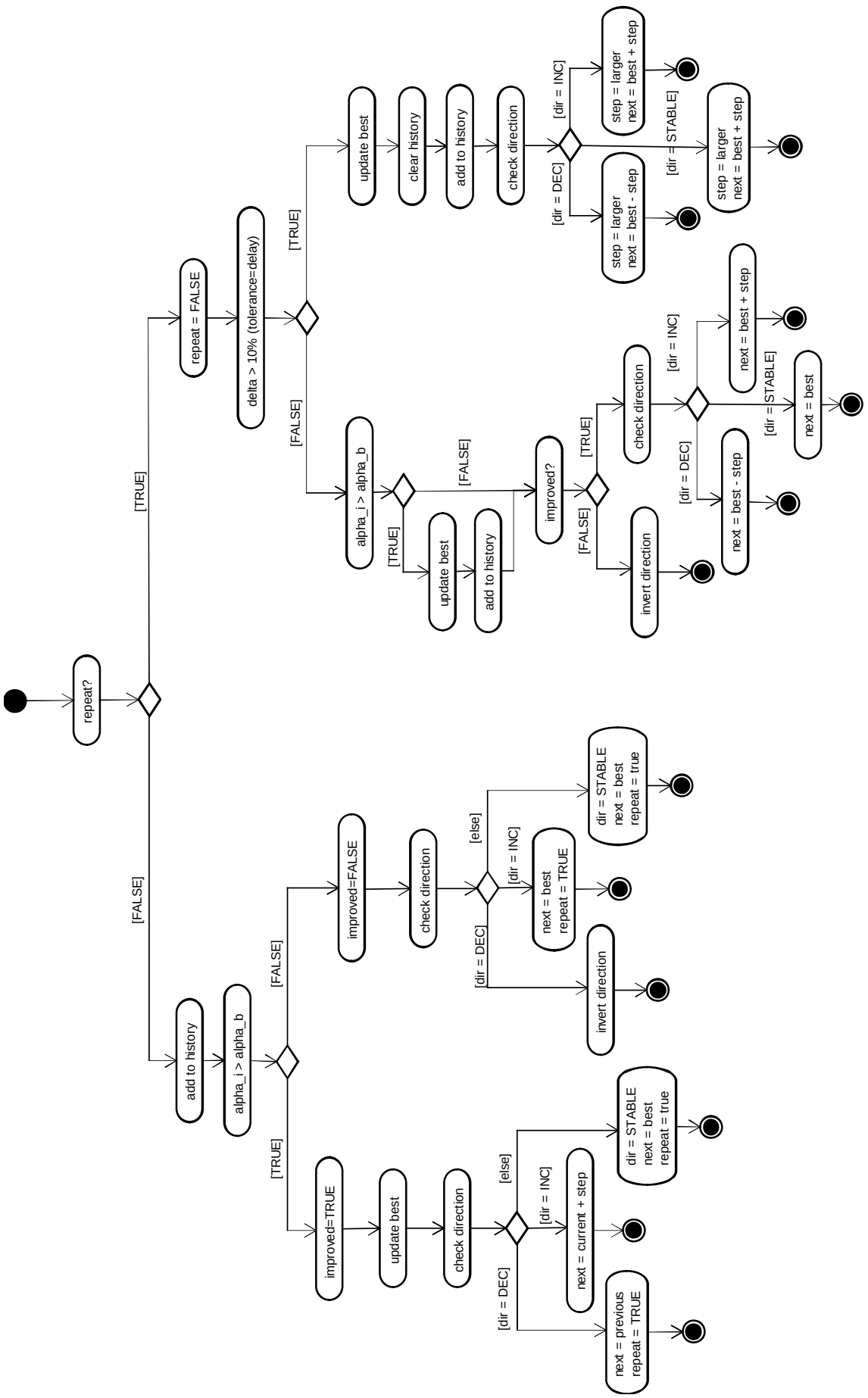


Figura 6.9: Fluxograma do algoritmo de agendamento adaptativo

a busca em torno do melhor valor conhecido, ou (ii) houve variação nas condições da rede (aumento ou diminuição do atraso).

Para o caso da opção (i), o algoritmo assume que o provável número ótimo de *threads* está na vizinhança do melhor valor atual e sendo assim, aplica-se uma técnica de "balanço". O próximo número de *threads* é escolhido invertendo-se a última direção de busca e aplicando-se um passo de tamanho igual a metade da diferença entre o número de *threads* anterior/próximo (valor absoluto, obtido do histórico de valores) e o melhor valor atual. A Figura 6.10 traz um exemplo das ações tomadas quando o o caso (i) é considerado.

Utiliza-se uma árvore binária não balanceada para identificar o anterior ou o próximo valor absoluto para o número de *threads* em torno do melhor valor atual. Tal estrutura também ajuda a decidir qual direção de busca dever ser tomada. Além disto, diferentes árvores são utilizadas para diferentes valores de atrasos, de modo que a comparação com o melhor atual seja justa e consistente.

Na Figura 6.10, os números dentro dos círculos representam o número de *threads* utilizado em uma determinada iteração que é representada pelo número situado sobre os círculos. Adicionalmente, a letra *b* corresponde ao atual melhor valor para o número de *threads*. Seguindo-se a sequência de iterações, fica claro como a técnica de "balanço" funciona e quão útil é representar-se o histórico como uma árvore binária.

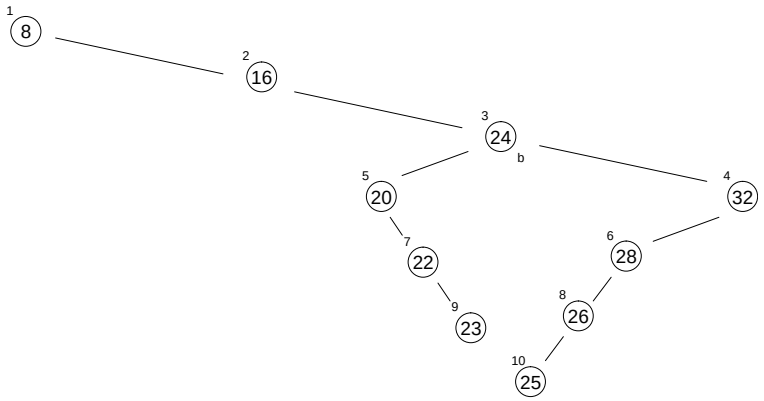


Figura 6.10: Exemplo de um histórico representado como uma árvore binária

Para o caso da opção (ii), o algoritmo assume a existência de atraso na rede e a ação imediata é reduzir o tamanho do passo no qual o próximo número de *threads* será calculado.

