Design of large scale applications of secure multiparty computation : secure linear programming

Design of large scale applications of secure multiparty computation : secure linear programming

de Hoogh, S.J.A.

DOI:
10.6100/IR735328

Published: 01/01/2012

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming / door Sebastiaan de Hoogh. – Eindhoven: Technische Universiteit Eindhoven, 2012.
Proefschrift. – ISBN 978-90-386-3203-2
NUR 919
Subject headings: Cryptology, multiparty computation, cryptographic protocols, linear programming, algorithms.

Printed by Printservice TU/e.
Cover: Het Paleis. Design by Brigitte Gedike

Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 30 augustus 2012 om 16.00 uur

door

Sebastiaan Jacobus Antonius de Hoogh

geboren te Dongen

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. H.C.A. van Tilborg

Copromotor:
dr.ir. L.A.M. Schoenmakers

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-3203-2

Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming

Summary

Secure multiparty computation is a basic concept of growing interest in modern cryptography. It allows a set of mutually distrusting parties to perform a computation on their private information in such a way that as little as possible is revealed about each private input. The early results of multiparty computation have only theoretical significance since they are not able to solve computationally complex functions in a reasonable amount of time. Nowadays, efficiency of secure multiparty computation is an important topic of cryptographic research.

As a case study we apply multiparty computation to solve the problem of secure linear programming. The results enable, for example in the context of the EU-FP7 project SecureSCM, collaborative supply chain management. Collaborative supply chain management is about the optimization of the supply and demand configuration of a supply chain. In order to optimize the total benefit of the entire chain, parties should collaborate by pooling their sensitive data.

With the focus on efficiency we design protocols that securely solve any linear program using the simplex algorithm. The simplex algorithm is well studied and there are many variants of the simplex algorithm providing a simple and efficient solution to solving linear programs in practice. However, the cryptographic layer on top of any variant of the simplex algorithm imposes restrictions and new complexity measures. For example, hiding the number of iterations of the simplex algorithm has the consequence that the secure implementations have a worst case number of iterations. Then, since the simplex algorithm has exponentially many iterations in the worst case, the secure implementations have exponentially many iterations in all cases.

To give a basis for understanding the restrictions, we review the basic theory behind the simplex algorithm and we provide a set of cryptographic building blocks used to implement secure protocols evaluating basic variants of the simplex algorithm. We show how to balance between privacy and efficiency; some protocols reveal data about the internal state of the simplex algorithm, such as the number of iterations, in order to improve the expected running times.

For the sake of simplicity and efficiency, the protocols are based on Shamir's secret sharing scheme. We combine and use the results from the literature on secure random number generation, secure circuit evaluation, secure comparison, and secret indexing to construct efficient building blocks for secure simplex.

The solutions for secure linear programming in this thesis can be split into two categories. On the one hand, some protocols evaluate the classical variants of the simplex algorithm in which numbers are truncated, while the other protocols evaluate the variants of the simplex algorithms in which truncation is avoided. On the other hand, the protocols can be separated by the size of the tableaus. Theoretically there is no clear winner that has both the best security properties and the best performance.

The rounding errors due to truncations may cause the simplex algorithm to become unstable leading to incorrect results. To securely determine correctness of the output of the protocols we show how to extract a certificate from the output and how to securely prove correctness of the result given this certificate. The protocols for extracting and verifying the certificates are, compared to protocols evaluating the simplex algorithm, very efficient.

We extend and generalize the ideas of using certificates to build efficient universally verifiable protocols, i.e., protocols allowing anyone to check correctness of the output without revealing the private results. Examples of universally verifiable protocols are, typically, relatively expensive protocols in which every transmitted message is accompanied by a noninteractive zero-knowledge proof. If the protocol evaluates a function of which a certificate of correctness can be efficiently extracted, then the protocol will be universally verifiable if the validation of the certificate is universally verifiable.

In addition to the case study on secure linear programming we discuss proofs of restricted shuffles. The scenario is as follows: given a list of encrypted data, some party permutes the list in such a way that nobody learns the permutation applied to the list. Then, the party proves in zero-knowledge that the resulting list consists of encryptions of the same list where the entries are permuted. The party proves in addition that the applied permutation satisfies some restrictions. We apply a result from the literature that provides zero-knowledge protocols for any permutation group represented as an automorphism group of some graph. In order to instantiate the protocols we provide hypergraphs of which the automorphism group is represented by either a rotation, an affine transformation, or a Möbius transformation.

Contents

Chapter 1

Introduction

Secure multiparty computation is a basic concept of growing interest in modern cryptography. It allows a set of mutually distrusting parties to perform a computation on their private information in such a way that as little as possible is revealed about each private input. Such interactive computations are typically described by *protocols* which are step-by-step descriptions of each action to be performed by each party. An *adversary* controlling some of the parties is neither able to prevent the correct result to be computed nor to gain additional information, even by making parties deviate from the protocol. In the 1980s it was proven that secure multiparty computation is feasible for any computable function [Yao82, Yao86, GMW87, BGW88].

Secure multiparty computation involves many different aspects. This makes defining security and designing protocols complicated. For example, a protocol can be designed to be secure against adversaries of some special type. The parties controlled by an adversary are usually called *corrupt* parties. An adversary is *passive* or *semi-honest* if the corrupt parties follow the protocol specification, but try to learn as much as possible from the data collected. An *active* adversary makes the corrupt parties to behave arbitrarily. In addition to those two properties, an adversary is *static* if the set of parties it is going to corrupt is decided at the start of the protocol and is fixed during the protocol execution. The adversary is *adaptive* if the parties are corrupted during protocol execution.

There are many ways of formally defining security of protocols [MR91, Bea91, Gol02]. Commonly, security against some adversary means that all information gained by the adversary can be *simulated* efficiently using only information that the adversary is allowed to know. Simulation usually means that an adversary communicates with some *simulator* that computes messages on behalf of the parties that are not corrupt. In other words, the protocol is said to be secure if the messages computed by the simulator are such that the adversary cannot distinguish them from messages it would have received from the parties that are not corrupt during protocol execution.

The definitions of security also depend on the environment in which a protocol is executed. A protocol can be secure in the *stand-alone* setting, where composition and interaction with other protocols is not considered. More general environments are considered in [Can00]. Ultimately, the environment allows any type of composition [Can01, PW01], which is also known as *universal composability*.

Another important aspect of secure multiparty computation is the model of communication, that describes how messages are transmitted among parties and what an adversary is allowed to observe. For example, in the *cryptographic* setting [Gol02] the parties communicate via a *broadcast* channel, where the adversary sees all communicated messages. Typically, security in this setting is based on a complexity assumption restricting the computing power of the adversary to be polynomial time. Examples of protocols in the

cryptographic setting are [FH96, JJ00, CDN01, DN03]. On the other hand, in the *information theoretic* setting, parties are connected via private channels allowing parties to send each other messages that are not visible to the adversary, if both parties are not corrupt. Security in this setting is *unconditional*, meaning that no restriction with respect to the computing power of the adversary is assumed. Examples of protocols in the information theoretic setting are [BGW88, CCD88].

The early protocols have only theoretical significance since they are not able to solve complex functions in a reasonable amount of time. However, conceptually they are able to solve practical problems such as electronic voting, secure data mining and secure collaborative supply chain management. Nowadays, efficiency of secure multiparty computation is an important topic of cryptographic research. Next to electronic voting schemes, in [BCD+09], a real life application of secure multiparty computation is presented, where some 1200 Danish farmers participated in a secure multiparty computation protocol to determine the market price of their sugarbeets. The protocol took roughly 30 minutes to complete.

This thesis is mostly about efficient multiparty computation. As a case study, we address the problem of secure linear programming to solve the problem of secure collaborative supply chain management by secure multiparty computation. We focus on *communication* complexity, i.e., the total number of communicated bits by each party, and *round* complexity. The latter counts the total number of interactive rounds, where in each successive round parties are sending messages that are dependent on received messages in earlier rounds.

In addition to efficiency, we address the issue in secure computation that the protocols guarantee nothing if none of the parties is honest. This is unacceptable in, for example, voting schemes and cloud computing.

In voting schemes voters send an encrypted vote to a group of talliers. The talliers engage in a protocol to compute the election result. It is unacceptable if nothing can be guaranteed if all talliers are corrupt. The notion of universal verifiability ensures that even if all talliers are corrupt, correctness of the election result can be verified.

In cloud computing, a group of computationally weak parties wish to evaluate some function on their private inputs. Instead of participating in a multiparty protocol, the parties provide encryptions of their private inputs to computationally strong servers that perform the computation. In this setting it is required that the validity of the computed results can always be checked.

In this thesis, we will show how to design protocols that guarantee a correct result, even if all parties are corrupt. This property will be called *universal verifiability* using the same terminology as commonly used in voting schemes. Precisely, we show how universally verifiability can be defined to ensure that correctness of the result of a protocol can always be verified by anyone. We show how to achieve universal verifiable multiparty computation from the protocols of [CDN01].

In addition, we show how universal verifiability can be efficiently achieved if there exists a certificate of correctness for the result of the function that can be validated at relative low computational costs. For example, we will show that computing an optimal solution to a linear program is a computationally expensive task, while validating whether the solution is indeed optimal to the given linear program is a computationally cheap task. We show that if a protocol that computes a solution to a linear program is extended by a universally verifiable protocol that validates whether the solution is indeed optimal, then

the whole protocol will be universally verifiable.

Secure Linear Programming

Optimization is essential in every day life, science, and industry. When traveling, for example, one wants to know the shortest, fastest, or most scenic route. Many physical systems converge to a state of minimal energy. Hence, optimization is required to study those systems. In industry, companies optimize their production numbers and selling prices for the best revenue; and construction processes are scheduled to optimize the throughput.

Optimization involves a broad collection of methods and techniques to solve all sorts of optimization problems. An optimization problem contains a function called the *objective*. The goal of optimization is to assign values to the variables such that the objective is optimized. These values may be constrained. In this case the optimization problem is called a *constrained optimization problem*. If the objective and the constraints can be written as linear functions we speak of *linear optimization* or *linear programming*, abbreviated as LP.

Linear programming has been subject of extensive research since George Dantzig used it to model the Air Force planning process during World War II in 1947. Development of the field of linear programming was due to the observation that the model also applies to a large number of economic, industrial and financial systems. In 1947 Dantzig proposed the *simplex algorithm*, the first practical method solving linear programs [DT97].

The simplex algorithm is iterative, where the total number of iterations depends on the choices made within each iteration. Although one can show that in the worst case it requires exponentially many iterations, in practice the method finishes almost always in a linear number of iterations [BT97, NW99].

Trying to solve linear programming problems as efficiently as possible, many variants of the simplex algorithms have been proposed and many completely new iterative methods have been developed. In 1979 Leonid Khachiyan showed that the linear programming problem is solvable in polynomial time, but a larger theoretical and practical breakthrough in the field came in 1984 when Narendra Karmarkar introduced a new *interior point method* for solving linear programming problems.

Nowadays, the simplex methods are still competitive to the interior point methods, where the interior point methods beat the simplex methods usually on some classes of very large problem instances [Mur05, Mil00].

At the time Dantzig invented the simplex algorithm there was no computer to run it. Thus, real life applications could not be solved because of their complexity. With this argument he persuaded the Pentagon to fund the development of computers. Their introduction has lead to a revolutionary treatment of planning processes [DT97]. The evolution of planning processes is still going on. The availability of fast computer networks and the desire of centralization lead to optimization problems involving many different parties sharing their data. This leads to new type of problems when some parties need to share data that they don't want to share.

For example, companies are often part of a *supply chain*, i.e., the collection of parties involved to provide customers with their needs. As a simple example, consider a customer buying some good from a warehouse. This warehouse most likely bought the good from distributors who are stocked by the manufacturers. The manufacturers may have bought raw materials from different suppliers in order to produce the good. The warehouses,

distributors, manufacturers and the suppliers of the raw materials are all part of the supply chain.

The process of optimizing the benefit of the supply chain is called *supply chain management*. It is well known that the best results are obtained by *collaborative supply chain management*, i.e., a process where all parties from the supply chain collaboratively compute the optimal configuration (stock quantities, prices, etc.) for the whole supply chain [AR93, CS60, LPG02, LB93]. In other words, if parties collaborate in optimizing the configuration of the supply chain, then the total benefit of the whole chain is optimized. This does not mean that the resulting configuration is best to all parties. However, when the benefit is properly shared among the parties then *all* parties will benefit.

In order to be able to collaboratively compute the optimal configuration of a supply chain, the parties need to share confidential data such as stocking costs, production capacities, shipping costs, prices and current sales data. Sharing this data implies risks on, for example, bargaining power. Indeed, if the exact costs of a supplier are public, customers can use this knowledge to obtain better prices. A solution to this problem is a system computing the desired optimum of the supply chain, where every party learns their part of the result and nothing more.

Solutions using multiparty computation are, for example, [Tof09] and [LA06]. Both solutions prove security using standard cryptographic techniques. Unfortunately they are able to solve only a sub-class of linear programs. Moreover, their performance allows solving very small problem instances only. Follow-ups to these solutions are [DK09, CH10b], focused on improving the performance.

While these solutions securely perform each step of an algorithm solving linear programs, other solutions such as [Vai09a, Vai09b, Du01, DK11, WRW11] apply a completely different approach: transform the given LP to a random LP securely, reveal it to all parties and solve the transformed LP publicly. In terms of efficiency, these solutions are clear winners. One only needs cryptographic primitives for the transformations, while the hard work —solving the actual LP— can be done without cryptography. But in terms of security it is unclear what can be guaranteed [BBR09]. A major issue is that the distribution of the transformations is unclear and certainly not uniform. Therefore, these solutions fall outside the scope of this thesis.

With the focus on secure implementations of Dantzig's simplex algorithm, this thesis provides

(i) an unified and rigorous description of relevant basic literature on linear programming and

(ii) describes efficient secure protocols for secure linear programming.

The description of the protocols will be such that any model of secure multiparty computation is applicable. However for the sake of simplicity and efficiency we will instantiate the protocols in the information theoretic setting in the presence of a passive adversary.

With respect to (i), we will describe the basic ideas behind the simplex algorithm. We show that its efficiency depends on several implementation choices. We point out what those choices are and what impact those will have in the corresponding secure protocols. For example, there are three classical implementations of the simplex algorithms of which the one known as the *revised simplex* is the most popular due to its efficiency. We show that, on the contrary, the secure protocols for the revised simplex algorithm will not be favorable.

With respect to (ii) we review and design efficient secure protocols for basic operations required by secure linear programming protocols. With respect to basic operations we provide an unified overview of secure protocols for integer arithmetic and for efficient generation of random numbers. We show how to apply these protocols to be able to perform secure arithmetic over truncated integers being represented by fixed point numbers. With respect to secure linear programming we show how to design efficient secure protocols from the implementations described in (i). For example, we show how to efficiently perform linear operations in a matrix without revealing the positions that are involved.

Roadmap of this Thesis

This thesis aims to be complete. We aim at providing every detail of the described protocols to provide full understanding that enables one to easily implement the protocols. Therefore, we will review existing tools of every protocol that is applied.

Below, we present an overview of the structure and contributions of this thesis.

Chapter 2: Preliminaries

This chapter introduces basic concepts of the cryptographic tools that are used in this thesis; with respect to the linear programming protocols our results are based on threshold linear secret sharing schemes.

Special attention will be paid to Σ-protocols and Σ-proofs that play an important role in Chapters 6 and 7. As a small contribution we present a proof that any Σ-protocol that is α-special sound, meaning that at least α accepting conversations are required to extract the secret, is a proof of knowledge by extending a proof from [Dam10] for the case $\alpha = 2$.

Chapter 3: Linear Optimization

This chapter discusses the problem of linear programming by a rigorous and unified description of the classic simplex algorithm that will form the basis for the secure implementations.

For linear programming problems on m constraints and n variables, we review the simplex iterations of the classical simplex algorithm, that updates an $(n + m + 1) \times (m + 1)$ matrix over $\mathbb{Q}$. In addition, we discuss extensions to the classical algorithm where the matrix updates are over $\mathbb{Z}$ and where smaller matrices are updated. We contribute a simple proof of an upper-bound of the values in the simplex tableaus needed to instantiate a cryptographic system.

Second, we present techniques to initialize the simplex iterations. Basically, this boils down to solving yet another linear program of which the iterations can be simply initialized. We will review two variants of algorithms for the two-phase simplex and big-M method. The classical algorithms for two-phase simplex and the big-M method require the addition of n columns to the simplex matrix. More advanced solutions show that in fact only one additional column suffices.

Finally, we show basic techniques to validate the results returned by the simplex algorithm. This will be important in secure simplex, since one may have good reasons to doubt validity of the result. The elegance of these techniques formed an inspiration to efficient universal verifiable protocols in Chapter 6.

Chapter 4: Building Blocks for Secure Linear Programming

This chapter provides an overview of efficient protocols with respect to Shamir's secret sharing scheme that forms a basis for the protocols in Chapter 5. We will focus on efficiency, but ensure security to be either perfect or statistical. This is joint work with Octavian Catrina [CH10a, Sec09].

First, we will show how to combine some properties of Shamir's secret sharing scheme with the multiplication protocol of [BGW88] to optimize performance. More precisely, we review the noninteractive generation of secret random numbers from [CDI05, DT08] for Shamir shared values, which are very efficient if the number of participating parties is limited. Based on [BGW88, CDI05], we will contribute an inner product protocol that has same round complexity and communication complexity as a single multiplication. This will play a central role in secure linear programming.

Second, given an associative binary operator $\odot$, we review techniques to do efficient

- *k-ary operation:* $y = x_1 \odot \cdots \odot x_k = \odot_{i=1}^{k} x_i$, and
- *prefix operation:* $y_j = \odot_{i=1}^{j} x_i$ for each $j = 1, \ldots, k$.

We will give generic protocol descriptions of logarithmic rounds for any associative binary operator. For important tools such as multiplication and prefix-or we contribute more efficient implementations based on descriptions in the literature.

Third, we consider the problem of integer comparison following the approach of [ST06]. To balance between round an communication complexity we present protocols to securely compute the result of comparisons of the form $x \leq y$ from both [VB10] having logarithmic round complexity and [Rei09] having constant round complexity. We contribute a new protocol for securely computing the result of an equality comparison that has $\log^*$ round complexity, where $\log^*(k) = \min\{i | \log^i(k) \leq 1\}$.

To enable implementation of the simplex algorithm on a matrix over $\mathbb{Q}$ we review basic protocols for fixed point arithmetic based on [CS10]. We contribute an improved protocol for division [CH10b].

Finally, we discuss the technique of hiding entries in a matrix by means of secret indexing [Tof09] and its applications with respect to linear programming.

Chapter 5: Secure Linear Programming

This chapter shows how to build secure multiparty protocols from the simplex algorithms described in Chapter 3 using the tools described in Chapter 4. Our approach follows the ideas of Toft [Tof09]. This is joint work with Octavian Catrina [Sec10].

First, we will describe how to build secure protocols for the simplex iterations. We contribute a full set of protocols evaluating each variant of the simplex algorithm with both integer tableaus and rational tableaus [CH10b]. We focus on efficiency; we will show several ideas and tricks to minimize the communication complexity and even improving the ideas from [CH10b, Sec10].

Second, we contribute a full set of protocols for initializing the simplex algorithms. We show what additional problems arise when securely connecting the phases of the two-phase simplex algorithms. We will show how to efficiently and securely implement the two-phase simplex algorithms and the big-M methods. We discuss that the choice between those implementations depends on how one balances between security and efficiency.

Finally, we contribute very efficient protocols for validating the result of the simplex protocol. We show how to extract a certificate that proves correctness of the result.

Chapter 6: Universal Verifiability

This chapter shows how to convert the protocols of [CDN01] into universally verifiable protocols. This is joint work with Berry Schoenmakers.

First we will define universal verifiability. Then, we will show how to make the protocols of [CDN01] universally verifiable by means of transforming interactive Σ-protocols into noninteractive Σ-proofs in such a way that the security of the original protocols are maintained in the random oracle model.

Second, we show that universal verifiability in secure circuit evaluation in general does not require every gate of a circuit that is evaluated to be universal verifiable. For universally verifiable secure linear programming, for example, just gates computing the result of the validation of the certificate of correctness need to be universally verifiable.

Chapter 7: Verifiable Restricted Shuffling

This chapter shows how to apply [TW10] to prove correctness of a restricted shuffle as an alternative to [HSSV09]. This is joint work with Berry Schoenmakers, Boris Skoric and José Villegas.