Visto não ser possível medir diretamente o atraso na rede, o método utilizado para distinguir-se entre os dois casos é o seguinte: o algoritmo repete o atual melhor valor para o número de *threads* e compara as razões obtidas. Se a diferença for maior do que um limite definido *a priori*, e.g., $\Delta > 10\%$, o algoritmo assume que existe atraso na rede, caso contrário, considera-se o caso (i). Note que mesmo que não haja atraso na rede, a ação de utilizar o atual melhor valor não é totalmente restritiva, visto que o algoritmo simplesmente repete a melhor tentativa. Esta abordagem pode ser considerada conservadora demais em oposição a uma possível abordagem *simplesmente use um passo maior* (a qual eventualmente poderia levar a melhores resultados, caso realmente exista atraso) No entanto, considera-se a primeira como ideal no sentido de que o algoritmo é capaz de corretamente identificar mudanças nas condições da rede e tomar a decisão acertada na próxima iteração. Além disto, repetindo-se o atual melhor valor não é considerada uma ação negativa, com respeito ao tempo de execução absoluto.

Algorithm 18: Adaptive Scheduling Algorithm

Input: Current number of threads NoT_i and the respective performance indicator α_i
Output: The next number of threads to be used NoT_{i+1}

```
1  if H = ∅ then
2  |   H ∪ (NoT_i, α_i);
3  |   NoT_b ← NoT_i;
4  |   α_b ← α_i;
5  |   dir ← INC;
6  |   return NoT_i + baseStep;
7  else
8  |   if repeat then
9  |   |   NoT_{i+1} ← HandleRepeat(NoT_i, α_i, H, NoT_b, α_b);
10 |   else
11 |   |   NoT_{i+1} ← HandleNoRepeat(NoT_i, α_i, H, NoT_b, α_b);
12 |   end
13 |   return NoT_{i+1};
14 end
```

Algorithm 19: HandleRepeat

Input: Current number of threads NoT_i and the respective performance indicator α_i, history H and the best known values (NoT_b, α_b)
Output: The next number of threads to be used NoT_{i+1}

```
1  if canInferOnDelay(α_i) then
2      newStepSize ← baseStep × delayIncreaser;
3      switch dir do
4          case INC
5              NoT_{i+1} ← NoT_b + newStepSize;
6          end
7          case DEC
8              NoT_{i+1} ← NoT_b − newStepSize;
9              if NoT_{i+1} < baseStep then
10                 NoT_{i+1} ← baseStep;
11             end
12         end
13         case STABLE
14             NoT_{i+1} ← NoT_b + newStepSize;
15         end
16     end
17     H = ∅;
18     H ∪ (NoT_i, α_i);
19     NoT_b ← NoT_i;
20     α_b ← α_i;
21 else
22     if α_i ≥ α_b then
23         NoT_b ← NoT_i;
24         α_b ← α_i;
25         H ∪ (NoT_i, α_i);
26     end
27     if improving then
28         switch dir do
29             case INC
30                 NoT_{i+1} ← NoT_b + baseStep;
31             end
32             case DEC
33                 NoT_{i+1} ← NoT_b − baseStep;
34                 if NoT_{i+1} < baseStep then
35                     NoT_{i+1} ← baseStep;
36                 end
37             end
38             case STABLE
39                 NoT_{i+1} ← NoT_b;
40             end
41         end
42     else
43         NoT_{i+1} ← invertSearchDir();
44     end
45 end
46 repeat ← false;
47 return NoT_{i+1}
```

Algorithm 20: HandleNoRepeat

Input: Current number of threads NoT_i and the respective performance indicator α_i, history H and the best known values (NoT_b, α_b)
Output: The next number of threads to be used NoT_{i+1}

```
1  if α_i ≥ α_b then
2      improving ← true;
3      switch dir do
4          case INC
5              NoT_{i+1} ← NoT_b + baseStep;
6          end
7          case DEC
8              NoT_{i+1} ← NoT_b;
9              repeat ← true;
10         end
11         case STABLE
12             NoT_{i+1} ← NoT_i;
13             repeat ← true;
14         end
15     end
16     NoT_b ← NoT_i;
17     α_b ← α_i;
18 else
19     improving ← false;
20     switch dir do
21         case INC
22             NoT_{i+1} ← NoT_b;
23             repeat ← true;
24         end
25         case DEC
26             NoT_{i+1} ← invertSearchDir();
27         end
28         case STABLE
29             NoT_{i+1} ← NoT_b;
30             repeat ← true;
31         end
32     end
33 end
34 H ∪ (NoT_i, α_i);
35 return NoT_{i+1}
```

6.4.2 Resultados Experimentais

O primeiro passo consiste em incluir-se o algoritmo de agendamento adaptativo na atual implementação do algoritmo de PL seguro e paralelo. Com base nos resultados anteriores, selecionou-se o modelo de sincronização a nível de *thread*. O algoritmo da Figura 6.9 é então invocado depois de cada iteração do protocolo e cada uma das *threads* recebe exatamente uma operação de cada vez. Depois que os processadores computam (independentemente) suas respectivas operações seguras, o tempo de execução é medido e o valor atual de α é calculado.

A Figura 6.11 apresenta os resultados (na média) de dois casos de testes: 8 e 16 processadores na presença de atraso. Para cada valor de atraso (em ms), i.e., $\{0, 50, 100, 150, 200, 250\}$, uma série de 50 testes foram executados. Para cada teste, 960 para o caso de 16 processadores e 720 multiplicações seguras para o caso de 8 processadores, foram executadas.

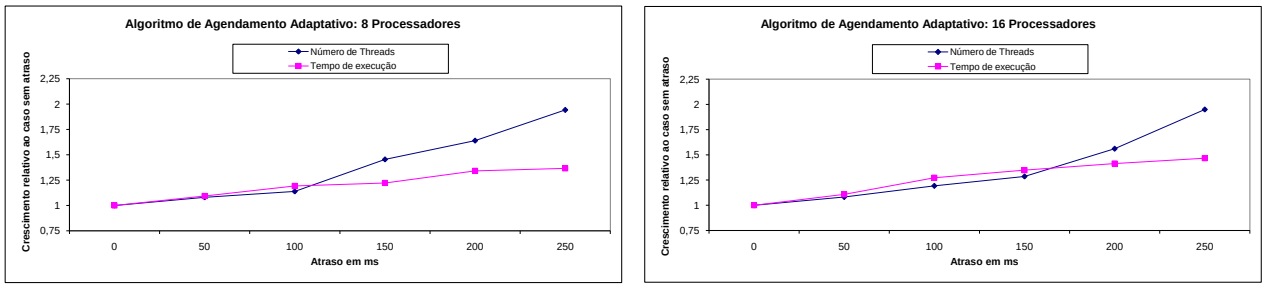


Figura 6.11: Desempenho do algoritmo de agendamento adaptativo para diferentes valores de atraso

O objetivo da primeira bateria de testes (Figura 6.11) é mostrar como o número de *threads* cresce com a presença de atraso na rede. Assim, os valores para melhor número de *threads* e tempo absoluto de execução são relativos ao caso para o qual não existe atraso. Como esperado, o algoritmo busca por um valor ótimo que aumenta em uma taxa maior do que o aumento no tempo de execução, a fim de compensar o atraso na rede. Tome o gráfico referente ao caso de 8 processadores, por exemplo. O número ótimo de *threads* para um atraso igual a 250ms é aproximadamente o dobro do valor referente ao caso no qual não existe atraso, enquanto que o tempo de execução é aproximadamente 25% maior. Note que o algoritmo oferece a possibilidade da escolha de um passo mais agressivo para aumento do número de *threads*, decisão tomada antes da execução do protocolo.

A próxima tarefa consiste avaliar como o agendamento adaptativo se comporta na presença de atrasos variáveis. O cenário utilizado para estes testes é bastate parecido com o cenário utilizado anteriormente. A principal diferença, no entanto, é a introdução de um *script* que adiciona (ou remove) pequenos atrasos na rede, de maneira aleatória. Em paralelo, o número de *threads* utilizado em cada iteração, assim como o valor do atraso são armazenados (*logging*) para posterior comparação e análise.

A Figura 6.12 mostra um dos 100 testes executados com máquinas de 16 *cores* resolvendo 960 opearções seguras cada. Tal exemplo evidencia as ações tomadas pelo algoritmo a fim de dinamicamente buscar pelo número ótimo de *threads* mesmo na presença de atrasos variáveis.

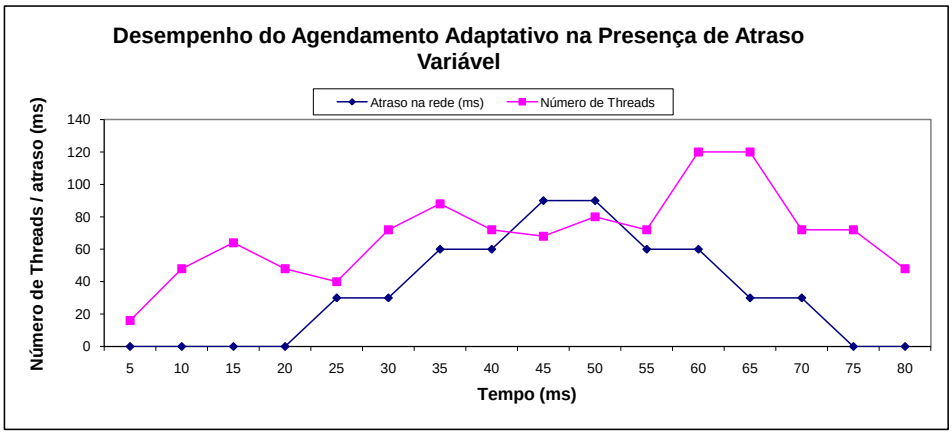


Figura 6.12: Desempenho do algoritmo de agendamento adaptativo com atraso variável

Não existe atraso nos primeiros 20 segundos de test, o que permite que o algoritmo aumente o número de *threads* até um dado limite (onde α pára de aumentar) e inicie a busca por um valor ótimo na vizinhança do atual melhor. No entanto, entre 20-45 segundos, pequenos atrasos são introduzidos na rede. O algoritmo identifica a presença de atrasos e começa a procurar por um novo ótimo, que agora precisa ser mais alto do que o atual melhor, mas ainda limitado pelo ponto onde α pára de aumentar (25-35 segundos). Como o valor do atraso continua a aumentar, a ação do algoritmo é aumentar o tamanho do passo a fim de diminuir o efeito do atraso mais rapidamente, até que um novo limite seja alcançado. A partir de 50 segundos, a situação torna a mudar e o valor do atraso é reduzido. O gráfico mostra o exato momento em que o algoritmo identifica a mudança no atraso: entre 60 e 65 segundos, onde o número de *threads* é repetido, o que permite ao algoritmo ter certeza de que houve uma variação no atraso e, sendo assim, reduzir o número de *threads* de acordo com a redução no atraso, até que o teste chega ao fim.

Cenários de atraso variável foram avaliados, mas naturalmente existe uma discussão na comunidade científica em relação a utilização de um gerador de tráfego, ao invés de um *script*, para obter resultados mais realísticos. Obviamente, a avaliação realizada aqui é capaz de capturar somente a variação de atrasos introduzida pelo *script*. Trabalhos futuros incluiriam explorar o desempenho ou adaptabilidade do algoritmo em condições reais de rede.

6.5 Considerações

Problemas de otimização de cadeias logísticas envolvem diversos parceiros de negócio. Foi provado que o compartilhamento de informações em cadeias logísticas melhora o desempenho, mas os participantes se recusam a compartilhar devido ao medo do vazamento de informação, visto que o envolvimento em problemas deste tipo frequentemente envolve o compartilhamento de informações vitais à sobrevivência dos negócios. Computação segura multi-parte oferece a garantia de segurança que pode ser capaz de convencer os participantes a compartilhar as informações necessárias para a otimização do plano diretor da cadeia logística. Não há necessidade de terceira parte confiável ou canais seguros de comunicação. De fato, não se faz necessário o estabelecimento de nenhum nível de confiança entre os participantes. Com CSM é possível obter-se uma segurança provável de que as entradas não estão sendo reveladas sem autorização. As computações são provadas seguras e corretas mesmo na presença de adversários passivos, i.e., participantes que arbitrariamente corrompem resultados intermediários como uma tentativa de obter informações sobre os dados privados de outros participantes. Desta forma, CSM previne a revelação dos dados de entrada e é capaz de resolver os problemas de otimização da cadeia logística. Ela oferece ambos: o plano ótimo sem revelar nenhum dado adicional e os riscos associados. A desvantagem é a implementação, que usualmente é bastante complicada e lenta: eficiência é um aspecto chave.

Técnicas de otimização como paralelismo são comumente utilizadas. Um exemplo é o protocolo proposto por Toft [93] para programação linear segura no qual se faz o uso do conceito de operações em pacotes (tantas operações quanto possível são executadas em paralelo) a fim de reduzir a complexidade de rodada. Ele assume recursos ilimitados para tal paralelismo, o que não é prático e portanto, requer uma análise mais detalhada.

Ainda no que diz respeito ao paralelismo, existe a opção de tornar um algoritmo paralelo de PL em um algoritmo seguro. Tal opção não havia sido considerada anteriormente. Este capítulo investigou a possibilidade de obter-se ganhos em desempenho (*speed-up*) ao tornar-se um algoritmo paralelo de PL em um algoritmo seguro. Ele comparou duas abordagens diferentes: (i) o algoritmo seguro de PL de Toft implementado de maneira paralela; e (ii) um algoritmo paralelo de simplex implementado de maneira segura. Tal comparação requeriu que o protocolo de [93] fosse revisado a fim de evidenciar os custos exatos para cada processador quando se considera o uso de recursos limitados, i.e., cenário real.

Além disto, conduziu-se uma busca por algoritmos paralelos de programação linear a fim de selecionar uma abordagem capaz de resolver problemas de otimização de cadeias logísticas com variáveis secretas, i.e., solução com preservação da privacidade. Foram consideradas diferentes abordagens para algoritmo paralelos de PL. Métodos de Ponto-Interior [96] oferecem *speed-up* considerável quando comparados com implementações sequenciais. A idéia básica é mover-se dentro da região praticável, à procura da solução ótima. Usualmente, a solução do problema converge de maneira bastante rápida.