To apply [TW10] we need to find (hyper)graphs of which the automorphism group is exactly the permutation group of the restricted shuffle. We show how to find such hypergraphs for the following restricted shuffles: rotation, affine transformation and Möbius transformation. We contribute simple graphs and proofs showing the desired properties.

Chapter 2

Cryptographic Primitives

This chapter presents cryptographic primitives and tools that form the basis for the results in the thesis. In addition, we will introduce terminology and notation that is used throughout the thesis.

We review secure multiparty computation based on Shamir's secret sharing [Sha79]. The replicated secret sharing scheme of [ISN87] is also discussed, which allows very efficient generation of Shamir shares of random values if the number of participating parties is small [CDI05].

Special attention will be paid to Σ-protocols and Σ-proofs that play an important role in Chapters 6 and 7. As a small contribution we present a proof that any Σ-protocol that is α-special sound, meaning that at least α accepting conversations are required to extract the secret, is a proof of knowledge by extending a proof from [Dam10] for the case $\alpha = 2$.

2.1 Basic Primitives

This section discusses the basic notations, assumptions and cryptographic schemes.

Vector Notation

We let $\mathbb{Z}$ denote the set of integers and we use $\mathbb{Z}_p$ as a shorthand notation of the set $\mathbb{Z}/p\mathbb{Z}$.

For any set $\mathcal{M}$ we use bold lowercase letters to denote a vector $\mathbf{v} \in \mathcal{M}^n$ of length n. By v_i we denote the i-th entry of $\mathbf{v}$. We use bold uppercase letters to denote a two dimensional matrix $\mathbf{M} \in \mathcal{M}^{n \times m}$ of n rows and m columns. By $\mathbf{m}_i$ we denote the i-th row of $\mathbf{M}$ and by $\mathbf{M}_j$ we denote the j-th column of $\mathbf{M}$. Finally, m_{ij} denotes the entry in row i and column j of $\mathbf{M}$.

Hardness Assumptions

We will give three basic hardness assumptions in the discrete log setting. Consider a cyclic group $G = \langle g \rangle$ of order prime p. The *discrete log (DL) assumption* is that it is infeasible given $h \in G$ to compute $\alpha \in \mathbb{Z}_p$ such that $h = g^\alpha$. A potentially stronger assumption based on the DL assumption is the *Diffie-Hellman (DH) assumption* saying that given g^α and g^β, it is infeasible to compute $h = g^{\alpha\beta}$. Lastly, the Decisional Diffie-Hellman (DDH) assumption is that it is infeasible given (g^α, g^β, h) to distinguish between h that is uniformly random and h that equals $g^{\alpha\beta}$.

We also give three assumptions in the RSA-setting. Consider a composite N having two prime factors p and q. The *factorization assumption* is that it is infeasible to find p and q given N. The *(strong) RSA assumption* is that given modulus N and $c \in \mathbb{Z}_N^*$ it is

infeasible to find m and $e > 1$ that satisfies $c = m^e \mod N$. The *Decisional Composite Residue (DCR) assumption* says that given $y \in \mathbb{Z}_{N^2}$ it is infeasible to decide whether an $x \in \mathbb{Z}_{N^2}$ exists such that $y = x^N \mod N^2$.

2.1.1 Indistinguishability

A *distribution ensemble* is a set $\mathcal{X} = \{X_i\}_{i \in \mathcal{I}}$, where X_i is a random variable and $\mathcal{I}$ an index set. We only consider random variables from a finite set.

Definition 2.1 (Statistical Distance)**.** *Let X and Y be two random variables, both taking values in some finite set V. The statistical distance between X and Y is defined as*

$$\Delta(X;Y) = \frac{1}{2} \sum_{v \in V} |\mathbb{P}[X = v] - \mathbb{P}[Y = v]|. \tag{2.1}$$

Definition 2.2 (Negligible)**.** *A nonnegative function $\delta : \mathbb{N} \to \mathbb{R}$ is called* negligible *if for every positive polynomial p there exists a $k_0 \in \mathbb{N}$ such that $\delta(k) \leq 1/p(k)$ for all $k \geq k_0$.*

Definition 2.3. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two distribution ensembles indexed by $\mathcal{I}$. Suppose that the sizes satisfy $|X_i| = |Y_i|$ for $i \in \mathcal{I}$ and all sizes are polynomial in $|i|$. Then, $\mathcal{X}$ and $\mathcal{Y}$ are said to be*

perfectly indistinguishable *if $\Delta(X_i; Y_i) = 0$ for all $i \in \mathcal{I}$. We write $\mathcal{X} \stackrel{d}{=} \mathcal{Y}$.*

statistically indistinguishable *if for all $i \in \mathcal{I}$, $\Delta(X_i; Y_i)$ is negligible as a function of $|i|$. We will write $\mathcal{X} \stackrel{s}{=} \mathcal{Y}$.*

computationally indistinguishable *if for all p.p.t. algorithms D and for all $i \in \mathcal{I}$*

$$|\mathbb{P}[D(X_i) = 1] - \mathbb{P}[D(Y_i) = 1]|$$

is negligible as a function of $|i|$. We will write $\mathcal{X} \stackrel{c}{=} \mathcal{Y}$.

2.1.2 Secret Sharing

In a secret sharing scheme parties $P_1, \ldots, P_n$ share their inputs among all parties in such a way that some agreed subset of parties, called *qualified set*, can reconstruct the secret, while any subset of parties that is not a qualified set cannot learn anything about the secret. The party sharing its secret among the other parties is called the *dealer*. A secret sharing scheme with n parties is called a (t, n)-secret sharing scheme, if any subset of $t < n$ parties is not a qualified set but any subset of $t + 1$ parties is.

Typically, a secret sharing scheme consists of three phases. In the *share generation* phase, the dealer generates shares of some secret s for each party. Then, the dealer provides each party P_k his share of s, denoted by $[s]_k$, in the *share distribution* phase. Let $[s]$ denote the collection of all shares of s. Finally, in the *reconstruction* phase, the parties compute s from pooling their shares.

We will present Shamir's secret sharing scheme, the standard additive secret sharing scheme and the replicated secret sharing scheme. In the following, if $\mathcal{S}$ is a set, then a uniformly random draw from $\mathcal{S}$ resulting in s is denoted by $s \in_R \mathcal{S}$. All arithmetic in this section is over some finite field $\mathbb{F}_q$.

The (t, n)-Shamir secret sharing scheme (Protocol 2.1 and Protocol 2.2) is as follows:

1. *Share Generation:* To share $s \in \mathbb{F}_q$, the dealer generates random $\alpha_1, \ldots, \alpha_t \in \mathbb{F}_q$ and puts $p(x) = s + \alpha_1 x + \cdots + \alpha_t x^t$. Then the dealer computes $[s]_i = p(i)$.

2. *Share Distribution:* For each $i \in \{1, \ldots, n\}$, the dealer sends $[s]_i$ to party P_i.

3. *Secret Reconstruction:* Let $\mathcal{D} \subset \{1, \ldots, n\}$ be a set of size $t+1$. Each party P_i for $i \in \mathcal{D}$ sends his share $[s]_i$ to all parties. Then, each party reconstructs the secret via Lagrange interpolation:

Protocol 2.1: $[s] \leftarrow \mathsf{SShare}(i, t, s)$

for party P_i **do**
  pick $\alpha_1, \ldots, \alpha_t \in_R \mathbb{F}_q^t$;
  foreach $j = 1, \ldots, n$ **do**
    $[s]_j \leftarrow s + \sum_{\ell=1}^{t} \alpha_\ell j^\ell$;
    send $[s]_j$ **to party** P_j;
return $[s]$

Protocol 2.2: $[s] \leftarrow \mathsf{SOpen}(\mathcal{D}, [s])$

foreach party $P_i \in \mathcal{S}$ **do send** $[s]_i$ **to all parties**;
$s = \sum_{i \in \mathcal{D}} [s]_i \prod_{j \in \mathcal{D}, j \neq i} \frac{-j}{i-j}$;
return s

Additive secret sharing (Protocol 2.3 and Protocol 2.4) is as follows:

1. *Share Generation:* To share $s \in \mathbb{F}_q$, the dealer generates random $n-1$ random values $s_1, \ldots, s_{n-1}$ and computes $s_n = s - \sum_{i=1}^{n-1} s_i$.

2. *Share Distribution:* The dealer sends share $[s]_i^A = s_i$ to each party P_i.

3. *Secret Reconstruction:* Each party P_i sends his share $[s]_i$ to all other parties. The parties compute $s = \sum_{i=1}^{n} [s]_i$.

It follows that only the collection of all parties can reconstruct the secret by adding their shares, while any other collection of parties misses at least one share and will learn nothing about s.

Protocol 2.3: $[s] \leftarrow \mathsf{AShare}(i, s)$

for party P_i **do**
  pick $[s]_1^A, \ldots, [s]_{n-1}^A \in_R \mathbb{F}_q^{n-1}$;
  foreach $j = 1, \ldots, n$ **do**
    send $[s]_j^A$ **to party** P_j;
return $[s]^A$

Protocol 2.4: $[s] \leftarrow \mathsf{AOpen}([s]^A)$

1 **foreach party** P_i **do send** $[s]_i$ **to all parties**;
2 $s = \sum_{i=1}^{n} [s]_i^A$;
3 **return** s

Replicated secret sharing may be viewed as a generalization of additive secret sharing. The idea is to allow any qualified set of parties to reconstruct the secret by adding (some of) their shares. In our applications we will use replicated secret sharing to noninteractively compute random Shamir shares.

Let $\mathcal{T} = \{T_1, \ldots, T_w\}$ be the collection of all possible subsets of parties of size t, where $w = \binom{n}{t}$. The (t, n)-replicated secret sharing scheme is as follows:

1. *Share Generation:* To share s, the dealer generates $w-1$ random values $s_1, \ldots, s_{w-1}$ and computes $s_w = s - \sum_{i=1}^{w-1} s_i$.

2. *Share Distribution:* The dealer sends $[s]_i^R = s_i$ to each party not in T_i.

3. *Secret Reconstruction:* Let $\mathcal{D} \subset \{1, \ldots, n\}$ be a set of size $t+1$. All parties P_i for $i \in \mathcal{D}$ pool their shares and reconstruct s by $s = \sum_{i=1}^{w} [s]_i$.

It follows that since any subset of at most t parties is a subset of at least one $\mathcal{T}_j$, they miss at least one share of s. By construction, each set of $t+1$ parties can reconstruct s, see Protocol 2.6, while any set of at most t parties learn nothing about s.

Protocol 2.5: $[s] \leftarrow \mathsf{RShare}(i, \mathcal{T}, s)$

1 $w \leftarrow |\mathcal{T}|$;
2 **for party** P_i **do**
3 **pick** $[s]_1^R, \ldots, [s]_{w-1}^R \in_R \mathbb{F}_q^{w-1}$;
4 **foreach** $j = 1, \ldots, n$ **do**
5 **send** $[s]_j^R$ **to each party** $P_j \notin \mathcal{T}_j$;
6 **return** $[s]^R$

Protocol 2.6: $[s] \leftarrow \mathsf{ROpen}(\mathcal{D}, \mathcal{T}, [s]^R)$

1 $w \leftarrow |\mathcal{T}|$;
2 **foreach party** $P_i \in \mathcal{D}$ **do**
3 **foreach** j **s.t.** $P_i \notin \mathcal{T}_j$ **do**
4 **send** $[s]_j^R$ **to all parties**;
5 $s = \sum_{k=1}^{w} [s]_k^R$;
6 **return** s

2.1.3 Commitment Schemes

In a commitment scheme, a committer C and a receiver R run the following scheme that consists of two-phases. In the committing phase, C chooses some random number and commits to x by sending $c = b(x, r)$ to R, for some function b. In the opening phase, C sends x and r to R who accepts only if $c = b(x, r)$.

A *commitment scheme* satisfies the following properties: *hiding* and *binding*. Informally, we say that a commitment scheme is hiding if no information about the committed value x is revealed by $b(x, r)$, and we say that it is binding if no committer can compute values x, x' and randomness r, r' such that $b(x, r) = b(x', r')$.

A classic example of a commitment scheme in the discrete log setting is Pedersen's commitment scheme. Suppose that $G = \langle g \rangle$ is a group of prime order p and $h \in G$ from which $\log_g(h)$ is unknown. The Pedersen's commitment scheme is as follows:

Pedersen's Commitment:

1. *Commitment:* To commit on value $x \in \mathbb{Z}_p$, C picks a uniform random $r \in \mathbb{Z}_p$ and sends
$$c = b(x, r) := g^x h^r$$
to R.

2. *Opening:* To open commitment c the sender sends x and r to R. Upon reception of x' and r' from C, R accepts if $c = g^{x'} h^{r'}$.

This scheme is *perfectly* hiding, since for any x the distribution of $b(x, r)$ for a uniformly random r is uniform over G. On the other hand the scheme is *computationally* binding under the DL assumption. Indeed, if C is able to compute x, x' and r, r', then from $g^x h^r = g^{x'} h^{r'}$ it follows that $h = g^{\frac{x-x'}{r'-r}}$.

A commitment scheme is called a *trapdoor commitment scheme* if the binding property is only satisfied if some trapdoor remains unknown. This property is useful for example to enable simulation of multiparty Σ-protocols used in for example [CDN01].

Pedersen's commitment scheme has a trapdoor on the binding property, namely $s = \log_g h$. Knowing s a committer can open any commitment $c = g^x h^r$ arbitrarily. Indeed, given $s \in \mathbb{Z}_p$ such that $h = g^s$, using the relation $x + sr = x' + sr'$, one computes x, x', r and r' such that $c = g^x h^r = g^{x'} h^{r'}$.

2.2 Zero-Knowledge Proofs

Let R be an NP relation. That is, $R = \{(x; w)\} \subset \{0,1\}^* \times \{0,1\}^*$ is a binary relation and it can be verified in polynomial time in $|x|$ whether $(x; w)$ belongs to R. The *language* induced by R is the set $L_R = \{x \mid \exists w : (x; w) \in R\}$.

A *zero-knowledge proof for relation* R is a two party scheme between a *prover* P and a *verifier* V, where both have input x. The aim of such a scheme is that P convinces V that he knows a *witness* w such that $(x, w) \in R$, without revealing any information about w.

2.2.1 Σ-protocols

Classical examples of zero-knowledge proofs are Σ-protocols. A Σ-protocol is a 3-round protocol, where the prover sends the first message a. The verifier responds with some random challenge c from which the prover computes the third message r based on his first message, the challenge and the witness. More precisely, a Σ-protocol is of the form as outlined in Figure 2.1 and is defined in the following definition, see also [Sch12]. The transcript of the protocol is denoted by (a, c, r) and is also known as a *conversation*. A conversation (a, c, r) is called *accepting* if the verifier accepts on input (a, c, r).

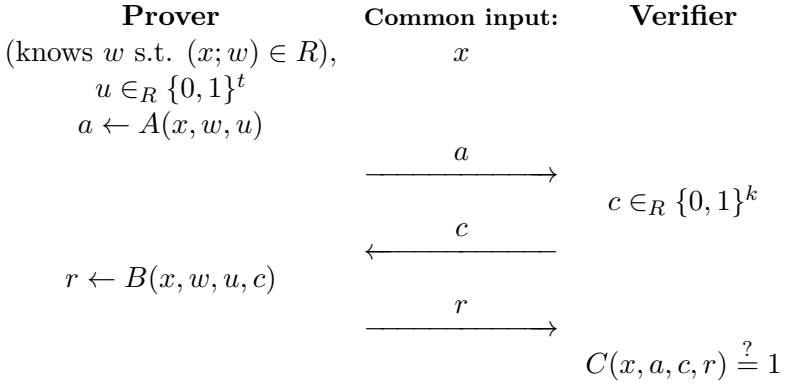


Figure 2.1: Σ-protocol for relation R, where A, B, C are p.p.t. algorithms

Definition 2.4 (Σ-protocol)**.** *A 3-round protocol of the form shown in Figure 2.1 is a Σ-protocol for relation R if the following three properties are satisfied:*

Completeness: *If P and V are honest, then V always accepts.*

Special soundness: *There exists a probabilistic polynomial time (p.p.t.) extractor E that on input x and two accepting conversations (a, c, r) and (a, c', r'), where $c \neq c'$, computes a witness w such that $(x; w) \in R$.*

Special honest verifier zero-knowledge: *There exists a p.p.t. simulator S that on input any $x \in \{0,1\}^*$ and any c computes accepting conversations (a, c, r). If $x \in L_R$, then the conversations returned by S have the same probability distribution as the conversations between honest P and honest V have on input x and c, where P uses any valid witness w such that $(x; w) \in R$.*

The special soundness property implies that, if any prover is able to provide two accepting conversations with the same initial message and two different challenges, a valid witness can be computed in polynomial time. One can prove, see for example [Dam10], that the special soundness property implies knowledge soundness [BG92] with knowledge error 2^{-k}.

Definition 2.5. *Let the knowledge error be given by the function $\kappa : \{0,1\}^* \to [0,1]$. A protocol between P and V is a proof of knowledge for relation R if the following is satisfied.*

Completeness: *If P and V are honest, then V always accepts.*

Knowledge soundness: *For any prover P^*, where $\epsilon(x) > \kappa(x)$ denotes the probability that V accepts on input x, there exists a probabilistic algorithm M that on input x and rewindable black-box access to P^* computes w such that $(x; w) \in R$ in expected time*

$$\frac{|x|^c}{\epsilon(x) - \kappa(x)},$$

where c is some constant. Any access to P^* is counted as one unit of time.

Hence, every corrupted prover P^* not knowing a valid witness has essentially a probability of 2^{-k} to let the verifier accept.

As a slight generalization, we show that a weaker special soundness requirement on Σ-protocols also suffices to achieve knowledge soundness.

Definition 2.6 (α-sound Σ-protocol)**.** *A 3-round protocol of the form shown in Figure 2.1 is an α-sound Σ-protocol for relation R if it satisfies the completeness and special honest verifier zero-knowledge properties of Definition 2.4 and also with respect to soundness satisfies*

α-Special soundness: *There exists a p.p.t. extractor E that on input x and $\alpha \geq 2$ accepting conversations $(a, c_1, r_1), \ldots, (a, c_\alpha, r_\alpha)$, where $c_i \neq c_j$ for all $1 \leq i < j \leq \alpha$, computes a witness w such that $(x; w) \in R$.*

The following theorem shows that for any constant α the knowledge error is exponentially small in the security parameter k.

Theorem 2.7. *An α-special sound Σ-protocol for relation R with challenge length t bits is a proof of knowledge with knowledge error $(\alpha - 1)2^{-k}$.*

Proof. We extend the proof of [Dam10], which covers the basic case of $\alpha = 2$.

To prove knowledge soundness with knowledge error $(\alpha - 1)2^{-k}$, consider any prover P^* with success probability $\epsilon > (\alpha - 1)2^{-k}$. Let $\mathbf{H}$ be the binary matrix having one row for each possible random choice u of P^* and one column for each possible challenge c, where each entry $h_{u,c}$ equals 1 if and only if for these random choices of P^* and challenge c the verifier accepts. Running P^* as a black-box and choosing a challenge at random, we can probe a random entry in $\mathbf{H}$. By rewinding P^* we can probe $\mathbf{H}$ in the same row since P^* will use the same randomness as before.

Our goal is to find α ones in a single row, as the corresponding accepting conversations allow us to extract a witness w in polynomial time using the α-special soundness extractor E.

To this end, we use algorithm M which runs the following two algorithms, named M_1 and M_2, in parallel.

Algorithm M_1:

1. Probe $\mathbf{H}$ until a 1 is found (first hit) in, say row u.
2. Probe a random entry in row u. If α ones are found in row u, return the α corresponding accepting conversations and stop.

3. Pick $r \in_R \{1, 2, \ldots, 4\alpha^3\}$. If $r = 1$, probe $\mathbf{H}$ randomly and if a 1 is found go to Step 1.
4. Go to Step 2.

Algorithm M_2:

1. Probe $\mathbf{H}$ until a 1 is found (first hit) in row u, say.
2. Search row u for $\alpha - 1$ other 1 entries. If success, return the α corresponding accepting conversations and stop.
3. Go to Step 1.

We will complete the proof by showing that M runs in expected time

$$O\left(\frac{1}{\epsilon - (\alpha-1)2^{-k}}\right).$$

We distinguish two cases.

Case I: $\epsilon \geq \frac{\alpha^2}{\alpha-1}2^{-k}$. In this case we show that the expected runtime of M_1 (and therefore of M) is $O(1/\epsilon)$, which is sufficient.