Implementações paralelas de métodos de ponto-interior incluem [53]. Apesar dos resultados serem bons, a estrutura dos problemas de otimização de cadeias logísticas impede que tais métodos sejam efetivos. A ração é a dependência da estrutura da matriz que representa o problema. Métodos de ponto-interior dependem demasiadamente da densidade da matriz, i.e., quantidade de elementos que possuem valor igual a zero e podem, portanto, ser excluídos das computações. A abordagem de [53] se baseia na identificação de estruturas-bloco na matriz do sistema e, a partir daí, propõe uma implementação paralela orientada a objetos. Visto que os valores da matriz que representa o sistema são criptografados, não é possível se identificar blocos de zeros ou se distinguir entre o que é zero e o resto, o que anula qualquer possibilidade de *speed-up*.

Entre os métodos para resolução de problemas de PL encontra-se o método de *Nelder-Mead* [74]. Em [62], Lee e Wiswall propõem uma generalização deste método, aplicável a procesadores paralelos. A chamada "Parallel Implementation of the Simplex Function Minimization Routine" requer problemas sem restrições. Umas das características de problemas de otimização de CL, mais especificamente PDCL, é a existência de várias restrições, como já descrito no Capítulo 3. De fato é possível reformular-se um problema com restrições em um problema sem restrições, aplicando-se certas transformações. No entanto, os custos de tal transformação, que envolve operações complexas, são razões suficientes para desencorajar a utilização desta abordagem.

O método mais utilizado para solução de problemas de PL é o método Simplex de Dantzig [36]. Ele é baseado na idéia de mover-se entre os cantos do simplex para vértices onde o valor da função objetivo é menor até que um mínimo local seja encontrado, ou até que o problema seja identificado como sem solução (ilimitado). Pela propriedade de convexidade, um mínimo local é também um mínimo global, de modo que a solução ótima foi encontrada. Regras específicas são aplicadas para a seleção dos elementos pivô, as quais determinam quão rápido a solução converge. A literatura contém diversas abordagem paralelas para o simplex. Em [90], um método simplex revisado e paralelo é proposto. Um modelo de paralelismo de colunas é introduzido. Ele funciona aplicando-se uma multiplicação matriz-vetor em paralelo e uma redução mínima repetidamente para cada iteração. Assim como o método de ponto-interior, o *speed-up* do método simplex revisado e paralelo depende da estrutura da matriz. Aliado a isto, este último requer operações complexas com matrizes que são consideradas caras mesmo quando computando de maneira não segura. Em [73], uma abordagem mestre-escravo é proposta. O modelo de paralelismo é em linha, sendo que cada processador trabalha com um sub-tableau independente. Entre as desvantagens relaciona-se a suposição da existência de limites superiores e inferiores para cada uma das restrições a fim de calcular os chamados limiares. Adicionalmente, tal abordagem requer ordenação de vetores, que precisaria ser executada em vetores nos quais os elementos são variáveis criptografadas. Além disto, o *speed-up* é dependente da estrutura da matriz. Por fim, o método simplex paralelo baseado no modelo BSP de [41] oferece custos computacionais reduzidos com uma técnica de paralelismo de linha. Este método não depende da estrutura da matriz nem do tipo de restrição. Ele somente requer que o problema esteja representado na forma canônica, que é a mesma hipótese assumida por Toft e Li e Atallah. O modelo BSP de [94] visa escalabilidade, portabilidade e previsibilidade, fornecendo novas fundações para o desenvolvimento de sistemas computacionais escaláveis e paralelos. Ele pode ser extendido para adequar-se aos requisitos de CSM, como descrito na Seção 6.2.2.1. Além disto, tal abordagem não inclui operações complexas com matrizes e o *overhead* para lidar com variáveis criptografadas é similar ao da solução de Toft.

Até então, nenhum modelo de computação segura multi-parte em paralelo para otimização de cadeias logísticas havia sido proposto. Entre as possibilidades, a abordagem baseada no modelo BSP foi escolhida como a melhor opção. Um modelo para computação paralela e segura, baseado no modelo BSP de [94] e na abordagem de [41] foi então, proposto. Ele é seguro e correto de acordo com as premissas do modelo de segurança semi-honesto [50].

Uma análise detalhada dos protocolos (i) e (ii) revelou que não existe diferença entre tornar um algoritmo paralelo de PL em um algoritmo seguro ou tornar um algoritmo seguro de PL em um algoritmo paralelo. No entanto, o interesse principal deste trabalho é fornecer soluções práticas para problemas de otimização de CL. Como mencionado anteriormente, aplicações reais incluem participantes geograficamente dispersos e condições de rede distintas podem influenciar o desempenho dos protocolos, como um todo. O próximo passo foi implementar a escolha mais simples e avaliá-la em condições de rede especiais, como na presença de atraso. Foram identificados dois métodos de sincronização, a nível de participante e a nível de *thread*. Os resultados experimentais mostraram que as

condições de rede influenciam ambos os métodos de maneira similar. Eles também apontaram para o fato de que o nível de paralelismo, i.e., número de *threads*, é um aspecto decisivo para o desempenho do solução. A escolha de tal número depende de algum indicador de desempenho que eventualmente depende das condições da rede de comunicação. Na prática, tais condições são variáveis e portanto, o número de *threads* para otimalidade também varia.

Finalmente, um algoritmo de agendamento adaptativo foi projetado, implementado e avaliado. Ele é capaz de selecionar dinamicamente o número de *threads* baseado na observação de um indicador de desempenho. O algoritmo se comporta como esperado, buscando por um valor ótimo para o número de *threads* por processador/core. Quanto mais alto for o atraso na rede, maior será o número de *threads*, o que aumenta o nível de utilização dos processadores e consequentemente melhora o desempenho do protocolo. Devido a utilização de operações em paralelo, o efeito do atraso na rede é também reduzido. Aliado a tudo isto, o algoritmo proposto é capaz de lidar com variações nas condições da rede como demonstrado na Seção 6.4.2.

Esta solução é um passo importante em direção a prover protocolo seguros e paralelos de programação linear para problemas de otimização de cadeias logísticas.

É importante ressaltar que os resultados experimentais apresentados dependem da implementação realizada. Como não foram apresentados métodos formais para validação da implementação não é possível garantir, por meio de prova formal, a veracidade dos resultados experimentais apresentados. Este fato deve ser levado em conta na análise e argumentação mas de modo algum invalida os resultados nem mesmo diminui a importância dos trabalhos realizados.

6.6 Conclusão

Programação linear segura permite aos parceiros de negócios interagir em atividades de cadeias logísticas *cross*-organizações sem que haja o perigo do vazamento de informações, mas a um alto custo computacional e de comunicação. Computação paralela oferece os meios para reduzir tais custos por maio da execução, em paralelo, de operações seguras, melhorando o desempenho das soluções.

Este capítulo teve ser foco em abordagens práticas de protocolos paralelos com preservação da privacidade para programação linear colaborativa utilizando-se variáveis criptografadas. Inicialmente foram comparadas, de maneira estritamente teórica, as abordagens de tornar um algoritmo seguro em um algoritmo paralelo e de tornar um algoritmo paralelo em um algoritmo seguro. Visto que as complexidades são similares, a escolha mais simples foi implementada: tornar um algoritmo seguro em um algoritmo paralelo. Dois métodos de sincronização foram identificados, implementados e comparados sob diferentes condições de rede. O desempenho é quase independente das condições de rede, mas o número de *threads*, ou seja, nível de paralelismo, precisa ser adaptado a tais condições a fim de obter-se resultados otimizados.

Sabendo-se que condições de rede como atraso não podem ser sempre determinados de antemão e mais do que isso, podem mudar durante a execução do protocolo, um algoritmo para agendamento adaptativo de *threads* foi apresentado e avaliado. Tal algoritmo realiza a seleção dinâmica do número de *threads* por processador a fim de obter um *speed-up* ótimo. Foi mostrado também, que o algoritmo proposto é capaz de adaptar-se até mesmo a condições variáveis de rede e executar próximo do valor ótimo.

Capítulo 7

Conclusão

7.1 Visão Geral do Trabalho

Esta dissertação argumenta sobre a existência de uma crescente tendência em direção ao provimento de segurança contra parceiros de negócios. Até então, somente computação segura multi-parte tem sido capaz de fornecer garantias de segurança.

Considerou-se, como caso de uso, o problema de computar o plano diretor da cadeia logística no qual diversos parceiros de negócio desejam, de maneira colaborativa, planejar produção, armazenamento e transporte. Em tal cenário, é imperativo que as informações trocadas sejam protegidas, de modo que não sejam reveladas sob hipótese alguma. CSM é capaz de fornecer garantias de segurança.

Programas lineares são capazes de modelar diversos problemas reais e esquemas com preservação da privacidade podem ser utilizados para assegurar os requisitos de negócio: plano diretor globalmente ótimo e garantia de que as informações a respeito do sistema serão mantidas privadas. A chamada programação linear segura permite a otimização de cadeias logísticas *cross*-organização, mas possui desempenho ruim.

Os trabalhos apresentados aqui focaram na praticalidade de protocolos seguros multi-parte para problemas de programação linear. A primeira parte, Capítulos 4 e 5, considerou protocolos simplex seguros nos quais o índice do elemento pivô é selecionado em claro. Foram identificadas duas deficiências no atual estado-da-arte e soluções foram propostas e avaliadas. Protocolos multi-parte seguros de permutação com complexidade de rodada reduzida assim como técnicas probabilísticas para evitar permutações são capazes de melhorar o desempenho das soluções, na prática.

A segunda parte deste trabalho considerou a otimização de algoritmos de programação linear com preservação de privacidade nos quais os elementos da matriz que representa o sistema são mantidos na forma criptografada. Devido ao crescimento do poder computacional, a computação paralela cria a oportunidade de melhorar o desempenho dos protocolos por meio da execução de operações seguras em paralelo. Foram consideradas técnicas de computação paralela aplicadas às melhores soluções conhecidas e um modelo de computação paralela segura multi-parte para problemas de otimização de cadeias logísticas foi proposto. Identificou-se que, para o desempenho ótimo, o número de *threads* deve ser escolhido como função das condições da rede. Visto que tais condições, como o atraso, não podem ser determinadas *a priori* e podem mudar durante a execução do protocolo, foi proposto um algoritmo de agendamento adaptativo que seleciona dinamicamente o número de *threads* a ser utilizado. Demonstrou-se que tal algoritmo é capaz de adaptar-se a variações nas condições da rede e executar próximo da otimalidade.

7.2 Revisão dos Objetivos

Esta seção revisa os objetivos descritos na Seção 1.2 e os relaciona de acordo com as contribuições realizadas.

1. **Propor técnicas para redução do número de permutações seguras multi-parte**.

   O Capítulo 4 foi inteiramente dedicado ao estudo, projeto e avaliação de um esquema, baseado na probabilidade de uma moeda, para a redução do número de permutações necessárias durante

a execução de protocolos de PL com preservação de privacidade. A moeda é lançada antes de cada iteração com uma dada probabilidade e o resultado decide se o protocolo deverá ou não permutar a matriz do sistema. Se a probabilidade da moeda for estimada de maneira que nenhum participante possa distinguir a razão pela qual se está permutando, i.e., se por repetição de um índice de uma linha ou de uma coluna ou se devido à decisão da moeda, nenhuma informação adicional pode ser inferida a respeito do número real de índices de elementos pivô sendo repetidos.

O estado-da-arte para este tipo de solução requer a invocação de protocolos seguros de permutação em todas as iterações do algoritmo simplex. Os resultados mostraram que não é necessário permutar a matriz do sistema em todas as iterações, mas devido à introdução do nosso esquema melhora-se o desempenho do protocolo, ao custo de revelar-se uma pequena quantidade de informação. Em última instância, o esquema probabilístico é capaz de controlar a relação entre segurança e desempenho de um protocolo seguro de PL.

2. **Projetar protocolos práticos, seguros e multi-parte de permutação para algoritmos de PL com preservação da privacidade nos quais o índice do elemento pivô é selecionado em claro**.

   A Seção 5.3 introduziu um protocolo de permutação segura multi-parte com complexidade linear de rodada para PL seguro no qual o índice do elemento pivô é mantido em claro. O protocolo pode ser considerado como uma extensão do protocolo seguro de embaralhamento e permutação para cenários duas partes de Li e Atallah [68]. Os participantes embaralham e permutam a matriz do sistema, um depois do outro, primeiro linhas depois colunas, e enviam a matriz permutada para o próximo participante. O último participante realiza um *multicast* da matriz permutada final para todos os participantes. Assim, garante-se que nenhum participante obtenha informações a respeito das permutações individuais (de outros participantes) e das posições iniciais dos elementos da matriz do sistema, ou seja, a privacidade é garantida. Ao fim, é necessário que se execute um protocolo de recuperação de índices, que, por ser dependente do protocolo de permutação, também possui complexidade linear de rodada.

   A solução proposta foi avaliada e os custos revelaram-se lineares no número de participantes e no termo de comunicação. Por isso, o desempenho da solução é drasticamente afetado quando se consideram problemas reais com diversos participantes e um eventual atraso na comunicação. Sendo assim, soluções melhoradas se fazem necessárias.

3. **Optimizar soluções de permutação segura multi-parte por meio do uso de protocolos com complexidade de rodada reduzida**.

   As interações entre os participantes são um recurso chave para computações seguras e distribuídas. A complexidade de rodada é definida em função destas interações e é uma das maiores barreiras quando se procuram protocolos seguros que sejam práticos [6]. Ela é diretamente afetada pelas condições da rede, como atraso, que é um elemento bastante comum em cenários de CL, nos quais os participantes se encontram geograficamente dispersos. Em resposta à fragilidade da solução apresentada na Seção 5.3 em relação às condições da rede, a Seção 5.4 introduziu um protocolo multi-parte seguro de permutação com complexidade reduzida de rodada.