We call a row of $\mathbf{H}$ heavy if it contains at least a fraction of $\frac{\alpha-1}{\alpha}\epsilon$ ones. We will show (i) that the probability that a first hit in Step 1 of M_1 is in a heavy row is at least $1/\alpha$, (ii) that the expected time M_1 spends in Steps 2 and 3 after a first hit is $O(1/\epsilon)$, and (iii) that if the first hit is in a heavy row then with probability at least $1/4$, M_1 will terminate in this row. Since α is constant this will imply that the expected runtime of M_1 is indeed $O(1/\epsilon)$.

(i) We show that all heavy rows together contain at least a fraction of $\frac{1}{\alpha}$ of all ones in $\mathbf{H}$. Let $\mathbf{H}'$ be the sub-matrix consisting of non-heavy rows of $\mathbf{H}$ and let $v_{h'}$ denote the number of entries in $\mathbf{H}'$ and similarly v_h denote the number of entries in $\mathbf{H}$. The number of ones in the heavy rows g satisfies

$$g > v_h\epsilon - v_{h'}\frac{\alpha-1}{\alpha}\epsilon \geq v_h\epsilon - v_h\frac{\alpha-1}{\alpha}\epsilon = \frac{v_h\epsilon}{\alpha}. \tag{2.2}$$

By the assumption on ϵ the number of ones in a heavy row is at least $\frac{\alpha-1}{\alpha}\epsilon 2^k \geq \alpha$.

(ii) By construction the expected time M_1 spends in a row does not exceed $\frac{4\alpha^3}{\epsilon} = O(1/\epsilon)$.

(iii) Let T denote the expected time to find $\alpha - 1$ ones after the first hit (not counting the time spent in Step 3). We show that if the first hit is in a heavy row, then with probability at least $1/2$, M_1 terminates within time $2T$, and further that with probability at least $1/2$, M_1 will not return to Step 1 within time $2T$.

First, let τ denote the number of probes to find $\alpha - 1$ ones in the same row of the first hit. From Markov's inequality we get that

$$\mathbb{P}[\tau > 2T] \leq T/(2T) = 1/2.$$

Hence, the probability that M_1 requires less than $2T$ probes to find $\alpha - 1$ distinct ones when the first hit is in a heavy row is larger than $1/2$.

To show that M_1 will not return to Step 1 within time $2T$ we need to compute a bound on T explicitly. To this end, suppose that $i < \alpha$ ones have been found in a heavy row. Then, there are at least $\frac{\alpha-1}{\alpha}\epsilon 2^k - i \geq \alpha - i > 0$ ones left in this row. Therefore, the expected number of probes, T_i, to find the $(i+1)$-st distinct 1 in this row satisfies

$$T_i = \frac{2^k}{(\alpha-1)/\alpha \cdot 2^k\epsilon - i} \leq \frac{\alpha^2}{\epsilon}, \tag{2.3}$$

for each $i = 1, \ldots, \alpha - 1$.

It follows that the expected number of probes to find $\alpha - 1$ ones when the first hit is in a heavy row satisfies

$$T = \sum_{i=1}^{\alpha-1} T_i \leq \frac{\alpha^3}{\epsilon}.$$

Let B_i denote the event that at the i-th invocation of Step 3 after the last first hit M_1 returns to Step 1. From the union bound we have the following inequality for the probability that M_1 requires more than $2T$ probes before it returns to Step 1:

$$1 - \mathbb{P}\left[\bigcup_{i=1}^{2T} B_i\right] \geq 1 - 2T\frac{\epsilon}{4\alpha^3} \geq 1 - 2\frac{\alpha^3}{\epsilon}\frac{\epsilon}{4\alpha^3} = \frac{1}{2}.$$

Case II: $\epsilon < \frac{\alpha^2}{\alpha-1}2^{-k}$. In this case we show that the expected runtime of M_2 (and therefore of M) is $O(1/(\epsilon - (\alpha-1)2^{-k}))$.

Let $\delta > 0$ be such that $\epsilon = (1+\delta)(\alpha-1)2^{-k}$. Then

$$0 < \delta < \frac{\alpha^2}{\alpha - 1}.$$

Observe that $\mathbf{H}$ has 2^{k+t} entries from which at least $(1+\delta)(\alpha-1)2^t$ are equal to 1. At most $(\alpha-1)2^t$ of these ones can be in rows having at most $(\alpha-1)$ ones. Thus, at least $\delta(\alpha-1)2^t$ of the ones are in rows having at least α ones. We call a row that has at least α ones semi-heavy.

Note that the fraction of ones in semi-heavy rows is at least

$$\frac{\delta(\alpha-1)2^t}{(1+\delta)(\alpha-1)2^t} = \frac{\delta}{\delta+1}$$

among all ones in $\mathbf{H}$ and

$$\frac{\delta(\alpha-1)2^t}{2^{t+k}} = \frac{\delta(\alpha-1)}{2^k}$$

among all entries in $\mathbf{H}$.

Hence, the probability that the first hit is in a semi-heavy row is $\delta/(1+\delta)$. For each first hit M_2 requires 2^k probes to search the entire row. Therefore, we expect to need $(1+\delta)/\delta 2^k$ probes in total for probing the entire row after each first hit. In addition to find a first hit in a semi-heavy row requires $((\alpha-1)\delta)^{-1}2^k$ probes.

In conclusion, the expected runtime of M_2 is equal to

$$2^k \left(\frac{1}{(\alpha-1)\delta} + \frac{\delta+1}{\delta} \right) = O\left(\frac{2^k}{(\alpha-1)\delta} \right).$$

This is nothing more than the time allowed:

$$\frac{1}{\epsilon - (\alpha-1)2^{-k}} = \frac{1}{(1+\delta)(\alpha-1)2^{-k} - (\alpha-1)2^{-k}} = \frac{2^k}{(\alpha-1)\delta}$$

□

2.2.2 Noninteractive zero-knowledge proofs

Noninteractive zero-knowledge proof systems (NIZK) are introduced in [BFM88]. A noninteractive proof system is a proof system where the only interaction between P and V is that P sends a message σ to V and V decides whether he accepts or rejects σ.

In this section we consider noninteractive versions of any Σ-protocol of the form given in Figure 2.1 by means of the Fiat-Shamir transform [FS86] and its generalized form [AABN02].

In [FS86] it is observed that interaction with an honest verifier can be removed in the random oracle model using a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ with security parameter k to compute a random challenge as follows. Let a be the first message of the Σ-protocol, then the prover computes a random challenge himself by $\mathcal{H}(a)$.

If $\mathcal{H}$ is public accessible, then if P broadcasts $(a, c = \mathcal{H}(a), r)$ anyone can check whether P knows a witness w such that $(v, w) \in R$, by checking whether (a, c, r) is accepting and that $c = \mathcal{H}(a)$; see also Protocol 2.7. This protocol is also known as a Σ-*proof*.

Security follows in the random oracle model [BR93]. However, in contrast to a real honest verifier $\mathcal{H}(a)$ is a fixed value, meaning that the oracle will return the same result each time it is queried on a. Therefore, it is not straightforward to show that the special soundness property from the underlying Σ-protocol implies knowledge soundness with respect to the Σ-proof. The problem is that rewinding a prover will not lead to two accepting conversations of the form (a, c, r) and (a, c', r'), where $c \neq c'$ since the random oracle $\mathcal{H}$ has a fixed output on input a.

Protocol 2.7: $\sigma \leftarrow \mathsf{FS}(\Sigma = (A, B), \mathcal{H}, x, w, t)$

1 $u \in_R \{0,1\}^t$;
2 $a \leftarrow A(x, w, u)$;
3 $c \leftarrow \mathcal{H}(a)$;
4 $r \leftarrow B(x, w, u, c)$;
5 **return** (a, c, r);

In [PS00] Pointcheval and Stern show how to produce two accepting proofs (a, c, r) and (a, c', r') from a prover P^* that provides accepting proofs of Protocol 2.7 with probability ϵ in time $O(1/\epsilon)$. Their result is also known as the *Forking Lemma*.

Theorem 2.8 (Forking Lemma (simplified version))**.** *Consider the Σ-protocol 2.7. Let P^* be a p.p.t. prover that can query the random oracle at most Q times. Suppose that P^* produces an accepting proof (a, c, r) with probability $\epsilon \geq 7Q/2^k$. Then there exists a p.p.t. M that controls P^* and produces two accepting conversations (a, c, r) and (a, c', r') such that $c \neq c'$ in expected time $O(Q/\epsilon)$, where each invocation of P^* counts a single step.*

Proof. This is a direct consequence of [PS00, Theorem 1] by considering the special case, where $m = \emptyset$.

A sketch of the proof is as follows. Firstly, one shows that if P^* provides an accepting proof (a, c, r), then with overwhelming probability P^* has queried $\mathcal{H}$ on input a. In other words, let $q_1, \ldots, q_Q$ denote the sequence of queries made by P^* to the random oracle $\mathcal{H}$, then with overwhelming probability there exists an $i \in \{1, \ldots, Q\}$ such that $q_i = a$.

Secondly, suppose that P^* provides an accepting proof (a, c, r), while having queried $\mathcal{H}$ on $q_1, \ldots, q_Q$, where $q_i = a$. Let $\mathcal{H}'$ be a random oracle such that $\mathcal{H}'(q_j) = \mathcal{H}(q_j)$ for all $j < i$ and $\mathcal{H}'(q_i) \neq \mathcal{H}(q_i)$. Then one shows that after $O(Q/\epsilon)$ *replays* of P^* with oracle $\mathcal{H}'$ the prover P^* provides a second accepting proof (a, c', r') with $c' = \mathcal{H}'(a) \neq \mathcal{H}(a) = c$ with some constant probability. A replay of P^* means that the prover is rewound to its starting position.

The sequences $\mathcal{H}(q_1), \ldots, \mathcal{H}(q_Q)$ and $\mathcal{H}'(q_1), \ldots, \mathcal{H}'(q_Q)$, where $\mathcal{H}'(q_j) = \mathcal{H}(q_j)$ for all $j < i$ and $\mathcal{H}'(q_i) \neq \mathcal{H}(q_i)$ are called a fork. A fork is called successful if P^* provides two accepting proofs (a, c, r) and (a, c', r'), where $c \neq c'$ after some integer N replays. This integer N is defined so that the total expected runtime is $O(Q/\epsilon)$.

In conclusion, let N_j be some integers. Then, M is on a high level defined by

1. Initialize $\ell = 0$.
2. Set $\ell = \ell + 1$ and run P^* until it provides an accepting proof (a, c, r). Let the queries made by P^* in the last run be denoted by $q_1, \ldots, q_Q$.
3. Let i be such that $q_i = a$. If no such i exists, then return to 2, else pick a new random oracle $\mathcal{H}'$ such that $\mathcal{H}'(q_j) = \mathcal{H}(q_j)$ for all $j < i$ and $\mathcal{H}'(q_i) \neq \mathcal{H}(a)$.
4. Replay P^* N_ℓ times. If the fork is successful then return the two accepting conversations, else return to 2.

□

Observe that Theorem 2.8 implies that the Σ-proof of Protocol 2.7 is a proof of knowledge with knowledge error $\kappa = 7Q/2^k$. Indeed, one can run M to let P^* produce two accepting proofs (a, c, r) and (a, c', r') in time $O(Q/\epsilon)$. Then one runs the extractor E of the Σ-protocol of Definition 2.4 to obtain a witness in an additional polynomial time.

In [AABN02] Abdalla et al. provide a different method to get a non-interactive Σ-proof, which they call the Generalized Fiat-Shamir Transform. Instead of computing the random challenge by $\mathcal{H}(a)$ one generates a random bit string z and computes the challenge by $\mathcal{H}(z, a)$. The resulting Σ-proof is given by Protocol 2.8.

Protocol 2.8: $\sigma \leftarrow \mathsf{GFS}(\Sigma = (A, B), \mathcal{H}, x, w, t, s)$

```
1 u ∈_R {0,1}^t;
2 a ← A(x, w, u);
3 z ∈_R {0,1}^s;
4 c ← H(z, a);
5 r ← B(x, w, u, c);
6 σ ← (z, a, c, r);
7 return σ;
```

One can show that the Σ-proof of Protocol 2.8 is also a proof of knowledge in the random oracle model.

Theorem 2.9. *Protocol 2.8 is a proof of knowledge.*

Proof. Abdalla et al. show in [AABN02] that if some prover P^* has non-negligible probability to forge accepting proofs, then one can use P^* to forge accepting conversations in the corresponding Σ-protocol of Figure 2.1 with non-negligible probability.

Thus, by Theorem 2.7 it follows that a witness can be extracted. □

2.3 Multiparty Computation Model

There are many ways to define security of cryptographic protocols. Most of them use the *simulation paradigm* which roughly states that if an adversary's view from a protocol execution can be generated from everything it is allowed to know, then the protocol is secure. Intuitively, this makes sense since if everything the adversaries observes in a protocol execution can be generated by himself, he learns nothing from the protocol execution.

Often, these models contain two worlds: the ideal world and the real world. In both worlds there are n parties $P_1, \ldots, P_n$. The parties wish to jointly compute the result of some function f, while revealing nothing about the private inputs and private outputs. In the real world the parties execute a protocol π in the presence of an adversary $\mathcal{A}$. In the ideal world, on the other hand, the parties send their inputs via a secure connection to a trusted party that replies with the desired result. The ideal world is such that any adversary $\mathcal{S}$, also called simulator, learns only public data and private data of the corrupted parties. If there exists an $\mathcal{S}$ that runs in expected polynomially time in the ideal world that generates views that are indistinguishable from the views by executing π, then one concludes that π securely evaluates f.

Canetti provides in [Can00] a model based on the simulation paradigm, that allows modular composition of protocols, i.e., the design of protocols, where simpler protocols are invoked as subroutines. We will discuss in this section this model, since it is simple and sufficient for the remainder of this thesis.

The model of [Can00] captures secure circuit evaluation. Circuit evaluation means that the computation of a function f is done by constructing an *arithmetic circuit*. The modular composition theorem of [Can00] states that if the circuit is secure under the assumption that the gates are secure and if all gates are secure, then the circuit is secure.

We will now present the real model and the ideal model for a static adversary, i.e., an adversary that starts with a set of parties it is going to corrupt and sticks to that set $\mathcal{C}$. In our setting, the adversary is allowed to corrupt a minority of the participants.

2.3.1 Real Model

In the real model there are n parties $P_1, \ldots, P_n$ running a protocol π and an adversary $\mathcal{A}$. Each party P_i starts with its private input $x_i^s \in \{0,1\}^*$ his public input $x_i^p \in \{0,1\}^*$ and random input $r_i \in \{0,1\}^*$. All parties have agreed upon some security parameter $k \in \mathbb{N}$. It is assumed that every two parties are connected via a *private channel*.

There is an adversary $\mathcal{A}$ that is t-limited, i.e., can corrupt up to t parties. The adversary $\mathcal{A}$ starts with some *auxiliary* input $z \in \{0,1\}^*$, random input $r_{\mathcal{A}} \in \{0,1\}^*$ and the set of

identities of corrupted parties $\mathcal{C} \subset \{1, \ldots, n\}$. Each party P_i, where $i \in \mathcal{C}$ will be controlled by $\mathcal{A}$.

The protocol is executed in interactive rounds. In each round the honest parties generate their messages. The adversary learns all messages that are addressed to corrupt parties, *before* it is going to generate and send the messages on behalf of the corrupt parties. This is called *rushing*. If the adversary is passive it generates messages on behalf of the corrupt parties as described by protocol π. But if $\mathcal{A}$ is active it will generate messages in an arbitrary way. Finally, the honest parties receive messages from the corrupt parties.

When the computation is finished, the parties and adversary generate locally their outputs as follows. The honest parties output whatever is specified by π. The corrupt parties output a special symbol $\perp$ and the adversary outputs its entire view. The view of $\mathcal{A}$ consists of its inputs $(z, r_{\mathcal{A}})$, all public inputs $(\mathbf{x}^p)$, the private and random inputs of the corrupt parties, and all messages send and received by the corrupt parties.

Let $\pi_{\mathcal{A}}(k, \mathbf{x}^s, \mathbf{x}^p, z, \mathbf{r}, r_{\mathbf{A}})_i$ denote the output of party P_i in the real model and, similarly, let $\pi_{\mathcal{A}}(k, \mathbf{x}^s, \mathbf{x}^p, z, \mathbf{r}, r_{\mathbf{A}})_{\mathcal{A}}$ denote the output of the adversary. Furthermore, let $\pi_{\mathcal{A}}(k, \mathbf{x}^s, \mathbf{x}^p, z, \mathbf{r}, r_{\mathcal{A}})$ denote the collection of all outputs from all parties and the adversary, and let $\pi_{\mathcal{A}}(k, \mathbf{x}^s, \mathbf{x}^p, z)$ denote the probability distribution of $\pi_{\mathcal{A}}(k, \mathbf{x}^s, \mathbf{x}^p, z, \mathbf{r}, r_{\mathcal{A}})$, where $\mathbf{r}$ and $r_{\mathcal{A}}$ are chosen uniformly at random. Finally, let $\pi_{\mathcal{A}}$ denote the distribution ensemble over all $k \in \mathbb{N}$ and $\mathbf{x}^s, \mathbf{x}^p, z \in \{0,1\}^*$.

2.3.2 Ideal Model

In the ideal model there are the n parties $P_1, \ldots, P_n$, a trusted party $\mathcal{T}$ and an adversary $\mathcal{S}$. Each party P_i has private input $x_i^s \in \{0,1\}^*$ and public input $x_i^p \in \{0,1\}^*$. All parties have agreed upon some security parameter $k \in \mathbb{N}$.

The trusted party $\mathcal{T}$ has an n-party function $f : \mathbb{N} \times (\{0,1\}^*)^{2n} \times \{0,1\}^* \to (\{0,1\}^*)^n$, where $f(k, \mathbf{x}^s, \mathbf{x}^p, r_f)_i$ denotes the i-th result of f on input security parameter k, private inputs $\mathbf{x}^s$, public inputs $\mathbf{x}^p$ and some random input r_f.

The adversary $\mathcal{S}$ starts with some auxiliary input $z \in \{0,1\}^*$, random input $r_{\mathcal{S}} \in \{0,1\}^*$ and the set of identities of corrupted parties $\mathcal{C} \subset \{1, \ldots, n\}$.

In the ideal model the computation proceeds as follows. First, the adversary $\mathcal{S}$ learns all private inputs of all corrupted parties P_i, where $i \in \mathcal{C}$. In addition it learns all public inputs $\mathbf{x}^p$ of all parties. If $\mathcal{S}$ is active it may modify the inputs (x_i^s, x_i^p) to (y_i^s, y_i^p). If $\mathcal{S}$ is passive then no substitution is made.

Secondly, all parties hand their inputs to the trusted party $\mathcal{T}$. Let $\mathbf{y}^s, \mathbf{y}^p$ denote the inputs $\mathcal{T}$ receives from all parties. Observe that $(y_i^s, y_i^p) = (x_i^s, x_i^p)$ if P_i is honest and $(\mathbf{y}^s, \mathbf{y}^p) = (\mathbf{x}^s, \mathbf{x}^p)$ if $\mathcal{S}$ is passive. Further $\mathcal{T}$ draws uniformly random r_f and sends $f(k, \mathbf{y}^s, \mathbf{y}^p, r_f)_i$ to party P_i.

Finally, all parties locally generate their output. Each honest P_i outputs $f(k, \mathbf{y}^s, \mathbf{y}^p, r_f)_i$, while each corrupt P_i outputs $\perp$. The adversary $\mathcal{S}$ outputs an arbitrary function of its current view. This view consists of its input, all public inputs, the private inputs and outputs of the corrupted parties.

Let $\mathcal{I}_{f,\mathcal{S}}(k, \mathbf{x}^s, \mathbf{x}^p, r_f, r_{\mathcal{S}})_i$ denote the output of party P_i in the ideal model and, similarly, let $\mathcal{I}_{f,\mathcal{S}}(k, \mathbf{x}^s, \mathbf{x}^p, r_f, r_{\mathcal{S}})_{\mathcal{S}}$ denote the output of the adversary. Furthermore, let $\mathcal{I}_{f,\mathcal{S}}(k, \mathbf{x}^s, \mathbf{x}^p, z, r_f, r_{\mathcal{S}})$ denote the collection of all outputs from all parties and the adversary, and let $\mathcal{I}_{f,\mathcal{S}}(k, \mathbf{x}^s, \mathbf{x}^p, z)$ denote the probability distribution of $\mathcal{I}_{f,\mathcal{S}}(k, \mathbf{x}^s, \mathbf{x}^p, z, r_f, r_{\mathcal{S}})$, where r_f and $r_{\mathcal{S}}$ are chosen uniformly at random. Finally, let $\mathcal{I}_{f,\mathcal{S}}$ denote the distribution

ensemble over all $k \in \mathbb{N}$ and $\mathbf{x}^s, \mathbf{x}^p, z \in \{0,1\}^*$.

Definition 2.10. *Let f be a function and π an n-party protocol. Then π is said to* t-securely *evaluate f if for any static t-limited adversary $\mathcal{A}$, there exists a static ideal adversary $\mathcal{S}$, having a runtime that is (expected) polynomial in the runtime of $\mathcal{A}$, such that*

$$\mathcal{I}_{f,\mathcal{S}} \stackrel{d}{=} \pi_{\mathcal{A}}.$$

Definition 2.10 provides *perfect* security. The definition can also be used to define *statistical* security or *computational* security, by replacing $\stackrel{d}{=}$ with $\stackrel{s}{=}$ or $\stackrel{c}{=}$ respectively.

2.3.3 Hybrid Model

To allow modular composition, a third model is introduced that allows certain subroutine calls to be replaced by calls to a trusted ideal party. More precisely, in the $(g_1, \ldots, g_\ell)$-hybrid model one considers a protocol π in which all parties have access to a trusted party $\mathcal{T}$ for computing the results of functions $g_1, \ldots, g_\ell$.

In the $(g_1, \ldots, g_\ell)$-hybrid model we have n parties $P_1, \ldots, P_n$ running protocol π, a trusted party $\mathcal{T}$ that acts as an oracle to the function $g_1, \ldots, g_\ell$ and a real life adversary $\mathcal{A}$. The protocol proceeds in interactive rounds as described in the real model. However, each time some g_i is evaluated one proceeds as in the ideal model.

We denote by $\pi^{(g_1,\ldots,g_\ell)}$ that protocol π is evaluated, where function calls of $g_1, \ldots, g_\ell$ are via a trusted party in the ideal model. Similar to $\pi_{\mathcal{A}}$ and $\mathcal{I}_{\mathcal{S}}$, we define $\pi_{\mathcal{A}}^{(g_1,\ldots,g_\ell)}$ to be the distribution ensemble over all inputs of the outputs of all parties and $\mathcal{A}$, where the random inputs are taken uniformly random.

Definition 2.11. *Let f be a function and π be an n-party protocol. Then $\pi^{(g_1,\ldots,g_\ell)}$ is said to t-securely evaluate f in the $(g_1, \ldots, g_\ell)$-hybrid model if for any static t-limited $(g_1, \ldots, g_\ell)$-hybrid adversary $\mathcal{A}$ there exist a static adversary $\mathcal{S}$ which has runtime that is polynomial in the runtime of $\mathcal{A}$ such that*

$$I_{f,\mathcal{S}} \stackrel{d}{=} \pi_{\mathcal{A}}^{(g_1,\ldots,g_\ell)}.$$

Again, Definition 2.11 provides *perfect* security. The definition can also be used to define *statistical* security or *computational* security, by replacing $\stackrel{d}{=}$ with $\stackrel{s}{=}$ or $\stackrel{c}{=}$ respectively.

In [Can00] it is shown that modular composition of protocols are security preserving; if $\rho_1, \ldots, \rho_\ell$ are n-party protocols that t-securely evaluate the n-party functions $g_1, \ldots, g_\ell$ respectively and if π t securely evaluates the n party function f in the $(g_1, \ldots, g_\ell)$-hybrid model, then π t-securely evaluates f.

Theorem 2.12 (Composition Theorem)**.** *Let $t < n$ and $\ell \in \mathbb{N}$ and let $g_1, \ldots, g_\ell$ and f be n-party functions. Let $\pi^{(g_1,\ldots,g_\ell)}$ be an n party protocol that t-securely evaluates f in the $(g_1, \ldots, g_\ell)$-hybrid model, where no more that one ideal evaluation call is made each round. Let $\rho_1, \ldots, \rho_\ell$ be n party protocols, where each ρ_i t-securely evaluates g_i. Let π be the protocol composed from $\pi^{(g_1,\ldots,g_\ell)}$, where each call to evaluate g_i is replaced by ρ_i. Then π t-securely evaluates f.*

This *Composition Theorem* also applies when security is statistical or computational, but one needs to be careful. For statistical security for example, suppose that $\Delta(\pi_{\mathcal{A}}^{(g_1,\ldots,g_\ell)}; \mathcal{I}_{f,\mathcal{S}}) =$

δ and $\Delta(\rho_{i,\mathcal{A}}; \mathcal{I}_{g_i,\mathcal{S}}) = \delta_i$. Suppose furthermore that π has ν_i subroutine calls to ρ_i. Then the total statistical difference $\Delta(\pi_{\mathcal{A}}; \mathcal{I}_{f,\mathcal{S}})$ is at most $\delta + \sum_{i=1}^{\ell} \nu_i \delta_i$.

In practice, Theorem 2.12 is applied as follows. To prove that π is secure, one proves firstly that $\rho_1, \ldots, \rho_\ell$ are secure. Let $\mathcal{S}_{g_i}$ denote the ideal model adversary constructed in those proofs. Secondly, one constructs $\mathcal{S}$, where each time ρ_i is run as a subroutine, by running the corresponding simulation $\mathcal{S}_{g_i}$. See [CDN01] as an example.

2.3.4 Multiparty Computation from Shamir Secret Sharing

We will show that Shamir Secret Sharing allows for secure multiparty computation in the model of [Can00]. Let f be an n party function and let C be the arithmetic circuit evaluating f.

We assume that in the first round, all parties share their private inputs using Shamir secret sharing. In the last round all results are opened using the recombination protocol for Shamir shares. For the sake of simplicity, we take f to be deterministic, having only private inputs of the parties and returning a single value, i.e., $f : (\{0,1\}^*)^n \to \{0,1\}^*$. Furthermore, we assume that all parties may learn the result $y = f(x_1, \ldots, x_n)$, where x_i is party P_i's private input. By using private opening gates as described in [CDN01] the following result can be extended to the case where each party has private output y_i.

Addition and subtraction of secrets: $[x \pm y]$ is locally computed by each party P_i from $[x]$ and $[y]$ by $[x \pm y]_i = [x]_i \pm [y]_i$.

Adding and subtraction of secret with a public constant: $[x \pm a]$ is locally computed by each party P_i by $[x \pm a]_i = [x]_i \pm a$.

Multiplication with a public constant: Multiplication of $[x]$ by a constant b is also done locally by each party P_i by $[bx] = b[x]_i$.

Multiplication of secrets: Computing shares of $[xy]$ given $[x]$ and $[y]$ requires an interactive protocol. Although $f(x)g(x) = (fg)(x)$ for any polynomials f and g, so each party could compute a share of xy by computing $[x]_i[y]_i$, this is not a correct share in the sense that if f and g are uniformly random t-degree polynomials then fg is a $2t$-degree polynomial and not uniformly random. The multiplication protocol of [BGW88] computes interactively a new random t-degree polynomial h, where $h(0) = xy$ if $2t < n$. Precisely, it performs the following steps, where we assume w.l.o.g. that the first $2t+1$ parties will do the interactive computations.

Protocol 2.9: $[c] \leftarrow \mathsf{Mul}([x], [y])$

1 **foreach party** $i = 1, \ldots, 2t+1$ **do**
2 $\quad m_i \leftarrow [x]_i[y]_i$;
3 $\quad [m_i] \leftarrow \mathsf{SShare}(m_i, t, n)$;
4 $[c] \leftarrow \sum_{i=1}^{2t+1} \left([m_i] \prod_{j=1, j\neq i}^{2t+1} \frac{-j}{i-j} \right)$;
5 **return** $[c]$

Theorem 2.13. *Let $t < n/2$. Let $f : (\mathbb{Z}_p)^n \to \mathbb{Z}_p$ be an n-party function. Let π be an n-party protocol evaluating circuit C consisting of arithmetic gates, i.e., addition, subtraction and multiplication gates, based on Shamir secret sharing. Suppose that C is such that in the first round all parties share their inputs and in the last round all parties learn their output by Shamir reconstruction. Then, π t-securely evaluates f.*

Proof. We need to give a simulator $\mathcal{S}$ in the ideal model that simulates transcripts and outputs of the parties and the view of the adversary. The simulator $\mathcal{S}$ is given the set $\mathcal{C}$ of identities of the corrupted parties of size t.

Recall that the model assumes a network with rushing, so the simulator $\mathcal{S}$ needs to act first every round before getting input from $\mathcal{A}$ on behalf of the corrupted parties.

In the first round, when all parties share their inputs, simulator $\mathcal{S}$ does the following.

- Pick a random value $r \in \mathbb{Z}_p$.
- For each honest party P_i, $i \notin \mathcal{C}$, share r using Shamir secret sharing scheme. Let p_i denote the polynomial that is used to share r on behalf of honest party P_i.
- For each honest party P_j record $[r]_j = p_i(j)$ for later use and for each corrupted party P_k, $k \in \mathcal{C}$, send $[r]_k = p_i(k)$ to the adversary $\mathcal{A}$.
- Receive shares $[x_j]_i$ from $\mathcal{A}$, where $i \notin C$ and $j \in \mathcal{C}$.
- Reconstruct x_j for each corrupted party using the Lagrange interpolation. This is possible since $\mathcal{A}$ is passive and $\mathcal{S}$ receives $n - t \geq n/2 \geq t + 1$ shares of each $[x_j]$.
- Send x_j for $j \in \mathcal{C}$ to the trusted party $\mathcal{T}$ to receive $y = f(x_1, \ldots, x_n)$.

In the next rounds the arithmetic gates are evaluated. We show what the simulator does depending on the gate.

Addition/Subtraction: To compute a consistent sharing for $[x+y]$, $\mathcal{S}$ takes the shares it recorded for the honest parties of $[x]$ and $[y]$. Then it computes and stores $[x \pm y]_i = [x]_i \pm [y]_i$ as the result for each honest party P_i. If the computation is done with a public value a then $\mathcal{S}$ sets $[x + a]_i$ to be equal to $[x]_i + a$.

Multiplication by a constant: To compute a consistent sharing for $[xb]$, $\mathcal{S}$ takes the shares it recorded for the honest parties of $[x]$. Then, it computes and stores $[xb]_i = [x]_i b$.

Multiplication: Simulator $\mathcal{S}$ just runs the multiplication protocol on behalf of the honest parties:

- Let $\mathcal{M}$ denote the set of parties that is assigned to perform the computation and let $\boldsymbol{\lambda}$ denote the corresponding length $2t + 1$ reconstruction vector. Then for honest P_i, where $i \in \mathcal{M}$ it takes its stored shares $[x]_i$ and $[y]_i$, and Shamir shares $m_i = [x]_i[y]_i$. Concretely, for each honest P_i , where $i \in \mathcal{M}$, $\mathcal{S}$ generates a uniformly random polynomial $h_i(x)$ restricted to $h_i(0) = m_i$ and sends $h_i(j)$ to $\mathcal{A}$ for each $j \in \mathcal{C}$ and stores $h_i(k)$ for each $k \notin \mathcal{C}$.
- Upon reception of all shares of $\mathcal{A}$ on behalf of each P_j, where $j \in \mathcal{C} \bigcap \mathcal{M}$, it computes and stores the resulting shares of the honest parties by $[xy]_i = \sum_{j \in \mathcal{M}} \lambda_j [m_j]_i$.

When opening the result y the parties are going to open $[y']$ in the last round. Observe that $\mathcal{S}$ has stored a consistent sharing of each input and output of each gate. It follows that the shares $\mathcal{S}$ has recorded are such that $\mathcal{S}$ could compute the shares the corrupting parties should have. This is necessary for the reconstruction part.

In the last round $\mathcal{S}$ needs to simulate reconstruction to y. However, since $\mathcal{S}$ does not know the real inputs of the honest parties and has simulated the inputs, most likely the value that corresponds to the shares that are opened will not be equal to y. The simulator $\mathcal{S}$ proceeds as follows to provide correct views:

- Let $\mathcal{D}$ be the size $t+1$ set of parties that open the result.
- If $\mathcal{D} \bigcap \mathcal{C} = \emptyset$, then $\mathcal{S}$ returns y obtained from $\mathcal{T}$ in the first round.
- Else, let $\boldsymbol{\lambda}$ denote the reconstruction vector and let $[y']_i$ be the shares $\mathcal{S}$ has on which the reconstruction will be done. From these shares $\mathcal{S}$ reconstructs y'. Then $\mathcal{S}$ picks $i \in \mathcal{D}$, where P_i is honest and sets $z_i = \lambda_i^{-1}(y - y') + [y']_i$. For every other honest P_j, where $j \in \mathcal{D}$, it sets $z_j = [y']_j$. $\mathcal{S}$ sends z_k to $\mathcal{A}$, where $k \in \mathcal{D}, k \notin \mathcal{C}$.

Next, we show that the views of $\mathcal{S}$ and $\mathcal{A}$ are indistinguishable.

In the simulation of the first round, the view of $\mathcal{A}$ consist of the inputs and all shares of each corrupt party. And with respect to the honest parties it has computed a random sharing of some random element on behalf of the hones parties. In the real protocol execution $\mathcal{A}$ sees all inputs and shares of the inputs of each corrupted party and t shares of the inputs of the honest parties. However, these t shares are perfectly indistinguishable to t uniformly randomly drawn numbers. Hence the views of $\mathcal{S}$ and $\mathcal{A}$ are indistinguishable.

Similarly, in the last round $\mathcal{S}$ sees nothing except for what it has learned in the past, while $\mathcal{A}$ learns y at this moment. Since having at most t shares, any opening to any value is equally likely. It follows that the simulated views of the last round is perfectly indistinguishable with the views of the last round of $\mathcal{A}$ in the real execution.

The simulated views of $\mathcal{S}$ during the circuit evaluation are by construction indistinguishable from the view of $\mathcal{A}$ corresponding real protocol executions. Indeed, $\mathcal{S}$ just follows the protocol in all steps. Moreover, since all shares are consistent, $\mathcal{S}$ can open each sharing to learn the shares held by the corrupt parties. □

Chapter 3

Linear Optimization

This chapter introduces linear programming with the focus on the implementation details. We provide an unified and rigorous description of the elementary simplex algorithms on which the secure variants of Chapter 5 are based.

In the first section we discuss, following [BT97] and [Lue73], basic definitions and theorems of linear programming leading to the simplex algorithm and to efficient validation of any solution. In addition, we address several issues with the simplex algorithm. For example, we show that the simplex algorithm by Dantzig [DT97] may not terminate, using [KM72], and how to enforce termination by using Bland's extension [Bla77]. Another important issue is that initialization of simplex is not trivial. Generally, it boils down to solving yet another linear program using the simplex algorithm, but where the linear program is such that initialization of the simplex algorithm is trivial.

The second section discusses how to implement the simplex iterations. While the simplex algorithm is defined over $\mathbb{Q}$, following [Ros05], we show how to modify the simplex algorithm so that all computations are over $\mathbb{Z}$. We provide a simple proof of the fact that these modifications are correct. We will also provide an upper bound on the size of the values that appear in these computations, which are needed to be able to initialize the cryptosystems used for secure linear programming. In addition, we describe three elementary implementations of the simplex algorithm: the original *large tableau* simplex algorithm, the *small tableau* or *condensed tableau* simplex and the popular *revised* simplex algorithm.

The third section describes methods to initialize the simplex iterations. We show two basic algorithms, the *two-phase* simplex algorithms and the algorithm for the *Big-M* method. Both algorithms add extra variables to the (*original*) linear program resulting in an *artificial linear program* on which any simplex algorithm can be easily initialized. The two-phase simplex algorithm first solves the artificial linear program and uses the optimal result to initialize the simplex algorithm to solve the original linear program. The big-M method on the other hand uses an extended implementation of the simplex algorithm to find the solution for the original linear program directly. We show the issues that arise when applying these techniques on the modified simplex algorithm that is defined over $\mathbb{Z}$.

3.1 Linear Programming

A linear programming problem is an optimization problem where the objective function is linear in the unknowns and the constraints are linear equalities and linear inequalities.

Any linear program can be written down in the following form [Lue73]:

$$\begin{array}{lccccccccc} \min & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n, & & \\ \text{subject to} & a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1, \\ & a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2, \\ & \vdots & & \vdots & & & & \vdots & & \vdots \\ & a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m, \\ \text{and} & x_1 \geq 0, & & x_2 \geq 0, & & \cdots, & & x_n \geq 0 & & \end{array}$$

where $x_1, \ldots, x_n$ are the unknowns.

Using vector notation, the linear program is written as

$$\begin{array}{rcl} \min & \mathbf{cx}, & \\ \text{subject to} \quad \mathbf{Ax} & = & \mathbf{b}, \\ \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{3.1}$$

where $\mathbf{x} \in \mathrm{R}^n$, $\mathbf{c} \in \mathrm{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. The vector $\mathbf{x}$ will be called a *solution* to the linear program. If $\mathbf{x}$ satisfies all constraints it will be called a *feasible solution*. The vector $\mathbf{x}^{\mathsf{opt}}$ denotes the feasible solution where $\mathbf{cx}$ is minimal.

In the remainder of this chapter we will call any linear program to be in *standard form* if it satisfies Eq. (3.1).

The theory of linear programming is closely related to convexity theory. Consider a linear program in standard form. The solutions to the constraints define a *convex polyhedron* K, i.e.,

$$K = \{\, \mathbf{x} \geq \mathbf{0} |\, \mathbf{Ax} = \mathbf{b} \}\,.$$

By linearity of the objective function and the convexity of K one can prove that the optimal value of the objective is attained at one of the *extreme points* of K, i.e., a vector in K that cannot be written as a convex combination of two other vectors in K [Lue73, BT97].

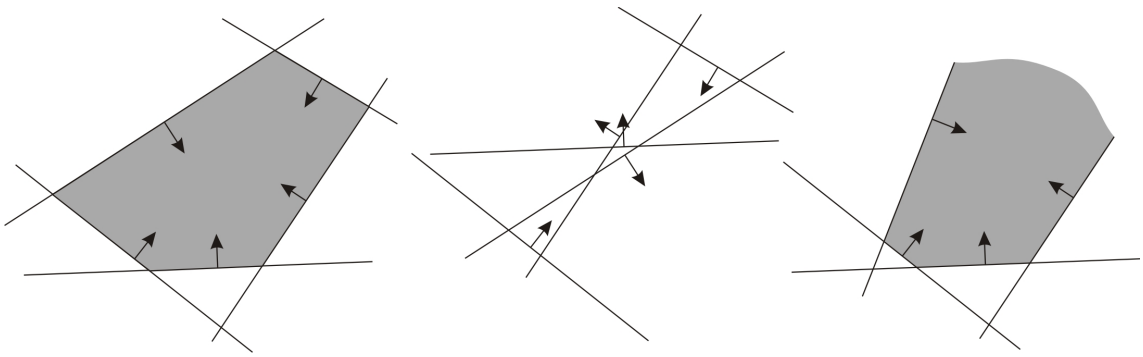


Figure 3.1: Three possible shapes of the polyhedra

Figure 3.1 shows three possible shapes of the polyhedron. From left to right they describe the following possibilities: if the polyhedron is closed and nonempty, then solutions exist, and the corresponding linear program will called *feasible*. But if the polyhedron is empty, then no feasible solution exists, and the corresponding linear program is called *infeasible*. Lastly, if the polyhedron is not bounded in a direction improving the objective of the corresponding linear program, then no bound exists on the optimum. In that case the linear program is called *unbounded*.

A naive method would be to compute $\mathbf{cx}$ for all extreme points in K and output the one minimizing $\mathbf{cx}$. This method is impractical as there may be exponentially many extreme points (see for example [KM72]).

The simplex algorithm, on the one hand, exploits the linearity and convexity properties of the LP to perform a more efficient search on the extreme points. It iteratively moves from one extreme point to an adjacent one improving the objective and stops if no such point exists. One can show that an extreme point that has no adjacent extreme point with better value for the objective, is optimal to the LP [BT97, Lue73]. Unfortunately, one can show that the simplex method may visit all extreme points. Hence exponentially many iterations are required [KM72].

Interior point methods, on the other hand, exploit the linearity and convexity properties of the LP to perform an efficient search in the interior of the polyhedron. They require only polynomially many iterations in the worst case. However, each iteration of any interior point method is expensive compared to the simplex algorithm.

We will discuss both methods in more detail in the next sections.

3.1.1 Simplex Algorithm

The simplex algorithm iteratively moves on the boundary of the polyhedron. Precisely, in each iteration, the simplex algorithm moves from one extreme point (*vertex*) to an adjacent vertex of the polyhedron that has improved costs. If no such vertex exists then the current vertex corresponds to an optimal solution.