   As permutações são representadas como matrizes e as combinações das permutações individuais de todos os participantes são calculadas separadamente, como uma única matriz para linhas e uma para colunas. Feito isso, a matriz do sistema é multiplicada por tais combinações. Uma estrutura de árvore-binária é utilizada para a multiplicação das matrizes, o que resulta em uma solução com complexidade logarítmica de rodada. Tal solução tira proveito do fato de que diversas sub-operações, como multiplicações seguras, são independentes e podem, portanto, ser executadas em paralelo.

   Existe uma relação custo-benefício entre complexidade de rodada, número de participantes, condições da rede e poder de paralelismo. Com o poder de computação atual, a solução linear eventualmente trará resultados melhores, mas a previsão é de que a solução logarítmica traga melhores resultados em um futuro próximo. Cadeias logísticas estão se tornando cada vez maiores devido ao mercado globalizado, não sendo rara a inclusão de centenas de participantes geograficamente dispersos. Aliado a isso, observa-se o crescimento massivo do poder computacional,

que fornece os meios para paralelismo em larga escala. A tendência está do lado da solução logarítmica, ou seja, com complexidade reduzida de rodada, de modo que os resultados serão mais expressivos no futuro, evidenciando ainda mais a qualidade da solução proposta.

4. **Projetar um modelo de computação segura multi-parte para máquinas paralelas capaz de cumprir os requisitos dos problemas de otimização de CL**.

   O poder computacional tem crescido vertiginosamente com o passar dos anos. Este crescimento tem permitido que problemas bastante complexos sejam resolvidos de maneira mais eficiente, especialmente utilizando-se técnicas de computação paralela. Eficiência é um aspecto chave para protocolos de computação segura multi-parte.

   A Seção 6.2.2.1 introduziu um modelo de computação segura multi-parte para máquinas paralelas. Tal modelo é baseado no modelo BSP proposto por [94], que fornece características de escalabilidade, portabilidade e previsibilidade, ao passo que encara máquinas paralelas como um conjunto de processadores (capazes de executar operações em paralelo), uma rede de comunicação entre tais processadores que entrega mensagens de maneira ponto-a-ponto e um mecanismo de sincronização. Técnicas de CSM são introduzidas a fim de assegurar a privacidade das entradas durante as computações, cumprindo os requisitos de segurança exigidos pelos parceiros de negócios. O modelo proposto na Seção 6.2.2.1 fornece a base para o algoritmo simplex seguro e paralelo baseado no modelo BSP, proposto na Seção 6.2.2.

5. **Melhorar o desempenho prático de algoritmos de PL com preservação da privacidade nos quais o índice do elemento pivô é mantido como uma variável secreta, mediante o uso de técnicas de paralelismo**.

   O Capítulo 6 explora a alternativa de implementar-se protocolos de CSM para problemas de PDCL em máquinas paralelas. Duas possíveis soluções para melhora do desempenho utilizando-se paralelismo são comparadas. A análise de complexidade dos resultados revelou que ambas as abordagens possuem complexidades de computação e comunicação similares.

   A solução mais simples foi implementada e dois diferentes modelos de sincronização foram identificados: sincronizar cada *thread* com sua *thread* remota equivalente (sincronização a nível de *thread*), ou sincronizar cada *thread* localmente e depois sincronizar os participantes (sincronização a nível de participantes). A avaliação dos resultados suporta a hipótese de que o primeiro método é mais rápido. Adicionalmente, concluiu-se que o desempenho ótimo de métodos de simplex paralelos depende da relação entre o número de *threads* instanciadas pelo sistema operacional e das condições de rede.

   Condições de rede, como atraso, dependem de diversos aspectos que não podem ser controlados ou preditos por um algoritmo distribuído. Sendo assim, apresentou-se e avaliou-se um algoritmo de agendamento adaptativo para a seleção dinâmica do número de *threads*, tornando desnecessário determinar estaticamente e *a priori* tal número para se obter um *speed-up* ótimo. Os resultados dos testes revelaram que o algoritmo é também capaz de lidar com condições variáveis da rede, melhorando a performance de protocolos simplex seguros em paralelo.

7.3 Contribuições e Resultados da Dissertação

Este trabalho apresentou várias contribuições não somente na área de algoritmos seguros multi-parte, mas também no que diz respeito a técnicas de redução de custos (complexidade) e melhora de performance de soluções (estado-da-arte) de programação linear com preservação da privacidade. Tais contribuições são listadas a seguir:

- Introdução de um esquema baseado na probabilidade de uma moeda para redução do número de permutações seguras em soluções de PL com preservação da privacidade nas quais o índice do elemento pivô é selecionado em claro. Este trabalho foi o primeiro a considerar tal classe de problemas na área de otimização de cadeias logísticas e os resultados mostraram melhorias consideráveis.

- Protocolos de permutação segura multi-parte para problemas de otimização de cadeias logísticas baseados em PL seguros ainda não haviam sido considerados formalmente pela literatura. Este trabalho propôs uma extensão, para o caso multi-parte, de um protocolo conhecido e um algoritmo com complexidade reduzida de rodada a fim de minimizar os efeitos das condições de rede, como o atraso. Os resultados revelaram que existe uma relação de compromisso entre complexidade de rodada, número de participantes, poder computacional e condições de rede, de forma que não existe uma solução absoluta para otimalidade, mas cada problema precisa ser considerado separadamente. Além disto, a solução proposta foi avaliada na teoria e na prática e uma boa estimativa de como tais parâmetros podem ser utilizados para se atingir resultados melhorados foi apresentada.

- Por meio de uma comparação extensiva entre algoritmos paralelos seguros concluiu-se que não existe melhora significativa quando se propõe uma versão segura de um algoritmo paralelo. Otimização de desempenho é investigada considerando-se o número de *threads*, o atraso na rede e o agendamento adaptativo. Introduziu-se e avaliou-se um algoritmo capaz de procurar, de maneira dinâmica, pelo número ótimo de *threads*, identificando um possível atraso na rede mediante a observação do *throughput* ou outro indicador de performance, e de agir adaptando a escolha de tal número a fim de minimizar a influência do atraso. Tal algoritmo é ainda capaz de adaptar-se a condições variáveis de rede, reduzindo os efeitos do atraso e levando os resultados para próximos dos resultados ótimos.

A maior parte das contribuições listadas acima foi aceita para publicação.

[1] KERSCHBAUM, F.; DEITOS, R. **Security Against the Business Partner**. In *2008 ACM Workshop on Secure Web Services (SWS)*, 2008.

[2] DEITOS, R.; KERSCHBAUM, F. **Parallelizing Secure Linear Programming**. In *Concurrency and Computation: Practice and Experience Special Issue on Multi-core Supported Network and System Security (CCPE)*, 2009.

[3] DEITOS, R.; KERSCHBAUM, F. **Improving Practical Performance on Secure and Private Collaborative Linear Programming**. In *DEXA 2009 – 1st International Workshop on Business Processes Security (BPS)*, 2009.