First, we show how to define vertices of the polyhedra. Second, we show how to define directions to move between adjacent vertices. Then we will prove three basic theorems needed to present the basic simplex algorithm.

A solution $\mathbf{x}$ is vertex of the corresponding polyhedron if and only if it is a *basic feasible solution* to the linear program [Lue73, BT97]. For any matrix $\mathbf{V} \in \mathbb{R}^{p\times n}$ we write

$$\mathbf{V_s} = (\mathbf{V}_{s_i} \dots \mathbf{V}_{s_m}),$$

where $\mathbf{s} = (s_1, \dots, s_m) \in \{1, \dots, n\}^m$. We will write $s \in \mathbf{s}$ if $s = s_j$ for some j.

Definition 3.1. *Suppose that a linear program is given in standard form. Let* $\mathbf{s} = (s_1, \dots, s_m) \in \{1, \dots, n\}^m$ *and*

$$\mathbf{B} = \mathbf{A_s} = (\mathbf{A}_{s_1} \dots \mathbf{A}_{s_m}).$$

Then, $\mathbf{s}$ *is called a* basis *if* $\mathbf{B}$ *is invertible. If* $\mathbf{s}$ *is a basis then* $\mathbf{B}$ *is called a* basis matrix. *The tuple* $\mathbf{u} = (u_1, \dots, u_{n-m}) \in \{1, \dots, n\}^{n-m}$ *is called a* co-basis *if no* $u_i \in \mathbf{s}$.

For any length n *vector* $\mathbf{v}$, v_i *is called* basic *if* $i \in \mathbf{s}$ *and* co-basic *otherwise.*

The vector $\mathbf{x} \in \mathbb{R}^n$ *is called a* solution *if* $\mathbf{Ax} = \mathbf{b}$. *If, furthermore,* $\mathbf{x} \geq \mathbf{0}$ *it is called a feasible solution. Let* $\mathbf{y} = \mathbf{B}^{-1}\mathbf{b}$. *A solution* $\mathbf{x}$ *is called* basic solution with respect to basis $\mathbf{s}$ *if*

$$x_i = \begin{cases} y_j, & \text{if } i = s_j, \\ 0, & \text{if } i \in \mathbf{u}, \end{cases}$$

in other words, if

$$\mathbf{x_s} = \mathbf{B}^{-1}\mathbf{b},$$
$$\mathbf{x_u} = \mathbf{0}.$$

If $\mathbf{y} \geq \mathbf{0}$ *then* $\mathbf{x}$ *is called* basic feasible solution with respect to basis $\mathbf{s}$.

Remark 3.2. Observe that if the rank of $\mathbf{A}$ is less than m, then any m columns are linearly dependent. It follows that no basic feasible solution exists. Therefore, only linear programs are considered, where $n \geq m$ and the rows of $\mathbf{A}$ are linearly independent[1]. ◇

The simplex algorithm is based on the following theorem, from which follows that an optimal solution, if it exists, is always in an extreme point of the corresponding polyhedron.

Theorem 3.3 (Fundamental Theorem of Linear Programming)**.** *Consider an LP in standard form, where* $\mathbf{A}$ *is an* $m \times n$ *matrix of rank* m*. Then,*

1. *if there is a feasible solution, then there is a basic feasible solution, and*
2. *if there is an optimal feasible solution, then there is an optimal basic feasible solution.*

Definition 3.4. *Let* $\mathbf{x}$ *be a basic feasible solution with respect to basis* $\mathbf{s}$ *and* $\mathbf{y}$ *be a basic feasible solution with respect to basis* $\mathbf{s}'$*. If* $s_i = s'_i$ *for all except one* $i \in \{1, \ldots, m\}$*, then* $\mathbf{x}$ *and* $\mathbf{y}$ *are called* adjacent*.*

Definition 3.5. *Consider an LP in standard form. Suppose that* $\mathbf{x}$ *is a basic solution with respect to basis* $\mathbf{s}$ *and co-basis* $\mathbf{u}$*. A length* n *vector* $\mathbf{d}$ *is called a*

- valid direction *at* $\mathbf{x}$ *if for all* $\theta > 0$*, the equality constraints* $\mathbf{A}(\mathbf{x} + \theta\mathbf{d}) = \mathbf{b}$ *are satisfied.*
- feasible direction *at* $\mathbf{x}$ *if* $\mathbf{x} + \theta\mathbf{d}$ *is a feasible solution for some* $\theta > 0$*.*
- ℓ-th basic direction at $\mathbf{x}$ *if* $d_j = 0$ *for all* $j \in \mathbf{u}$*,* $j \neq \ell$*, where* $\ell \notin \mathbf{s}$ *and if* $d_\ell = 1$*. The* ℓ*-th basic direction is often denoted by* $\mathbf{d}^\ell$*.*

Lemma 3.6. *Suppose that* $\mathbf{x}$ *is a basic feasible solution to an LP in standard form with respect to basis* $\mathbf{s}$*. Let* $\mathbf{B}$ *be the basis matrix. Then,*

(i) $\mathbf{Ad} = \mathbf{0}$ *for any valid direction* $\mathbf{d}$*,*

(ii) any feasible direction at $\mathbf{x}$ *is a valid direction at* $\mathbf{x}$*, and*

(iii) the ℓ*-th basic feasible direction* $\mathbf{d}^\ell$ *is unique and satisfies*

$$\mathbf{d}^\ell_{\mathbf{s}} = -\mathbf{B}^{-1}\mathbf{A}_\ell. \tag{3.2}$$

Proof. (i) Let $\mathbf{d}$ be a valid direction at $\mathbf{x}$. Then

$$\mathbf{A}\left(\mathbf{x} + \theta\mathbf{d}\right) = \mathbf{b}, \tag{3.3}$$

for all $\theta > 0$. Since $\mathbf{x}$ is feasible it follows that $\mathbf{Ax} = \mathbf{b}$. Hence

$$\theta\mathbf{Ad} = \mathbf{0},$$

for all $\theta > 0$. Hence $\mathbf{Ad} = \mathbf{0}$.

(ii) Let $\mathbf{d}$ be a basic feasible direction at $\mathbf{x}$. Since $\mathbf{x} + \theta\mathbf{d}$ is a feasible solution for some $\theta > 0$ Eq. (3.3) holds for some $\theta > 0$. Again from the feasibility of $\mathbf{x}$ and θ being nonzero it follows that $\mathbf{Ad} = \mathbf{0}$ so Eq. (3.3) holds for all $\theta > 0$. Hence $\mathbf{d}$ is a valid direction.

[1] If $n = m$ and $\text{rank}(A) = m$, then there exists only one basic solution, so there is nothing to optimize.

(iii) Let $\mathbf{d}$ be an ℓ-th basic feasible direction at $\mathbf{x}$. Since $\mathbf{d}$ is valid we have by (i) that $\mathbf{Ad} = \mathbf{0}$. Since all nonbasic entries in $\mathbf{d}$ are equal to zero, except for d_ℓ which is equal to one,

$$\mathbf{Ad} = \mathbf{Bd_s} + \mathbf{A}_\ell.$$

Hence Eq. (3.2) follows and $\mathbf{d_s}$ is uniquely determined. By Definition 3.5 it follows that all entries of $\mathbf{d}$ are uniquely determined. □

The simplex algorithm iteratively moves from one basic feasible solution to an adjacent one that improves the objective until no such adjacent basic feasible solution exists. In other words, every iteration, the simplex algorithm being at solution $\mathbf{x}$, searches a valid basic direction $\mathbf{d}^\ell$ such that

(i) $\mathbf{d}^\ell$ *is cost-improving:* $\mathbf{cd}^\ell < 0$,

(ii) $\mathbf{d}^\ell$ *is a feasible direction at* $\mathbf{x}$*:* for some $\theta > 0$, $\mathbf{x} + \theta\mathbf{d}^\ell$ is feasible,

(iii) $\mathbf{d}^\ell$ *leads to an adjacent vertex:* $\mathbf{x}' = \mathbf{x} + \theta\mathbf{d}^\ell$ is basic feasible for some $\theta > 0$.

The following three theorems summarize the ideas behind the simplex algorithm. For more details about the background of the simplex algorithm see [BT97, Lue73, DT97].

Theorem 3.7 ([BT97])**.** *Suppose that* $\mathbf{x}$ *is a basic feasible solution to an LP in standard form with respect to basis* $\mathbf{s}$ *and co-basis* $\mathbf{u}$*. If no cost-improving valid basic direction exists at* $\mathbf{x}$*, then* $\mathbf{x}$ *is a* optimal solution*.*

Proof. By Lemma 3.6 any basic feasible direction $\mathbf{d}^\ell$ satisfies

$$\mathbf{d}^\ell_\mathbf{s} = -\mathbf{B}^{-1}\mathbf{A}_\ell.$$

For each ℓ-th basic feasible direction at $\mathbf{x}$ the changes in costs are computed by

$$\overline{c}_\ell = \mathbf{cd}^\ell = \mathbf{c_s}\mathbf{d}^\ell_\mathbf{s} + c_\ell d_\ell = c_\ell - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}_\ell. \tag{3.4}$$

Since $\mathbf{d}_\ell$ is not cost improving it follows that $\overline{c}_\ell \geq 0$.

Suppose that $\mathbf{y} \neq \mathbf{x}$ is a feasible solution to the linear program. Let $\mathbf{v} = \mathbf{y} - \mathbf{x}$. Since both $\mathbf{x}$ and $\mathbf{y}$ are feasible it follows that $\mathbf{Ax} = \mathbf{Ay} = \mathbf{b}$ and, therefore, $\mathbf{Av} = \mathbf{0}$. Hence

$$\mathbf{Av} = \mathbf{Bv_s} + \sum_{i \in \mathbf{u}} \mathbf{A}_i v_i = \mathbf{0}.$$

It follows that

$$\mathbf{v_s} = -\sum_{i \in \mathbf{u}} \mathbf{B}^{-1}\mathbf{A}_i v_i$$

and the cost difference between $\mathbf{x}$ and $\mathbf{y}$ is given by

$$\mathbf{c}(\mathbf{y} - \mathbf{x}) = \mathbf{cv} = \mathbf{c_u v_u} + \mathbf{c_s v_s} = \sum_{i \in \mathbf{u}} \left(c_i - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}_i\right) v_i = \sum_{i \in \mathbf{u}} \overline{c}_i v_i \geq 0,$$

since $\mathbf{v_u} \geq \mathbf{0}$. Indeed, $y_i \geq 0$ and $x_i = 0$ for all $i \in \mathbf{u}$ by the feasibility of $\mathbf{y}$.

Hence $\mathbf{cx} \leq \mathbf{cy}$ and, therefore, $\mathbf{x}$ is optimal. □

Notice that Eq. (3.4) implies that $\mathbf{d}^\ell$ is cost-improving if and only if $\overline{c}_\ell < 0$. The vector $\overline{\mathbf{c}}$ is often called the *cost-reduced vector*, which is defined as follows.

Definition 3.8. *Consider a linear program in standard form. Let* $\mathbf{s}$ *be a basis. Then,*

$$\overline{\mathbf{c}} = \mathbf{c} - \mathbf{c_s}\mathbf{A_s}^{-1}\mathbf{A}$$

is called the cost-reduced vector with respect to basis $\mathbf{s}$.

Lemma 3.9. *Consider a linear program in standard form. Let* $\mathbf{s}$ *be a basis. Then any basic feasible direction* $\mathbf{d}^i$ *is cost-improving if and only if the cost-reduced vector* $\overline{\mathbf{c}}$ *with respect to* $\mathbf{s}$ *satisfies* $\overline{c}_i < 0$. *Furthermore,* $\overline{c}_j = 0$ *for any* $j \in \mathbf{s}$.

Proof. Let $\mathbf{B} = \mathbf{A_s}$ be the basis matrix, $\overline{\mathbf{c}}$ be the cost-reduced vector with respect to basis $\mathbf{s}$, and $\mathbf{d}^i$ be the i-th basic feasible direction with respect to basis $\mathbf{s}$.

Suppose that $\mathbf{d}^i$ is cost improving. Then by Eq. (3.2)

$$\overline{c}_i = c_i - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}_i = c_i d^i_i + \mathbf{c_s}\mathbf{d}^i_{\mathbf{s}} = \mathbf{c}\mathbf{d}^i.$$

Hence $\mathbf{d}^i$ is cost improving if and only if $\overline{c}_i < 0$.

Next, let $j \in \mathbf{s}$. Then $s_k = j$ for some k and

$$\overline{c}_j = c_j - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}_j = c_{s_k} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{B}_k = c_{s_k} - \mathbf{c}_s\mathbf{e}_k = 0.$$

□

Theorem 3.10. *Suppose that* $\mathbf{x}$ *is a feasible solution to an LP in standard form. If* $\mathbf{d}$ *is a cost-improving feasible direction at* $\mathbf{x}$ *having nonnegative entries only, then the LP is unbounded.*

Proof. Suppose that $\mathbf{d}$ is a cost-improving feasible direction at $\mathbf{x}$ having nonnegative entries only. Being a feasible direction it will also be a valid direction. Hence for all $\theta > 0$ the equality constraints are satisfied. Moreover, from $\mathbf{d} \geq \mathbf{0}$ it follows that $\mathbf{x} + \theta\mathbf{d} \geq \mathbf{0}$ for all $\theta > 0$. Hence for all $\theta > 0$ the solution $\mathbf{x} + \theta\mathbf{d}$ is feasible.

Since $\mathbf{d}$ is cost-improving

$$\mathbf{c}\mathbf{d} < 0,$$

and thus

$$\mathbf{c}(\mathbf{x} + \theta\mathbf{d}) = \mathbf{c}\mathbf{x} + \theta\mathbf{c}\mathbf{d}$$

is unbounded since θ is unbounded. □

Theorem 3.11. *Suppose that* $\mathbf{x}$ *is a basic feasible solution to an LP in standard form corresponding to basis* $\mathbf{s}$. *Let* $\mathbf{d}$ *be the* ℓ*-th cost-improving valid basic direction at* $\mathbf{x}$ *with at least one negative entry. Then for any* $0 \leq \theta \leq \theta^*$ *the solution* $\mathbf{x}' = \mathbf{x} + \theta\mathbf{d}$ *is feasible where*

$$\theta^* = \min\left\{ -\frac{x_i}{d_i} \,\middle|\, d_i < 0 \text{ and } i \in \mathbf{s} \right\}. \tag{3.5}$$

Moreover, if $\theta = \theta^*$ *then* $\mathbf{x}'$ *is basic feasible.*

Proof. Since $\mathbf{d}$ is a valid direction and $\mathbf{x}$ a solution it follows for all $\theta \geq 0$ that $\mathbf{x}'$ is a solution.

Let $0 \leq \theta < \theta^*$. Suppose that j and k are such that $\theta^* = -x_j/d_j$ and $j = s_k$. Then, all co-basic entries of $\mathbf{x}'$ satisfy $x'_i = x_i + \theta d_i = 0$, if $i \neq \ell$, and $x'_\ell = \theta$. All basic entries of $\mathbf{x}'$ satisfy $x'_i \geq 0$, if $d_i \geq 0$, but also if $d_i < 0$:

$$x'_i = x_i + \theta d_i \geq x_i - \frac{x_j}{d_j} d_i \geq x_i - \frac{x_i}{d_i} d_i = 0.$$

Hence $\mathbf{x}'$ is feasible.

If $\theta = \theta^*$ then $x'_j = 0$ and $x'_\ell = \theta \geq 0$ then $\mathbf{x}$' is basic with basis

$$\mathbf{s}' = (s_1, \ldots, s_{k-1}, \ell, s_{k+1}, \ldots, s_m).$$

□

Remark 3.12. Note that θ^* can be equal to zero if for some j both $d_j < 0$ and $x_j = 0$. In such a case $\mathbf{d}$ is called a *degenerate direction* and changing the basis does not result in a new solution. But since the valid basic directions depend on the current basis (see Eq. (3.2)) changing the basis may lead to new directions where $\theta^* > 0$. ◇

Definition 3.13. *Consider a linear program in standard form. Let* $\mathbf{s}$ *be a basis corresponding to basic feasible solution* $\mathbf{x}$*. Suppose that* $\mathbf{d}$ *is a basic feasible cost-improving direction, where* $d_j < 0$*. If* $x_j = 0$*, then* $\mathbf{d}$ *is called a* degenerate direction at $\mathbf{x}$ *and the linear program is called* degenerate*.*

In conclusion, given a basic feasible solution $\mathbf{x}$ with respect to basis $\mathbf{s}$ the simplex method performs the following steps during each iteration.

Basic Simplex.

Entering-Variable: Let $\mathbf{s}$ be a basis and $\mathbf{B} = \mathbf{A}_\mathbf{s}$ the corresponding basis matrix. Pick a cost-improving basic feasible direction $\mathbf{d}$, or equivalently (Lemma 3.9), pick ℓ such that $\overline{c}_\ell < 0$. From the proof of Theorem 3.11 it follows that x_ℓ will become basic. If no such ℓ exist, then output current solution being optimal.

Leaving-Variable: Compute θ from Eq. (3.5) and pick k that minimizes θ. Again by Theorem 3.11 it follows that x_k will become co-basic. If no such k exist then stop while reporting that the LP is unbounded.

Update the Basis Replace s_k by ℓ in $\mathbf{s}$.

In the first two steps of the algorithm there is some freedom in how to implement the algorithm: if multiple entries of $\overline{\mathbf{c}}$ are negative then one can choose freely among them, and if there are multiple values for k that minimize θ then again one is free to choose among them. A rule restricting the choice of ℓ and k is called a *pivoting rule*.

The performance due to pivoting rules strongly depends on the problem instance itself. To illustrate this, we describe two well known pivoting rules: Dantzig's original pivoting rule and Bland's pivoting rule [Bla77].

Let basic feasible solution $\mathbf{x}$ and basis $\mathbf{s}$ be given at the start of an iteration of the simplex algorithm. Consider the following restrictions on the choice of the variable to become basic and the variable to become co-basic.

Dantzig's Original Pivoting Rule: With respect to the entering variable, choose ℓ such that

$$\ell = \operatorname{argmin} \{\overline{c_i} | \overline{c_i} < 0\}, \tag{3.6}$$

where $\operatorname{argmin}(\mathbf{x}) = i$ if and only if $x_i = \min(\mathbf{x})$.

Bland's Pivoting Rule [Bla77]: With respect to the entering variable, choose ℓ such that

$$\ell = \min \{i | \overline{c}_i < 0\}. \tag{3.7}$$

Let $\mathbf{d}^\ell$ be the corresponding basic feasible direction. With respect to the leaving variable, choose k such that

$$k = \operatorname{argmin} \left\{ s_i \,\middle|\, d_i^\ell < 0 \text{ and } -\frac{x_{s_i}}{d_{s_i}} = \min \left\{ -\frac{x_{s_j}}{d_{s_j}} \,\middle|\, d_{s_j}^\ell < 0 \right\} \right\}. \tag{3.8}$$

Remark 3.14. One easily verifies that the problem instances given by [KM72] require exponentially many iterations if Dantzig's original pivoting rule is applied, where Bland's rule requires just one iteration.

More importantly, Beale provides in [Bea55] an example of a linear program on which the simplex algorithm with Dantzig's original pivoting rule will never terminate. This is due to the fact that at some stage Dantzig's pivoting rule selects only degenerate basic feasible directions in such a way that the corresponding basis updates yields a sequence of bases that is repeated over and over. This is called *cycling*, see Definition 3.15. ◇

Definition 3.15. *Suppose that the simplex algorithm is applied to solve a linear program in standard form. We say that the simplex algorithm* cycles *between bases* $\mathbf{s}^1, \ldots, \mathbf{s}^p$ *if* $\mathbf{s}^i$ *will be updated by the simplex algorithm to* $\mathbf{s}^{i+1}$*, for all* $i = 1, \ldots, p$*, where* $\mathbf{s}^{p+1} = \mathbf{s}^1$*.*

Bland's pivoting rule is designed to be a very simple rule so that cycling cannot occur [Bla77]. It uses the fact that the simplex can cycle only if the linear program is degenerate.

Lemma 3.16. *Consider a linear program in standard form. If the simplex algorithm cycles, then the linear program is degenerate.*

Proof. Suppose that the simplex algorithm cycles between the bases $\mathbf{s}^1, \ldots, \mathbf{s}^p$. Let $\mathbf{x}^i$ be the basic feasible solution corresponding to basis $\mathbf{s}^i$. Observe that the costs should remain constant during the cycle. Indeed by construction of the simplex algorithm $\mathbf{cx}^1 \leq \cdots \leq \mathbf{cx}^p \leq \mathbf{cx}^{p+1} = \mathbf{cx}^1$.

Next, suppose that $\mathbf{d}$ is a cost improving basic feasible direction such that $\mathbf{x}^{i+1} = \mathbf{x}^i + \theta\mathbf{d}$ for some positive θ. Suppose furthermore that $\mathbf{s}^{i+1}$ is obtained from $\mathbf{s}^i$ by replacing s_k^i by ℓ. Since $\mathbf{d}$ is cost-improving $\mathbf{cd} < 0$. By $\mathbf{cx}^i = \mathbf{cx}^{i+1} = \mathbf{c}(\mathbf{x}^i + \theta\mathbf{d})$ it follows that $\theta = 0$.

Since $\mathbf{d}$ is a basic feasible direction and ℓ enters the basis replacing s_k^i, $d_\ell = 1$ and $d_{s_k} < 0$. Furthermore, from $\theta = 0$ we have by Eq. (3.5) that $x_{s_k} = 0$. □

Theorem 3.17 (Bland's anti-cycling rule:)**.** *Consider a linear program in standard form. The simplex algorithm using Bland's pivoting rule will not cycle.*

Proof. Suppose on the contrary, that the simplex algorithm under Bland's pivoting rule cycles given a linear program in standard form. Suppose that the simplex algorithm cycles successively between the bases $\mathbf{s}^1, \ldots, \mathbf{s}^p$ with corresponding co-bases $\mathbf{u}^1, \ldots, \mathbf{u}^p$. Let $\mathcal{T}$ be the set of indices that leave and enter the basis at some iteration in the cycle, i.e.,

$$\mathcal{T} = \{i \mid \exists (j,k) : i \in s_j \wedge i \in u_k\}$$

Next, suppose that $q = \max \mathcal{T}$. We will show that due to cycling, at some iteration q should enter the basis and at some other iteration q should leave the basis, resulting in a contradiction to the choice of the pivot element.

Suppose that q enters basis $\mathbf{s}^i$ and leaves basis $\mathbf{s}^j$, where $i, j \in \{1, \ldots, p\}$ and $i \neq j$. Let $\overline{\mathbf{c}}$ be the cost-reduced vector with respect to basis $\mathbf{s}^i$ and let $\overline{\mathbf{c}'}$ be the cost-reduced vector with respect to basis $\mathbf{s}^j$. Since q enters the basis $\mathbf{s}^i$ by Bland's pivoting rule, $\overline{c}_j \geq 0$, for all $j < q$, and $\overline{c}_q < 0$.

Suppose that basis $\mathbf{s}^{j+1}$ is obtained from $\mathbf{s}^j$ by replacing $s^j_r = q$ with t. Hence $t \in \mathcal{T}$ and $t \neq q$. Let $\mathbf{d}^t$ be the basic feasible direction with respect to basis $\mathbf{s}^j$. By Lemma 3.6 $\mathbf{A}\mathbf{d}^t = \mathbf{0}$ and $\mathbf{d}^t_{\mathbf{s}^j} = -\mathbf{A}^{-1}_{\mathbf{s}^j}\mathbf{A}_t$. Since $\overline{\mathbf{c}}$ is a cost-reduced vector with respect to basis $\mathbf{s}^i$ and $\mathbf{d}^t$ the t-th basic feasible direction with respect to basis $\mathbf{s}^j$, we have

$$\overline{\mathbf{c}}\mathbf{d}^t = \left(\mathbf{c} - \mathbf{c}_{\mathbf{s}^i}\mathbf{A}^{-1}_{\mathbf{s}_i}\mathbf{A}\right)\mathbf{d}^t = \mathbf{c}\mathbf{d}^t = c_t - \mathbf{c}_{\mathbf{s}^j}\mathbf{A}^{-1}_{\mathbf{s}_j}\mathbf{A}_t = \overline{c'}_t < 0.$$

Hence there should be a k such that $\overline{c}_k d^t_k < 0$. Since $\overline{c}_k \neq 0$ it follows from Lemma 3.9 that k is not in the basis $\mathbf{s}^i$. By $d^t_k \neq 0$ then either k is in the basis $\mathbf{s}^j$ or $k = t$. If $k = t \neq q$, then $\overline{c}_k d^t_k < 0$ and $d^t_t = 1$ implies that $\overline{c}_k = \overline{c}_t < 0$. But since $t \in \mathcal{T}$ and $t \neq q$ implies $t < q$, which is a contradiction to the fact that $\overline{c}_t \geq 0$. So k is in the basis $\mathbf{s}^j$. And hence $k \in \mathcal{T}$, since it is not in the basis $\mathbf{s}^i$.

Since $k \in \mathcal{T}$ we have by Lemma 3.9 that $x_k = 0$. Furthermore, $\overline{c}_q < 0$ and $d^t_{s_r} = d^t_q < 0$ so $\overline{c}_q d^t_q > 0$, but $\overline{c}_k d^t_k < 0$. Hence $k \neq q$. By the choice of q it follows that $k < q$, contradicting the fact that q leaves basis $\mathbf{s}^j$. □

3.1.2 Interior Point Methods

The interior point methods are theoretically more efficient than the simplex methods, since their worst case running time is polynomial. In practice, however, it is not all clear which performs best. We will briefly discuss the main ideas behind the interior point methods and show a simple example.

The interior point methods iteratively move between feasible solutions $\mathbf{x}$, where $\mathbf{x} > 0$. The idea is that within the interior of the polyhedron one has more freedom in choosing a feasible direction than on the boundary. In particular one can move into the direction where the cost improves the most, i.e., $-\mathbf{c}$. This direction is also called the direction of *steepest descent*. If one is at the center of the polyhedron, then moving into this direction will typically result in significant progress.

However, when $\mathbf{x}$ is not at all in the center of the polyhedron, then moving into the direction of steepest descent typically results in moving towards the boundary. Being close to the boundary limits the choice of the next direction, limiting progression to the optimum.

This observation is exploited in Karmarkar's method [Kar84] and Dikin's method [Dik74]. Their approach is to transform the polyhedron each iteration so that the transformed solution is in the center of the polyhedron and makes a significant improvement on the

transformed costs by going into the direction of steepest descent. If the improvement of the costs in the original program is very small, one can show that one is close to the optimum.

The claims by Karmarkar of his method being faster than the simplex methods stimulated new improvements to the simplex methods but also the development of numerous alternative interior point methods. For example, instead of going into the direction of steepest descent leading to costly transformations, one could define a *central path*, i.e., a (typically nonlinear) path through the center of the polyhedron that hits the boundary in the optimal solution. Being close to the central path one can make a significant improvement towards the solution by going in the direction of the path. In those methods, in every iteration, one tries to decide whether the current solution is close to the central path. If so, the direction of the path is followed, otherwise, one tries to move to a new solution close to the central path. For more details about these *path following methods* we refer to [NW99, Chapter 14].

The following *primal affine* algorithm illustrates an interior point method. It is Dikin's Method [Dik74] based on the description in [DT97]. Let $\mathbf{x}$ be an interior feasible point. We note that $\mathbf{A}$ has again full row rank.

Algorithm 3.1 (Dikin's Primal Affine Method)**.**

Centering First, the LP is transformed to an equivalent LP by letting $\mathbf{x}_t = \mathbf{D}^{-1}\mathbf{x} = \mathbf{1}$, where

$$\mathbf{D} = \begin{pmatrix} x_1 & \cdots & 0 \\ & \ddots & \\ 0 & \cdots & x_n \end{pmatrix}.$$

Note that $\mathbf{D}$ has an inverse since $\mathbf{x}$ is an interior point, i.e., $\mathbf{x} > \mathbf{0}$. With $\mathbf{A}_t = \mathbf{AD}$ and $\mathbf{c}_t = \mathbf{Dc}$, the transformed equivalent LP becomes

$$\begin{array}{lrcl} \min & \mathbf{c}_t\mathbf{x}, & & \\ \text{subject to} & \mathbf{A}_t\mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}. \end{array}$$

The current solution is $\mathbf{x}_t = \mathbf{1}$.

Compute direction of steepest descent: The direction of steepest descent is $-\mathbf{c}_t$, but since most likely $\mathbf{A}_t\mathbf{c}_t \neq \mathbf{0}$ it will not be a valid direction and, therefore, it will not be a feasible direction (cf. Lemma 3.6). Its projection onto the null space of $\mathbf{A}_t$ will result in a feasible direction of steepest descent. Let

$$\mathbf{P}_t = \mathbf{I} - {\mathbf{A}_t}^T(\mathbf{A}_t{\mathbf{A}_t}^T)^{-1}\mathbf{A}_t$$

be the $n \times n$ projection matrix onto the null space of $\mathbf{A}_t$, which can be computed since $\mathbf{A}_t$ has full row rank. Compute

$$\mathbf{d} = -\mathbf{P}_t\mathbf{c}_t.$$

Move to the new interior point Compute the new solution

$$\mathbf{x}'_t = \mathbf{e} + \frac{\alpha}{\theta}\mathbf{d},$$

where

$$\theta = -\min\{d_j | j \in \{1, \ldots, n\}\}$$

and $0 < \alpha < 1$. Hence $\mathbf{x}'_t > \mathbf{0}$ is an interior feasible point. If $\theta < 0$, then $\mathbf{d}$ has no negative entries. Hence, by Theorem 3.10 it follows that the LP is unbounded. Compute the corresponding solution of the original LP by $\mathbf{x}' = \mathbf{D}\mathbf{x}'_t$.

Check termination condition Terminate if $\mathbf{x} \approx \mathbf{x}'$, where some stopping criterium is defined.

Remark 3.18. The computational expensive part is the computation of the direction of steepest descent. One way of doing this step more efficiently is by computing $\mathbf{d}$ directly using a Cholesky decomposition or a QR decomposition as follows. Write

$$\mathbf{d} = -\mathbf{c}_t - \mathbf{A}_t^T \mathbf{v},$$

where $\mathbf{v}$ is the solution to

$$\mathbf{A}_t \mathbf{A}_t^T \mathbf{v} = -\mathbf{A}_t \mathbf{c}_t. \tag{3.9}$$

Hence,

$$\mathbf{v} = -(\mathbf{A}_t \mathbf{A}_t^T)^{-1} \mathbf{A}_t \mathbf{c}_t$$

and

$$\mathbf{d} = \left(\mathbf{I} - \mathbf{A}_t^T (\mathbf{A}_t \mathbf{A}_t^T)^{-1} \mathbf{A}_t\right)(-\mathbf{c}_t) = -\mathbf{P}_t \mathbf{c}_t.$$

To solve Eq. (3.9) efficiently, one computes $\mathbf{y} = -\mathbf{A}_t \mathbf{c}_t$ and decomposes $\mathbf{A}_t \mathbf{A}_t^T$ for example as $\mathbf{L}\mathbf{L}^T$ (Cholesky decomposition), where $\mathbf{L}$ is a lower triangular matrix. If $\mathbf{A}$ has rank m and $\mathbf{x} > \mathbf{0}$ then $\mathbf{A}_t \mathbf{A}_t^T$ is a symmetric positive definite matrix and, therefore, the Cholesky decomposition is applicable. One solves

$$\mathbf{L}\mathbf{u} = \mathbf{y}$$

and

$$\mathbf{L}^T \mathbf{v} = \mathbf{u}$$

to find $\mathbf{v}$. ◇

The most recent interior point methods are based on path following techniques and use nonlinear optimization techniques such as the Newton's method and the Barrier method. The hardest computational part of these algorithms is solving one or more systems of linear equations like in the given example. While the simplex algorithm is exact in principle, the interior point methods reach the optimal solution in the limit. On the other hand, the interior point methods are applicable to more general optimization problems such as semidefinite programming.

With respect to performance it is hard to decide whether the interior point methods will do better than the simplex methods [Mur05, Mil00, DT97, DT03]. Furthermore, since the optimum is on the boundary of the polyhedron, the interior point methods will reach the optimum only in its limit [NW99, DT97].

Due to simplicity, exact computations, and practical performance we will focus on the simplex algorithm.

3.1.3 Initial Feasible Solution

Both the simplex methods and the interior point methods move iteratively between feasible points in the polyhedron corresponding to the LP in standard form. In order to setup the methods one needs to identify a feasible solution. This is usually done by solving yet another LP; the *artificial LP*.

Consider the following artificial LP:

$$\begin{array}{lrcl} \min & \sum_{i=1}^{m} y_i, & & \\ \text{subject to} & \mathbf{A}\mathbf{x}+\mathbf{y} & = & \mathbf{b}, \\ & (\mathbf{x},\mathbf{y}) & \geq & \mathbf{0}, \end{array} \tag{3.10}$$

where $\mathbf{y} \in \mathbb{R}^m$ is called the vector of *artificial variables*. If $\mathbf{b} \geq \mathbf{0}$, then LP (3.10) is initialized by $\mathbf{y} = \mathbf{b}$. We will assume without loss of generality that $\mathbf{b} \geq \mathbf{0}$. One can easily modify the equality constraints in the original LP in standard form so that all equations hold where the righthand side is nonnegative.

Theorem 3.19 shows that solving LP (3.10) yields either a feasible solution to the original LP in standard form or the conclusion that the original LP has no feasible solutions at all.

Theorem 3.19. *Suppose that LP in standard form is given, where* $\mathbf{b} \geq \mathbf{0}$*. Consider its corresponding artificial LP* (3.10)*. Let* $(\mathbf{x}^{\mathsf{opt}}, \mathbf{y}^{\mathsf{opt}})$ *denote the optimal solution to the artificial LP. Then*

(i) the artificial LP is bounded and has at least one feasible solution,

(ii) if $\sum_{i=1}^{m} y_i = 0$ *then* $\mathbf{x}^{\mathsf{opt}}$ *is a feasible solution to the given LP,*

(iii) if $\sum_{i=1}^{m} y_i > 0$ *then the given LP has no feasible solution.*

Proof. (i) From the nonnegativity constraints $\mathbf{y} \geq \mathbf{0}$ it follows that $\min \sum_{i=1}^{m} y_i$ is bounded from below by zero. And from $\mathbf{b} \geq \mathbf{0}$ it follows that $\mathbf{y} = \mathbf{b}$ is a feasible solution to the artificial LP.

(ii) Let $(\mathbf{x}^{\mathsf{opt}}, \mathbf{y}^{\mathsf{opt}})$ be the optimal solution of the artificial LP. Suppose firstly that $\mathbf{y}^{\mathsf{opt}} = \mathbf{0}$. Then,

$$\mathbf{A}\mathbf{x}^{\mathsf{opt}} = \mathbf{b}$$

such $\mathbf{x}^{\mathsf{opt}}$ is feasible to the given LP.

(iii) Next suppose that $\mathbf{y}^{\mathsf{opt}} \neq 0$. Let $\mathbf{x}^*$ be a feasible solution to the given LP. From $\mathbf{A}\mathbf{x}^* = \mathbf{b}$ and $\mathbf{x}^* \geq \mathbf{0}$ it follows that $(\mathbf{x}^*, \mathbf{0})$ is feasible to the artificial LP contradicting the optimality of $(\mathbf{x}^{\mathsf{opt}}, \mathbf{y}^{\mathsf{opt}})$. □

Theorem 3.20 generalizes Theorem 3.19 by considering a more general artificial linear program.

Theorem 3.20. *Suppose that an LP in standard form is given. Consider the following artificial LP*

$$\begin{array}{lrcl} \min & \sum_{i=1}^{p} y_i, & & \\ \text{subject to} & \mathbf{A}\mathbf{x}+\mathbf{C}\mathbf{y} & = & \mathbf{b}, \\ & (\mathbf{x},\mathbf{y}) & \geq & \mathbf{0}, \end{array} \tag{3.11}$$

where $\mathbf{C} \in \mathbb{R}^{m \times p}$ *for* $1 \leq p \leq m$*. Suppose that it has a feasible solution. Let* $(\mathbf{x}^{\mathsf{opt}}, \mathbf{y}^{\mathsf{opt}})$ *denote the optimal solution to the artificial LP* (3.11)*. Then*

(i) the linear program (3.11) *is bounded,*

(ii) if $\sum_{i=1}^{p} y_i = 0$ *then* $\mathbf{x}^{\mathsf{opt}}$ *is a feasible solution to the given LP,*

(iii) if $\sum_{i=1}^{p} y_i > 0$*, then the given LP has no feasible solution.*

Proof. (i) From the nonnegativity constraints $\mathbf{y} \geq \mathbf{0}$ it follows that $\min \sum_{i=1}^{p} y_i$ is bounded from below by zero.

(ii) Let $(\mathbf{x}^{\mathsf{opt}}, \mathbf{y}^{\mathsf{opt}})$ be the optimal solution of the artificial LP. Suppose firstly that $\mathbf{y}^{\mathsf{opt}} = \mathbf{0}$. Then,

$$\mathbf{A}\mathbf{x}^{\mathsf{opt}} = \mathbf{b}.$$

This $\mathbf{x}^{\mathsf{opt}}$ is feasible to the given LP.

(iii) Next, suppose that $\mathbf{y}^{\mathsf{opt}} \neq 0$. Let $\mathbf{x}^*$ be a feasible solution to the given LP. From $\mathbf{A}\mathbf{x}^* = \mathbf{b}$ and $\mathbf{x}^* \geq \mathbf{0}$ it follows that $(\mathbf{x}^*, \mathbf{0})$ is feasible to the artificial LP, contradicting the optimality of $(\mathbf{x}^{\mathsf{opt}}, \mathbf{y}^{\mathsf{opt}})$. □

Theorem 3.21 shows a nice property of the simplex algorithm when applied to any artificial linear program. It shows that if the cost-reduced vector has negative entries only at positions of the artificial variables, then the corresponding basic feasible solution is already optimal, or the given linear program has no feasible solution at all. It means that the simplex iterations do not need to compute the cost-reduced entries for the artificial variables.

Theorem 3.21. *Consider the general artificial LP* (3.11) *corresponding to a given LP in standard form. Let* $(\mathbf{x}, \mathbf{y})$ *be a basic feasible solution with respect to basis* $\mathbf{s}$*. If the corresponding cost-reduced vector* $\overline{\mathbf{c}}$ *satisfies* $\overline{c}_i \geq 0$ *for all* $1 \leq i \leq n$*, then the current solution is either optimal or the given LP is infeasible.*

Proof. For the sake of simplicity we will write $\mathbf{z} = (\mathbf{x}, \mathbf{y})$. Hence $z_i = x_i$, if $1 \leq i \leq n$, and $z_i = y_{i-n}$, if $n+1 \leq i \leq n+p$. Let $\mathbf{c}$ denote the cost vector corresponding to the artificial LP. Hence $\mathbf{c}\mathbf{z} = \sum_{i=1}^{p} y_i$.

Suppose that $\overline{\mathbf{c}}$ has only nonnegative entries in the first n positions and at least one negative entry in the next p positions. Let

$$\mathcal{D} = \{d_1, \ldots, d_q\} = \{s_1, \ldots, s_m\} \cap \{n+1, \ldots, n+p\}$$

denote the set of basic artificial variables in the current solution $\mathbf{z}$.

We will show that from $\mathbf{z}$ a feasible and optimal solution to another artificial linear program, with respect to the original LP, can be computed so that by Theorem 3.20 one can conclude that either the original LP is infeasible or $\mathbf{x}'$ is a feasible solution to the original LP.

Consider the linear program that is obtained by removing the co-basic artificial variables from the artificial LP, i.e., consider

$$\begin{array}{lcl} \min & \mathbf{c}'\mathbf{v} = \sum_{i=n+1}^{q} v_i, & \\ \text{subject to} & \left(\mathbf{A} \;\; \mathbf{C}_{d_1-n} \;\; \ldots \;\; \mathbf{C}_{d_q-n} \right) \mathbf{v} & = \mathbf{b}, \\ & \mathbf{v} & \geq \mathbf{0}, \end{array} \tag{3.12}$$

where $\mathbf{v} \in \mathbb{R}^{n+q}$ and $q < p$.