7.4 Perspectivas Futuras

A melhoria do desempenho de protocolos de computação segura multi-parte é um campo de estudo em andamento. As contribuições desta dissertação abrem um amplo leque de possíveis direções para trabalhos futuros. Uma possibilidade diz respeito à extensão da técnica probabilística proposta no Capítulo 4 considerando a posição em que os índices repetem. Além disto, um algoritmo com complexidade constante de rodada para permutação segura multi-parte seria uma extensão natural dos trabalhos desenvolvidos.

No que diz respeito a protocolos seguros e paralelos de programação linear, podem-se citar duas direções distintas a serem exploradas. A largura de banda é um parâmetro para desempenho de rede que não foi considerado nos trabalhos. O impacto da largura de banda limitada poderia ser explorado em trabalhos futuros. Lista-se, ainda, a questão do atraso variável na rede. Existe uma discussão dentro da comunidade científica a respeito de que a utilização de um gerador de tráfego resultaria em resultados mais realísticos. Naturalmente, os resultados apresentados nesta dissertação são capazes de capturar uma variação no atraso limitada pela variação introduzida pelo próprio *script*. Na lista de trabalhos futuros se inclui a exploração do desempenho do algoritmo proposto sob condições reais de rede. Adicionalmente, poderia se adicionar um esquema de recuperação de erros a fim de evitar perdas no sistema.

Finalmente, cita-se a extensão dos protocolos para suporte ao modelo de segurança malicioso com adversários adaptativos.

Referências Bibliográficas

[1] P. Ahuja and J. Orlin. Inverse optimization. *Operations Research*, 49(5):771–783, 2001.

[2] K. Anand and M. Goyal. Incentives for information acquisition and information dissemination in a supply-chain. 2005.

[3] Mikhail Atallah, Marina Bykova, Jiangtao Li, Keith Frikken, and Mercan Topkara. Private collaborative forecasting and benchmarking. In *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 103–114, New York, NY, USA, 2004. ACM. ISBN 1581139683.

[4] W. Baker, D. Hylender, and A. Valentine. 2008 data breach investigations report. Technical report, 2008.

[5] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *PODC '89: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 201–209, New York, NY, USA, 1989. ACM. ISBN 0897913264.

[6] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, New York, NY, USA, 1990. ACM. ISBN 0-89791-361-2.

[7] Donald Beaver. Minimal-latency secure function evaluation. In *EUROCRYPT 2000*, pages 335–350. Springer, 2000.

[8] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM Press. ISBN 0897912640.

[9] Peter Bogetoft, Dan L. Christensen, Ivan Damgard, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus D. Nielsen, Jesper B. Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, Feb 2008.

[10] Miklos Bona. *Combinatorics of Permutations*. Chapman Hall-CRC, 2004.

[11] Dan Boneh and Matthew Franklin. Ecient generation of shared RSA keys. In *In Advances in Cryptology/CRYPTO 97*, volume 1294, pages 425–439, 1997.

[12] E. Brickell and Y. Yacobi. On privacy homomorphisms. In *Advances in Cryptology (EUROCRYPT'87)*, volume 304, pages 117–126, New York, USA, 1987. Springer.

[13] G. Cachon. Supply chain coordination with contracts. *Handbooks in Operations Research and Management Science: Supply Chain Management*, 2003.

[14] G. Cachon and M. Fisher. Supply chain inventory management and the value of shared information. *Management Science*, 46(8):1032–1048, 2000.

[15] G. Cachon and M. Lariviere. Contracting to assure supply: how to share demand forecasts in a supply chain. *Management Science*, 47(5):629–646, 2001.

[16] G. Cachon and P. Zipkin. Competitive and cooperative inventory policies in a two stage supply chain. *Management Science*, 45(7):936–953, 1999.

[17] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 21–30, New York, NY, USA, 2002. ACM. ISBN 1-58113-612-9.

[18] R. Canetti, U. Friege, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. Technical report, Cambridge, MA, USA, 1996.

[19] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, New York, NY, USA, 1988. ACM Press. ISBN 0897912640.

[20] F. Chen. Information sharing and supply chain coordination. *Handbook of Operations Research and Management Science: Supply Chain Management.*

[21] F. Chen. Echelon reorder points, installation reorder points, and the value of centralized demand information. *Management Science*, 44(12):221–234, 1998.

[22] F. Chen. Auctioning supply contracts. 2001.

[23] F. Chen and B. Yu. Quantifying the value of leadtime information in a single-location inventory system. 2001.

[24] Vasek Chvatal. *Linear Programming.* W. H. Freeman (September 15, 1983), 1983.

[25] Andrew J. Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Manage. Sci.*, 50(12 Supplement):1782–1790, 2004. ISSN 0025-1909.

[26] C. Corbett. Stochastic inventory systems in a supply chain with asymmetric information: Cycle stocks, safety stocks, and consignment stock. *Operations Research*, 49(4):487–500, 2001.

[27] Ronald Cramer and Ivan Damgard. Multiparty computation, an introduction. 2004.

[28] Ronald Cramer, Ivan B. Damgard, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations with dishonest minority. In *In Eurocrypt '99*, volume 1561, pages 311–326, 1999.

[29] Ronald Cramer, Ivan Damgard, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 280–299, London, UK, 2001. Springer-Verlag. ISBN 3-540-42070-3.

[30] I. Damgard, M. Jurik, and J. Nielsen. A generalization of paillier's public-key system with applications to electronic voting, 2003.

[31] Ivan Damgard and Ronald Cramer. Zero-knowledge proof for finite field arithmetic, or: Can zero-knowledge be for free. In *In Proc. CRYPTO*, volume 1462, pages 424–441, 1998.

[32] Ivan Damgard, Matthias Fitzi, Eike Kiltz, Jesper Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. pages 285–304. 2006.

[33] Ivan Damgard, Martin Geisler, and Mikkel Kroigard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1:22–31, 2008.

[34] Ivan B. Damgard and Mads J. Jurik. A length-flexible threshold cryptosystem with applications. In *In proceedings of ACISP'03, LNCS series*, volume 2727, pages 350–364, 2003.

[35] G. B. Dantzig. *Linear Programming and Extensions.* 1963.

[36] George B. Dantzig and Mukund N. Thapa. *Linear Programming 1: Introduction.* Springer-Verlag New York, LLC, 1997.

[37] Rafael Deitos, Florian Kerschbaum, and Philip Robinson. A comprehensive security architecture for dynamic, web service based virtual organizations for businesses. In *SWS '06: Proceedings of the 3rd ACM workshop on Secure web services*, pages 103–104, New York, NY, USA, 2006. ACM. ISBN 1-59593-546-0.

[38] V. Deshpande and L. Schwarz. Optimal capacity choice and allocation in decentralized supply chains. 2005.

[39] Vinayak Deshpande, Mikhail Atallah, Marina Blanton, Keith Frikken, Jiangtao Li, and Leroy Schwarz. Secure collaborative planning, forecasting, and replenishment. Technical report, 2006.

[40] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, San Diego, CA, USA, August 2004.

[41] Wang Dong, Hu Baosheng, Peng Qinke, and Tan Yudong. Bsp-based parallel simplex method. In *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, volume 2, pages 635–639 vol.2, 2000.