By construction $\mathbf{v} = x_1, \ldots, x_n, y_{d_1-n}, \ldots, y_{d_q-n}$ is a basic feasible solution to LP (3.12) with basis $\mathbf{s}'$, where

$$s'_i = \begin{cases} s_i, & \text{if } s_i \leq n, \\ n+k, & \text{if } s_i = d_k. \end{cases}$$

Let $\mathbf{B} = \begin{pmatrix} \mathbf{A} & \mathbf{C} \end{pmatrix}_{\mathbf{s}}$ be the basis matrix corresponding to solution $\mathbf{z}$ to the artificial LP (3.11). And let $\mathbf{B}' = \begin{pmatrix} \mathbf{A} & \mathbf{C}_{d_1-n} & \ldots & \mathbf{C}_{d_q-n} \end{pmatrix}_{\mathbf{s}'}$ be the basis matrix corresponding to solution $\mathbf{v}$ to the artificial LP (3.12). Observe that

$$\mathbf{B}_i = \mathbf{A}_{s_i} = \mathbf{A}_{s'_i} = \mathbf{B}'_i,$$

if $s_i \leq n$, and

$$\mathbf{B}_i = \mathbf{C}_{s_i-n} = \mathbf{C}_{d_k-n} = \mathbf{B}'_i,$$

if $s_i = d_k > n$. Hence, $\mathbf{B} = \mathbf{B}'$.

Similarly, the cost reduced-vector $\overline{\mathbf{c}'}$ with respect to $\mathbf{v}$, satisfies, for $i \leq n$, using $\mathbf{c}'_{\mathbf{s}'} = \mathbf{c}_{\mathbf{s}}$

$$\overline{c'}_i = -\mathbf{c}'_{\mathbf{s}'}\mathbf{B}'^{-1}\mathbf{A}_i = -\mathbf{c}_{\mathbf{s}}\mathbf{B}^{-1}\mathbf{A}_i = \overline{c}_i \geq 0.$$

And if $i = n + k \leq n + q$ then it satisfies

$$\overline{c'}_i = 1 - \mathbf{c}'_{\mathbf{s}'}\mathbf{B}'^{-1}\mathbf{C}_{d_k} = 1 - \mathbf{c}_{\mathbf{s}}\mathbf{B}^{-1}\mathbf{C}_{d_k} = \overline{c}_{d_k} = 0,$$

by Lemma 3.9 since d_k is basic with respect to $\mathbf{s}$.

Therefore, $\overline{\mathbf{c}'} \geq \mathbf{0}$ and, therefore, $\mathbf{v}$ is optimal to Eq. (3.12). The theorem follows from Theorem 3.20 since LP (3.12) is an artificial LP of the same form and corresponds to the given LP. □

3.1.4 Verification of the Result

This section shows how the result can be verified very efficiently using *certificates*. Our definitions are based on the definitions used in complexity theory (see [Hro01]).

Definition 3.22. *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be some sets and $\mathcal{X} \subseteq \mathcal{S}_1$. A polynomial time computable function $g : \mathcal{S}_1 \times \mathcal{S}_2 \to \{0, 1\}$ is called a* validating function for $\mathcal{X}$*, if*

$$\mathcal{X} = \{\, w \in \mathcal{S}_1 |\, \exists c \in \mathcal{S}_2 :\ g(w, c) = 1 \}\,.$$

If g is a validating function for $\mathcal{X}$ and $g(w, c) = 1$, then c is called a certificate of the fact $w \in \mathcal{X}$*.*

Let x be a boolean expression. We will write $|x|_b$ to denote the boolean evaluation of the expression x. For example let $x \in \{0, 1\}$ and $y \in \{0, 1\}$, then $|x \wedge y|_b = 1$ if and only if $x = 1$ and $y = 1$.

Let $\mathcal{U} = (\mathbb{R}^{m\times n} \times \mathbb{R}^m \times \mathbb{R}^n)$ be the set representing all linear programs in standard form, i.e., $(\mathbf{A}, \mathbf{b}, \mathbf{c}) \in \mathcal{U}$ corresponds to an LP in standard form. For the sake of simplicity we will call any tuple $(\mathbf{A}, \mathbf{b}, \mathbf{c}) \in \mathcal{U}$ an LP if we mean the LP to be represented by $(\mathbf{A}, \mathbf{b}, \mathbf{c})$.

Example 3.23 (Certificate of Feasibility). Let

$$\mathcal{X} = \{\, (\mathbf{A}, \mathbf{b}, \mathbf{c}) \in \mathcal{U} |\, \text{LP } (\mathbf{A}, \mathbf{b}, \mathbf{c}) \text{ is feasible} \}$$

be the set of feasible linear programs in standard form. By definition $(\mathbf{A}, \mathbf{b}, \mathbf{c}) \in \mathcal{U}$ is feasible if and only if there exists an $\mathbf{x} \in \mathbb{R}^n$ such that

$$\mathbf{Ax} = \mathbf{b} \wedge \mathbf{x} \geq \mathbf{0}.$$

Hence $g : \mathcal{U} \times \mathbb{R}^n \to \{0, 1\}$ defined by

$$g\left((\mathbf{A}, \mathbf{b}, \mathbf{c}), \mathbf{x}\right) = |\mathbf{Ax} = \mathbf{b} \wedge \mathbf{x} \geq \mathbf{0}|_b$$

is a validating function for $\mathcal{X}$ and $\mathbf{x}$ is a *certificate of the fact that* $(\mathbf{A}, \mathbf{b}, \mathbf{c}) \in \mathcal{X}$, i.e., $\mathbf{x}$ is a *certificate of feasibility* of the LP $(\mathbf{A}, \mathbf{b}, \mathbf{c})$. ◇

It follows from Example 3.23 that if a solution $\mathbf{x}$ is a certificate of feasibility, then the given LP is feasible, but in particular that $\mathbf{x}$ is a feasible solution. Similarly, we use a *certificate of optimality* to prove that a given result $\mathbf{x}^{\mathsf{opt}}$ is indeed optimal. Such a certificate typically requires the *dual* of the given LP, which we will introduce next.

3.1.4.1 Certificate of Optimality

To validate optimality of a solution efficiently one considers the *dual* of a linear program. Here we will introduce the dual linear program and its relation to a given linear program in standard form. Then we show how these relations are used to provide a *certificate of optimality*.

Given an LP in standard form, its dual LP is given by

$$\begin{array}{rll} \max & \mathbf{pb}, & \\ \text{subject to} & \mathbf{pA} & \leq \quad \mathbf{c}, \end{array} \tag{3.13}$$

where the $\mathbf{p} \in \mathbb{R}^m$ are the unknowns (Lagrange multipliers). The given linear program is called the *primal* linear program corresponding to the dual linear program Eq. (3.13).

The following theorems, which we will prove for the primal LP in standard form, will form the basis for validating an optimal solution.

Theorem 3.24 (Weak Duality)**.** *Let* $\mathbf{x}$ *be a feasible solution to the primal LP in standard form and let* $\mathbf{p}$ *be a feasible solution to the corresponding dual LP. Then,*

$$\mathbf{pb} \leq \mathbf{cx}.$$

Proof. Since $\mathbf{x}$ is feasible to the primal LP it satisfies $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} \geq 0$. The feasibility of $\mathbf{p}$ implies that $\mathbf{pA} \leq \mathbf{c}$. Thus,

$$\mathbf{pb} = \mathbf{pAx} \leq \mathbf{cx}.$$

□

Theorem 3.25. *Let* $\mathbf{x}$ *be a feasible solution to the primal LP in standard form and* $\mathbf{p}$ *be a feasible solution to the corresponding dual LP. If* $\mathbf{pb} = \mathbf{cx}$*, then both* $\mathbf{x}$ *and* $\mathbf{p}$ *are optimal.*

Proof. Let $\mathbf{x}'$ be any feasible solution to the primal LP. Then from Theorem 3.24 it follows that

$$\mathbf{cx} = \mathbf{pb} \leq \mathbf{cx}',$$

thus proving the optimality of $\mathbf{x}$.

Similarly, let $\mathbf{p}'$ be any feasible solution to the dual LP. Then, by Theorem 3.24 again, we have

$$\mathbf{pb} = \mathbf{cx} \geq \mathbf{p}'\mathbf{b},$$

thus proving the optimality of $\mathbf{p}$. □

Theorem 3.26 (Strong Duality)**.** *If the primal LP in standard form has an optimal solution, then so does the corresponding dual LP, and the optimal costs are equal.*

Proof. Suppose that the primal LP has an optimal solution. By the fundamental theorem of linear programming (Theorem 3.3) there is a basic optimal solution $\mathbf{x}$ with respect to some basis matrix $\mathbf{B}$. From Theorem 3.7 it follows that the corresponding cost-reduced vector satisfies $\overline{\mathbf{c}} \geq 0$. Hence by Definition 3.8

$$\mathbf{c} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A} \geq 0,$$

and, therefore,

$$\mathbf{c_s}\mathbf{B}^{-1}\mathbf{A} \leq \mathbf{c}.$$

Therefore, the vector

$$\mathbf{p} = \mathbf{c_s}\mathbf{B}^{-1} \tag{3.14}$$

is feasible to the dual LP. Moreover,

$$\mathbf{pb} = \mathbf{c_s}\mathbf{B}^{-1}\mathbf{b} = \mathbf{c_s}\mathbf{x_s} = \mathbf{cx}.$$

Hence by Theorem 3.25 it follows that $\mathbf{p}$ is optimal. □

It follows from Theorem 3.25 that if $\mathbf{x}$ is feasible to the primal LP and $\mathbf{p}$ is feasible to the corresponding dual LP, then $\mathbf{pb} = \mathbf{cx}$ if and only if both $\mathbf{x}$ and $\mathbf{p}$ are optimal. Hence a certificate of optimality for the given LP in standard form is the tuple $(\mathbf{x}, \mathbf{p})$.

3.1.4.2 Certificate of Infeasibility

Note that by Theorem 3.19 the artificial LP (3.10) with respect to an LP in standard form has an optimal solution and by the strong duality theorem (Theorem 3.26) also its dual has. Therefore, a certificate of optimality exists for the artificial LP. From this certificate we can derive a certificate for infeasibility using Farkas' lemma.

Theorem 3.27 (Farkas' Lemma)**.** *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then, exactly one of the following holds:*

(i) there exists some $\mathbf{x} \geq \mathbf{0}$ such that $\mathbf{Ax} = \mathbf{b}$, or

(ii) there exists some $\mathbf{p}$ such that $\mathbf{pA} \leq \mathbf{0}$ and $\mathbf{pb} > 0$.

Proof. (i) and (ii) cannot hold both since they would imply that

$$0 < \mathbf{pb} = \mathbf{pAx} \leq \mathbf{0x} = 0.$$

Suppose (i) does not hold. Given $\mathbf{A}$ and $\mathbf{b}$ consider the LP in standard form, where $\mathbf{c}$ is chosen arbitrarily. Let $\mathbf{p}$ be the optimal solution to the dual linear program corresponding to the artificial LP (3.10), which can be derived from Eq. (3.13) as follows:

$$\begin{array}{lll} \max & \mathbf{pb}, & \\ \text{subject to} & \mathbf{p}[\mathbf{A}|\mathbf{I}] & \leq \quad [\mathbf{0}|\mathbf{1}], \end{array}$$

where $\mathbf{0}$ denotes a length n vector consisting of zeros and $\mathbf{1}$ a length m vector consisting of ones.

By Theorem 3.25 it follows that the optimal costs for the dual linear program and the artificial linear program are equal. By Theorem 3.19 it follows that the optimal costs of the artificial linear program are positive by the infeasibility of the original LP. Hence $\mathbf{pb} > 0$.

Observe that from the dual feasibility of $\mathbf{p}$ that $\mathbf{pA} \leq 0$, hence (ii) holds. □

From Theorem 3.27 it follows that any $\mathbf{p}$ satisfying (ii) provides a proof that the given LP is infeasible. It follows that a certificate of infeasibility is given by $\mathbf{p}$.

3.1.4.3 Certificate of Unboundedness

From Theorem 3.10 it follows that if $\mathbf{x}$ is a feasible solution and some direction $\mathbf{d} \geq 0$ is a feasible direction at $\mathbf{x}$ improving the costs, then the LP is unbounded. Hence a certificate of unboundedness is given by $(\mathbf{x}, \mathbf{d})$.

3.2 Implementations of the Simplex Iterations

Recall from Section 3.1.1 that the simplex algorithm performs the following steps during each iteration given an LP in standard form and a basic feasible solution $\mathbf{x}$, with basis $\mathbf{s}$ and basis matrix $\mathbf{B} = \mathbf{A}_\mathbf{s}$:

1. **Entering Variable:** Pick ℓ such that $\overline{c}_\ell < 0$, where

   $$\overline{\mathbf{c}} = \mathbf{c} - \mathbf{c}_\mathbf{s}\mathbf{B}^{-1}\mathbf{A}.$$

   If no such ℓ exists then output current solution, where $\mathbf{x}_\mathbf{s} = \mathbf{B}^{-1}\mathbf{b}$, being the optimum.

2. **Leaving Variable:** Compute

   $$\theta = \min\left\{ -\frac{x_{s_i}}{d_{s_i}} \middle| d_i < 0 \right\},$$

   where $\mathbf{d}_\mathbf{s} = -\mathbf{B}^{-1}\mathbf{A}_\ell$, and $\mathbf{x}_{s_i} = \mathbf{B}^{-1}\mathbf{b}$. Let k be the index such that $\theta = -\frac{x_k}{d_k}$. If no such k exists, then exit and report "unbounded LP".

3. **Update Basis:** Replace s_k by ℓ in $\mathbf{s}$ and update $\mathbf{B}$ by replacing the k-th column of $\mathbf{B}$ by $\mathbf{A}_\ell$.

Naively, to compute $\overline{\mathbf{c}}$ and $\mathbf{d}$ one needs to compute $\mathbf{B}^{-1}$ at each iteration. We will show in the remainder of this section how to avoid computing these matrix inverses.

A precise rule to select ℓ and k is called a pivoting rule, see Remark 3.14. In this Section, for simplicity, we will give the algorithms where Dantzig's original pivoting rule is applied.

3.2.1 Large Tableau Simplex

This section shows how all data used in the simplex method can be represented by a matrix and that each iteration reduces to elementary row operations, i.e., adding rows and multiplying the rows by a scalar. Then, observing that the values in the matrix will be rational even if the inputs are integer, we show how to avoid computing with fractions. Lastly an upper bound on the size of the values in the representation will be derived.

3.2.1.1 The Simplex Tableau

Let $\mathbf{T}$ be an $(m+1) \times (n+1)$ matrix as follows

$$\left.\begin{array}{ccc|c} a'_{11} & \dots & a'_{1n} & b'_1 \\ \vdots & & \vdots & \vdots \\ a'_{m1} & \dots & a'_{mn} & b'_m \\ \hline \overline{c}_1 & \dots & \overline{c}_n & -z \end{array}\right. , \tag{3.15}$$

Definition 3.28. *For any LP in standard form, let* $\mathbf{x}$ *be a basic feasible solution with respect to basis* $\mathbf{s}$ *and let* $\mathbf{B} = \mathbf{A}_\mathbf{s}$ *be the basis matrix. Let* $\mathbf{T}$ *be given by Eq.* (3.15)*. If*

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}_\mathbf{s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix} \tag{3.16}$$

then $\mathbf{T}$ *is called the* simplex tableau *corresponding to basis* $\mathbf{s}$*.*

The following proposition shows that by elementary row operations on $\mathbf{T}$ one can transform $\mathbf{T}$ to tableau $\mathbf{T}'$ corresponding to the new basis $\mathbf{s}'$ after one simplex iteration. In the following, let $\mathbf{I}_m$ denote the $m \times m$ identity matrix and $\mathbf{e}_i$ the i-th column of $\mathbf{I}_m$. The vector $\mathbf{e}_i$ is also know as the *i-th unity* vector.

Theorem 3.29. *Let* $\mathbf{T}$ *be the tableau corresponding to basis* $\mathbf{s}$*. Let*

$$\mathbf{T}' = \mathbf{Q}\mathbf{T},$$

where $\mathbf{Q}$ *is the matrix corresponding to row reduction, or* pivot*, on element* $t_{k\ell} \neq 0$*, where* $1 \leq k \leq m$ *and* $1 \leq \ell \leq n$*, i.e.,*

$$t'_{ij} = t_{ij} - \frac{t_{i\ell}t_{kj}}{t_{k\ell}} \qquad \text{if } i \neq k \tag{3.17}$$

$$t'_{kj} = \frac{t_{kj}}{t_{k\ell}}. \tag{3.18}$$

Then $\mathbf{T}'$ *is the tableau corresponding to basis* $\mathbf{s}' = (s_1, \dots, s_{k-1}, \ell, s_{k+1}, \dots, s_m)$*.*

Proof. Let $\mathbf{T}$ be a tableau corresponding to basis $\mathbf{s}$. Let $k \in \{1, \dots, m\}$ and $\ell \in \{1, \dots, n\}$. Suppose $t_{k\ell} \neq 0$ and let $\mathbf{T}' = \mathbf{Q}\mathbf{T}$ satisfy Eq. (3.17) and Eq. (3.18). We will show that $\mathbf{T}'$ satisfies Definition 3.28, i.e.,

$$\mathbf{T}' = \begin{pmatrix} \mathbf{B}'^{-1} & \mathbf{0} \\ -\mathbf{c}_{\mathbf{s}'}\mathbf{B}'^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix}, \tag{3.19}$$

where $\mathbf{s}' = (s_1, \dots, s_{k-1}, \ell, s_{k+1}, \dots, s_m)$ and $\mathbf{B}' = \mathbf{A}_{\mathbf{s}'}$.

Since $\mathbf{T}$ is a tableau with basis $\mathbf{s}$ it holds that

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{A}' & \mathbf{b}' \\ \overline{\mathbf{c}} & -z \end{pmatrix},$$

where $\mathbf{B} = \mathbf{A_s}$, $\mathbf{A}' = \mathbf{B}^{-1}\mathbf{A}$, $\mathbf{b}' = \mathbf{B}^{-1}\mathbf{b}$, $\overline{\mathbf{c}} = \mathbf{c} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}$, and $z = \mathbf{c_s}\mathbf{B}^{-1}\mathbf{b}$.

From Eq. (3.17) and Eq. (3.18) it follows that $\mathbf{Q} \in \mathbb{R}^{(m+1)\times(m+1)}$ is equal to

$$\begin{pmatrix} 1 & & 0 & -\frac{a'_{i\ell}}{a'_{k\ell}} & 0 & \dots & 0 \\ & \ddots & & \vdots & \vdots & & \vdots \\ 0 & & 1 & -\frac{a'_{(k-1)\ell}}{a'_{k\ell}} & 0 & \dots & 0 \\ 0 & & 0 & \frac{1}{a'_{k\ell}} & 0 & & 0 \\ 0 & \dots & 0 & -\frac{a'_{(k+1)\ell}}{a'_{k\ell}} & 1 & & 0 \\ \vdots & & \vdots & \vdots & & \ddots & \\ 0 & \dots & 0 & -\frac{\overline{c}_\ell}{a'_{k\ell}} & 0 & & 1 \end{pmatrix}. \tag{3.20}$$

Let

$$\mathbf{QT} = \mathbf{Q}\begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{Q}'\mathbf{A}' & \mathbf{Q}'\mathbf{b}' \\ \overline{\mathbf{c}'} & -z' \end{pmatrix}, \tag{3.21}$$

where $\mathbf{Q}' \in \mathbb{R}^{m\times m}$ consists of the first m rows and columns of $\mathbf{Q}$.

We will show that

(i)

$$\mathbf{Q}'\mathbf{B}^{-1} = \mathbf{B}'^{-1} \tag{3.22}$$

implying $\mathbf{Q}'\mathbf{A}' = \mathbf{B}'^{-1}\mathbf{A}$ and $\mathbf{Q}'\mathbf{b}' = \mathbf{B}'^{-1}\mathbf{b}$, and,

(ii) $\overline{\mathbf{c}'} = \mathbf{c} - \mathbf{c_{s'}}\mathbf{B}'^{-1}\mathbf{A}$ and $-z' = -\mathbf{c_{s'}}\mathbf{B}'^{-1}\mathbf{b}$, implying that $\mathbf{QT}$ satisfies Eq. (3.19).