[42] Wenliang Du and Zhijun Zhan. A practical approach to solve secure multi-party computation problems. In *NSPW '02: Proceedings of the 2002 workshop on New security paradigms*, pages 127–135, New York, NY, USA, 2002. ACM Press. ISBN 158113598X.

[43] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *Eur. J. Inf. Syst.*, 2007(1):1–15, January 2007.

[44] Pierre A. Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, pages 90–104, London, UK, 2000. Springer-Verlag. ISBN 3-540-42700-7.

[45] Keith B. Frikken and Mikhail J. Atallah. Privacy preserving route planning. In *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 8–15, New York, NY, USA, 2004. ACM. ISBN 1581139683.

[46] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *Lecture Notes in Computer Science*, pages 330–342. Springer-Verlag, 2007.

[47] V. Gaur, A. Giloni, and S. Seshadri. Information sharing in a supply chain under arma demand. *Management Science*, 51:961–969, 2005.

[48] S. Gavirneni, R. Kapuscinski, and S. Tayur. Value of information in capacitate supply chains. *Management Science*, 45(1):16–24, 1999.

[49] Niv Gilboa. Two party RSA key generation. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 116–129, London, UK, 1999. Springer-Verlag. ISBN 3-540-66347-9.

[50] O. Goldreich. Secure multi-party computation. Technical report, Weizmann Institute of Science, 2002.

[51] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM. ISBN 0897912217.

[52] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[53] Jacek Gondzio and Robert Sarkissian. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, June 2003.

[54] S. Graves. A single item inventory model for a nonstationary demand process. *Manufacturing & Service Operations Management*, 1(1):50–61, 1999.

[55] Jens Groth. Cryptography in subgroups of z_n^*. pages 50–65. 2005.

[56] Martin Hirt and Jesper Nielsen. Robust multiparty computation with linear communication complexity. pages 463–482. 2006.

[57] Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography '2001, volume 1992 of Lecture Notes in Computer Science*, pages 119–136, 2001.

[58] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, New York, NY, USA, 1984. ACM. ISBN 0897911334.

[59] F. Kerschbaum, J. Haller, and Y. Karabulut. Pathtrust : A trust-based reputation service for virtual organization formation. 2006.

[60] F. Kerschbaum, D. Dahlmeier, A. Schröpfer, and D. Biswas. On the practical importance of commmunication complexity for secure multi-party computation protocols. *ACM Symposium on Applied Computing*, 2009.

[61] Florian Kerschbaum and Rafael J. Deitos. Security against the business partner. In *SWS '08: Proceedings of the 2008 ACM workshop on Secure web services*, pages 1–10, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-292-4.

[62] Donghoon Lee and Matthew Wiswall. A parallel implementation of the simplex function minimization routine. *Comput. Econ.*, 30(2):171–187, September 2007. ISSN 0927-7099.

[63] H. Lee and S. Whang. Decentralized multi-echelon supply chains: Incentives and information. *Management Science*, 45(5):633–640, 1999.

[64] H. Lee, P. Padmanabhan, and S. Whang. The bullwhip effect in supply chains. *Sloan Management Review*, 38:93–102, 1997.

[65] H. Lee, V. Padmanabhan, and S. Whang. Information distortion in a supply chain: The bullwhip effect. *Management Science*, 43(4):546–558, 1997.

[66] H. Lee, K. So, and C. Tang. The value of information sharing in a two-level supply chain. *Management Science*, 45(5):626–643, 2000.

[67] Hau L. Lee and Seungjin Whang. Information sharing in a supply chain. *International Journal of Manufacturing Technology and Management*, 1:79–93, 2000.

[68] Jiangtao Li and M. J. Atallah. Secure and private collaborative linear programming. *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2006.

[69] L. Li. Information sharing in a supply chain with horizontal competition. *Management Science*, 48(9):1196–1212, 2003.

[70] Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test, 2003.

[71] Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154 (2):370–381, February 2006.

[72] B. Mishra, S. Raghunathan, and X. Yue. Information sharing in supply chains: Incentives for information distortion. 2005.

[73] N. R. Natraj, G. L. Thompson, and F. Harche. Solving linear programs using distributed parallel computing. Technical report, Carnegie Mellon University, 1994.

[74] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[75] Jesper B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *In Advances in Cryptology - CRYPTO'03*, volume 2729, pages 247–264, 2003.

[76] Manfred Padberg. *Linear Optimization and Extensions*. 1999.

[77] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Eurocrypt*, 1592:223–238, 1999.

[78] R. Pibernik and E. Sucky. Centralised and decentralised supply chain planning. *International Journal of Integrated Supply Management*, 2(1/2):6–27, 2006.

[79] Richard Pibernik and Eric Sucky. An approach to inter-domain master planning in supply chains. *International Journal of Production Economics*, 108(1-2):200–212, July 2007.

[80] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, New York, NY, USA, 1989. ACM Press. ISBN 0897913078.

[81] S. Raghunathan. Information sharing in a supply chain: A note on its value when demand is nonstationary. *Management Science*, 47(4):605–610, 2001.

[82] D. Rappe. *Homomorphic cryptosystems and their applications*. PhD thesis, University of Dortmund, Dortmund, Germany, 2004.

[83] Ravi, Edward, Hal, and Charles. Role-based access control models, 1996.

[84] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. pages 169–177. Academic Press, 1978.

[85] Luigi Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27:31–41, 1997.

[86] Gabriel Rosenberg. Enumeration of all extreme equilibria of bimatrix games with integer pivoting and improved degeneracy check. Technical report, http://www.cdam.lse.ac.uk/Reports/Abstracts/cdam-2005-18.html, 2005.

[87] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, pages 44–60, London, UK, 1998. Springer-Verlag. ISBN 3-540-64792-9.

[88] Bruce Schneier. *Applied Cryptography*. Wiley & Sons, 2nd edition, October 1996. ISBN 0471117099.

[89] Victor Shoup. Practical threshold signatures. In *EUROCRYPT 2000*, volume 1807, pages 207–220, 2000.

[90] Wei Shu and Min-You Wu. Sparse implementation of revised simplex algorithms on parallel computers. In *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, March 1993.

[91] Marius-Călin Silaghi. Zero-knowledge proofs for mix-nets of secret shares and a version of elgamal with modular homomorphism, 2005.

[92] A. Tarantola. *Inverse Problem Theory: Methods for Data Fitting and Model Parameter Estimation*. Elsevier, Amsterdam, 1987.

[93] Tomas Toft. *Primitives and Applications for Multi-Party Computation.* PhD thesis, University of Aarhus, 2007.

[94] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990. ISSN 0001-0782.

[95] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions.* 2001.

[96] S. J. Wright. *Primal-Dual Interior-Point Methods.* 1997.

[97] Andrew C. Yao. Protocols for secure computations. *Proceedings of the 23rd Annual IEEE Symposium on Foundations*, 1982.

[98] Gavriel Yarmish. *A Distributed Implementation of the Simplex Method.* PhD thesis, Polytechnic University, 2001.