(i) We have

$$(\mathbf{Q}'\mathbf{A}'_j)_i = a'_{ij} - \frac{a'_{i\ell}}{a'_{k\ell}}a'_{kj}, \tag{3.23}$$

for $i \neq k$ and

$$(\mathbf{Q}'\mathbf{A}'_j)_k = \frac{a'_{kj}}{a'_{k\ell}}. \tag{3.24}$$

First observe that

$$\begin{aligned} \mathbf{Q}'\mathbf{B}^{-1}\mathbf{B}' &= \mathbf{Q}'\left(\mathbf{B}^{-1}\mathbf{B}'_1, \dots, \mathbf{B}^{-1}\mathbf{B}'_m\right) \\ &= \mathbf{Q}'\left(\mathbf{B}^{-1}\mathbf{A}_{s_1}, \dots, \mathbf{B}^{-1}\mathbf{A}_{s_{k-1}}, \mathbf{B}^{-1}\mathbf{A}_\ell, \mathbf{B}^{-1}\mathbf{A}_{s_{k+1}}, \dots, \mathbf{B}^{-1}\mathbf{A}_{s_m}\right) \\ &= \mathbf{Q}'\left(\mathbf{e}_1, \dots, \mathbf{e}_{k-1}, \mathbf{A}'_\ell, \mathbf{e}_{k+1}, \dots, \mathbf{e}_m\right). \end{aligned} \tag{3.25}$$

For $i \neq k$ we have $\mathbf{Q}'\mathbf{e}_i = \mathbf{Q}'_i = \mathbf{e}_i$. And Eqs. (3.23) and (3.24) imply $\mathbf{Q}'\mathbf{A}_\ell = \mathbf{e}_k$. Hence by Eq. (3.25)

$$\mathbf{Q}'\mathbf{B}^{-1}\mathbf{B}' = \mathbf{I}_m.$$

(ii) From Eq. (3.20) it follows that

$$\begin{aligned}
\overline{c}'_j &= \overline{c}_j - \frac{\overline{c}_\ell a'_{kj}}{a'_{k\ell}} \\
&= \left(c_j - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}_j\right) - \frac{a'_{kj}\left(c_\ell - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}_\ell\right)}{a'_{k\ell}} \\
&= c_j - \sum_{i\in\mathbf{s}} c_i \left(a'_{ij} - \frac{a'_{kj}a'i\ell}{a'_{k\ell}}\right) - c_\ell \frac{a'_{kj}}{a'_{k\ell}} \\
&= c_j - \sum_{i\in\mathbf{s}'} c_i \mathbf{Q}'\mathbf{A}'_i \\
&= c_j - \mathbf{c_{s'}}\mathbf{B'}^{-1}\mathbf{A}_j, \qquad (3.26)
\end{aligned}$$

for $j = 1, \ldots n$, where we used the identities from Eq. (3.23), Eq. (3.24), and Eq. (3.22).

Similarly,

$$-z' = -z - \frac{\overline{c}_\ell b'_k}{a'_{k\ell}} = \cdots = -\mathbf{c_{s'}}\mathbf{B'}^{-1}\mathbf{b}. \qquad (3.27)$$

□

Notice that in the tableau, the values $\overline{\mathbf{c}}$ are present in the last row and the valid basic directions are easily extracted from the columns of $\mathbf{A}'$, indeed by Eq. (3.2)

$$\mathbf{d}^i_\mathbf{s} = -\mathbf{B}^{-1}\mathbf{A}_i = -\mathbf{A}'_i.$$

With the tableau representation, the simplex method becomes:

1. **Entering Variable:** Pick ℓ such that $t_{(m+1)\ell} < 0$. If no such $\ell \leq n$ exists then the output is the current solution which is optimal, i.e.,

   $$\mathbf{x}_{s_i} = \mathbf{t}_{i,n+1},$$

   for all $i = 1, \ldots, m$ and $x_j = 0$ for $j \notin \mathbf{s}$.

2. **Leaving Variable:** Compute

   $$\theta = \min\left\{ \frac{t_{i(n+1)}}{t_{i\ell}} \,\middle|\, t_{i\ell} > 0 \text{ and } i \in \{1, \ldots, m\} \right\}. \qquad (3.28)$$

   Let k be the index such that $\theta = \frac{t_{k(n+1)}}{t_{k\ell}}$. If no such k exists, then exit and report "unbounded LP".

3. **Update Basis and Tableau:** Replace s_k by ℓ in $\mathbf{s}$ and compute $\mathbf{T}'$ from $\mathbf{T}$ by using Eqs. (3.17) and (3.18) in Theorem 3.29.

The following algorithms provide precisely each step of the simplex iterations.

Algorithm 3.1: $(\mathbf{T}, \mathbf{s}, \text{pred}) \leftarrow \mathsf{Iterate}_{\mathsf{LT},\mathsf{RP}}(\mathbf{T}, \mathbf{s})$

Input: $\mathbf{T}, \mathbf{s}$.
Output: $\mathbf{T}, \mathbf{s}, \text{pred}$.

$(\ell, k) \leftarrow \mathsf{FindPivotElement}(\mathbf{T})$;
if $\ell = 0$ **then**
  return $(\mathbf{T}, \mathbf{s}, \mathsf{Optimal})$;
else if $k = 0$ **then**
  return $(\mathbf{T}, \mathbf{s}, \mathsf{UnboundedLP})$;
$\mathbf{T} \leftarrow \mathsf{Pivot}_{\mathsf{RP}}(\mathbf{T}, \ell, k)$;
$s_k \leftarrow \ell$;
return $\mathsf{Iterate}_{\mathsf{RP}}(\mathbf{T}, \mathbf{s})$;

Algorithm 3.2: $(\ell, k) \leftarrow \mathsf{FindPivotElement}(\mathbf{T})$

Input: $\mathbf{T}$.
Output: ℓ, k.

$\ell \leftarrow \operatorname{argmin}\left\{ t_{(m+1)i} \middle| \, i \in \{1, \ldots, n\} \right\}$;
if $t_{(m+1)\ell} \geq 0$ **then**
  return $(0, 0)$;
foreach $i \in \{1, \ldots, m\}$ **do**
  if $t_{i\ell} > 0$ **then** $r_i \leftarrow \frac{t_{i(n+1)}}{t_{i\ell}}$;
  else $r_i \leftarrow \infty$;
$k \leftarrow \operatorname{argmin}(r_1, \ldots, r_m)$;
if $r_k = \infty$ **then**
  return $(\ell, 0)$;
return (ℓ, k);

Algorithm 3.3: $\mathbf{T}' \leftarrow \mathsf{Pivot}_{\mathsf{RP}}(\mathbf{T}, k, \ell)$

Input: $\mathbf{T}, \ell, k$.
Output: $\mathbf{T}'$.

foreach $i \in \{1, \ldots, n+1\}$ **do**
  foreach $j \in \{1, \ldots, m+1\}$ **do**
    if $i \neq k$ **then**
      $t'_{ij} \leftarrow t_{ij} - \frac{t_{i\ell} t_{kj}}{t_{k\ell}}$;
    else
      $t'_{ij} \leftarrow \frac{t_{ij}}{t_{k\ell}}$;
return $\mathbf{T}'$;

3.2.1.2 Integer Pivoting

Observe that row reduction is defined over $\mathbf{Q}$, even if all inputs are integer. Many cryptographic tools, however, are able to deal with integers –or rather field elements– only. The results of [Ros05] and [AP01] describe a way to change the pivoting procedure in the simplex algorithm so that in each iteration the tableau contains integer values only.

These tableaux differ only in a constant factor from the rational tableaux containing the correct values. The following proposition extends their approach by making sure that the integer tableau is a positive multiple of the corresponding rational tableau. In addition, we provide a simple and complete proof of correctness.

Theorem 3.30. *Suppose that an LP is given in standard form, where all coefficients* $\mathbf{A}, \mathbf{c}$ *and* $\mathbf{b}$ *are integer. Let* $\mathbf{T}$ *be its tableau corresponding to basis* $\mathbf{s}$ *and basis matrix* $\mathbf{B}$*. Let* $\widetilde{\mathbf{T}} = |\det(\mathbf{B})|\mathbf{T}$*. Then all values in* $\widetilde{T}$ *are integer.*

Proof. Let $\alpha := \operatorname{sgn}(\det(\mathbf{B}))$.

By the definition of a determinant it follows that every square submatrix of $\mathbf{A}$ has an integer determinant. Hence $\det(\mathbf{B})$ and $\det(\mathbf{B}')$ are integer as well as $\operatorname{adj}(\mathbf{B})$ and $\operatorname{adj}(\mathbf{B}')$. From linear algebra we know that

$$\operatorname{adj}(\mathbf{M}) = \det(\mathbf{M})\mathbf{M}^{-1}$$

for any invertible square matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$.

Observe that by Definition 3.28

$$\widetilde{\mathbf{T}} = |\det(\mathbf{B})|\mathbf{T} = \alpha \begin{pmatrix} \operatorname{adj}(\mathbf{B}) & \mathbf{0} \\ -\mathbf{c_s}\operatorname{adj}(\mathbf{B}) & \det(\mathbf{B}) \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix}, \tag{3.29}$$

being a product of two integer matrices. Therefore, $\widetilde{\mathbf{T}}$ is integer. □

Theorem 3.31 (Integer Pivoting)**.** *Suppose that an LP is given in standard form, where all coefficients are integer. Let* $\mathbf{T}$ *be its tableau corresponding to basis* $\mathbf{s}$ *at the beginning of an iteration of the simplex algorithm. Suppose that the basis is updated, where* ℓ *enters the basis and* s_k *leaves the basis. Let tableau* $\mathbf{T}' = \mathbf{QT}$ *be the resulting tableau corresponding to basis* $\mathbf{s}'$*. Let* $\widetilde{\mathbf{T}} = |\det(\mathbf{B})|\mathbf{T}$ *and* $\widetilde{\mathbf{T}'} = |\det(\mathbf{B}')|\mathbf{T}'$*, where* $\mathbf{B} = \mathbf{A_s}$ *and* $\mathbf{B}' = \mathbf{A_{s'}}$*. Then*

$$\widetilde{t}'_{ij} = \frac{\widetilde{t}_{\ell k}\widetilde{t}_{ij} - \widetilde{t}_{i\ell}\widetilde{t}_{kj}}{|\det(\mathbf{B})|} \qquad \text{if } i \neq k \tag{3.30}$$

$$\widetilde{t}'_{kj} = \widetilde{t}_{kj}, \tag{3.31}$$

Proof. Let $\alpha := \operatorname{sgn}(\det(\mathbf{B}))$. Let $\mathbf{A}' = \mathbf{B}^{-1}\mathbf{A}$ and $\widetilde{\mathbf{A}'} = |\det(\mathbf{B})|\mathbf{A}'$ and, similarly, $\mathbf{A}'' = \mathbf{B}'^{-1}\mathbf{A}$ and $\widetilde{\mathbf{A}''} = |\det(\mathbf{B}')|\mathbf{A}''$. We show that the entries $\widetilde{\mathbf{A}'}$ can be written as a determinant depending on $\mathbf{B}$.

$$\begin{aligned}
\widetilde{a'}_{ij} &= \alpha\det(\mathbf{B})a'_{ij} \\
&= \alpha\det(\mathbf{B})\det\left(\mathbf{e}_1,\ldots,\mathbf{e}_{i-1},\mathbf{A}'_j,\mathbf{e}_{i+1},\ldots,\mathbf{e}_m\right) \\
&= \alpha\det(\mathbf{B})\det\left(\mathbf{B}^{-1}\left(\mathbf{B}_1,\ldots,\mathbf{B}_{i-1},\mathbf{A}_j,\mathbf{B}_{i+1},\ldots,\mathbf{B}_m\right)\right) \\
&= \alpha\det(\mathbf{B})\det(\mathbf{B}^{-1})\det\left(\mathbf{A}_{s_1},\ldots,\mathbf{A}_{s_{i-1}},\mathbf{A}_j,\mathbf{A}_{s_{i+1}},\ldots,\mathbf{A}_{s_m}\right),
\end{aligned}$$

hence, using $\det(\mathbf{B}) \neq 0$,

$$\widetilde{a'}_{ij} = \alpha\det\left(\mathbf{A}_{s_1},\ldots,\mathbf{A}_{s_{i-1}},\mathbf{A}_j,\mathbf{A}_{s_{i+1}},\ldots,\mathbf{A}_{s_m}\right). \tag{3.32}$$

Hence, From Eq. (3.32) it follows that

$$\widetilde{a'}_{k\ell} = \alpha\det(\mathbf{B}'). \tag{3.33}$$

Observe that from the pivot row selection Eq. (3.28) we have $a'_{k\ell} > 0$. Hence $\widetilde{a}'_{k\ell} = |\det(\mathbf{B})|a'_{k\ell} > 0$, and from Eq. (3.33) it follows that $\operatorname{sgn}(\mathbf{B}') = \alpha$ and hence

$$\widetilde{a}'_{k\ell} = |\det(\mathbf{B}')|,$$

which equals the value of the current pivot element $\widetilde{t}_{k\ell}$.

Theorem 3.29 implies that $\mathbf{T}' = \mathbf{QT}$ satisfies Eqs. (3.17) and (3.18). Hence, using Eqs. (3.29) and (3.33),

$$\begin{aligned}\widetilde{t}'_{ij} &= |\det(\mathbf{B}')|t'_{ij} \\ &= \frac{\widetilde{t}_{k\ell}}{|\det(\mathbf{B})|}\left(\frac{\widetilde{t}_{ij}\widetilde{t}_{k\ell} - \widetilde{t}_{i\ell}\widetilde{t}_{kj}}{\widetilde{t}_{k\ell}}\right) \\ &= \frac{\widetilde{t}_{ij}\widetilde{t}_{k\ell} - \widetilde{t}_{i\ell}\widetilde{t}_{kj}}{|\det(\mathbf{B})|},\end{aligned}$$

if $i \neq k$. Also,

$$\begin{aligned}\widetilde{t}'_{kj} &= |\det(\mathbf{B}')|t'_{kj} \\ &= \widetilde{t}_{k\ell}\frac{\widetilde{t}_{kj}}{\widetilde{t}_{k\ell}} \\ &= \widetilde{t}_{kj}.\end{aligned}$$

□

Since all entries in $\widetilde{\mathbf{T}}$ have the same sign as the entries in $\mathbf{T}$ one can run Algorithm 3.9 to select the pivot element. With respect to Algorithm 3.1 only the updating part is different. Algorithm 3.4 shows precisely each step of the simplex algorithm keeping all values in the tableau integer.

Algorithm 3.4: $(\mathbf{T}, \mathbf{s}, \text{pred}, q) \leftarrow \mathsf{Iterate}_{\mathsf{LT},\mathsf{IP}}(\mathbf{T}, \mathbf{s}, q)$

Input: $\mathbf{T}, q, \mathbf{s}$.
Output: $\mathbf{T}, q, \mathbf{s}, \text{pred}$.

```
(ℓ, k) ← FindPivotElement(T);
if ℓ = 0 then
    return (T, s, Optimal, q);
else if k = 0 then
    return (T, s, UnboundedLP, q);
(T, q) ← Pivot_IP(T, ℓ, k, q);
s_k ← ℓ;
return Iterate_IP(T, s, q);
```

3.2.1.3 Size of the Numbers in the Tableau

The efficiency of simplex using integer pivoting will be dominated by the size of the numbers in the tableau.

Algorithm 3.5: $(\mathbf{T}', q) \leftarrow \mathsf{Pivot}_{\mathsf{IP}}(\mathbf{T}, k, \ell, q)$

Input: $\mathbf{T}, \ell, k$.
Output: $\mathbf{T}'$.

$q' = t_{k\ell}$;
foreach $i \in \{1, \ldots, m+1\}$ **do**
  foreach $j \in \{1, \ldots, n+1\}$ **do**
    if $i \neq k$ **then**
      $t'_{ij} \leftarrow \frac{t_{ij}t_{k\ell} - t_{i\ell}t_{kj}}{q}$;
    else
      $t'_{ij} = t_{ij}$;
return $(\mathbf{T}', q')$;

Remark 3.32. The solution of Li and Atallah [LA06] keeps its tableau integer valued by pivoting using Eqs. (3.30) and (3.31) where the (proper) division by $|\det(\mathbf{B})|$ is omitted. It follows that in their solution the size of the numbers in the tableau doubles each iteration, i.e., grows exponentially fast. ◇

By applying the division in Eq. (3.30) the size of the numbers in the tableau is bounded if the inputs are bounded [Tof07]. Actually, we follow the result of [Goe94], which gives an upper bound on the size of the output only, to derive an improved upper bound for all values in the tableau using the results in the proof of Theorem 3.31.

For any vector $\mathbf{v}$ let $\|\mathbf{v}\|$ denote its Euclidian norm. Theorem 3.34 provides an upper bound on the values in any integer tableau $\widetilde{\mathbf{T}}$.

Lemma 3.33 (Hadamard's Inequality [Had93])**.** *For any square matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$, we have*

$$\det(\mathbf{X}) \leq \prod_{i=1}^{n} \|\mathbf{X}_i\|.$$

Theorem 3.34. *Given an LP in standard form, where the bit size of the coefficients $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ is bounded by some positive integer N. Let $\mathbf{x}$ be a basic feasible solution $\mathbf{x}$ with basis $\mathbf{s}$ and basis matrix $\mathbf{B}$. Then its corresponding tableau $\mathbf{T}$ satisfies*

$$|\det(\mathbf{B})t_{ij}| < 2^L,$$

where

$$L = (m+1)N + \frac{m}{2}\log(m) + \log(m+1). \tag{3.34}$$

Proof. We consider the four parts of the tableau

$$\mathbf{T} = \begin{pmatrix} \mathbf{A}' & \mathbf{b}' \\ \bar{\mathbf{c}} & -z \end{pmatrix},$$

where $\mathbf{A}' = \mathbf{B}^{-1}\mathbf{A}$, $\mathbf{b}' = \mathbf{B}^{-1}\mathbf{b}$, $\bar{c} = \mathbf{c} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}$ and $z = \mathbf{c_s}\mathbf{B}^{-1}\mathbf{b}$.

Let $\widetilde{T} = |\det(\mathbf{B})|\mathbf{T}$.

From Eq. (3.32) and Hadamard's Inequality it follows that

$$\begin{aligned}
|\widetilde{a}_{ij}| &= |\det(\mathbf{B})||a'_{ij}| \\
&= \left|\det\left(\mathbf{A}_{s_1}, \ldots, \mathbf{A}_{s_{i-1}}, \mathbf{A}_j, \mathbf{A}_{s_{i+1}}, \ldots, \mathbf{A}_{s_m}\right)\right| \\
&\leq \|\mathbf{A}_j\| \prod_{k=1, k\neq i}^{m} \|\mathbf{A}_{s_k}\| \\
&\leq \prod_{k=1}^{m} \sqrt{\sum_{\ell=1}^{m} (2^N)^2} \\
&= m^{m/2} 2^{mN}, \qquad (3.35)
\end{aligned}$$

Similarly,

$$\begin{aligned}
|\widetilde{b}_{i(n+1)}| &= |\det(\mathbf{B})||b_{ij}| \\
&= \left|\det\left(\mathbf{A}_{s_1}, \ldots, \mathbf{A}_{s_{i-1}}, \mathbf{b}, \mathbf{A}_{s_{i+1}}, \ldots, \mathbf{A}_{s_m}\right)\right| \\
&\leq \|\mathbf{b}\| \prod_{k=1, k\neq i}^{m} \|\mathbf{A}_{s_i}\| \\
&\leq m^{m/2} 2^{mN}. \qquad (3.36)
\end{aligned}$$

Finally, since $\mathbf{B}$ is a square submatrix of $\mathbf{A}$ of size m, by Eq. (3.35) the determinant of $\mathbf{B}$ satisfies $|\det(\mathbf{B})| \leq m^{m/2} 2^{mN}$. Hence

$$\begin{aligned}
|\widetilde{\overline{c}}_{j(m+1)j}| &= |\det(\mathbf{B})||\overline{c}_j| \\
&= \left|\det(\mathbf{B})\mathbf{c}_j - \mathbf{c_s}\left(\det(\mathbf{B})\mathbf{B}^{-1}\mathbf{A}_j\right)\right| \\
&\leq \left|\det(\mathbf{B})\mathbf{c}_j| + |\mathbf{c_s}\left(\widetilde{\mathbf{A}'}_j\right)\right| \\
&\leq m^{m/2} 2^{(m+1)N} + m 2^N \left(m^{m/2} 2^{mN}\right) \\
&= (m+1) m^{m/2} (2^{(m+1)N}), \qquad (3.37)
\end{aligned}$$

and, similarly, by Eq. (3.36) it follows that

$$\begin{aligned}
|\widetilde{-z}_{(m+1)(n+1)}| &= \left|-\mathbf{c_s}\det(\mathbf{B})\mathbf{B}^{-1}\mathbf{b}\right| \\
&\leq m 2^N \left(m^{m/2} 2^{mN}\right).
\end{aligned}$$

Hence all entries in $\widetilde{\mathbf{T}}$ satisfy Eq. (3.37), i.e.,

$$|\widetilde{t}_{ij}| \leq 2^{(m+1)N + \frac{m}{2}\log m + \log(m+1)} = 2^L,$$

where L satisfies Eq. (3.34). □

If $\mathbf{A}$ has sparse columns, i.e., the columns of $\mathbf{A}$ contain lots of zeros, then $|\det(\mathbf{B})|$ will be smaller. Also, the values in $\widetilde{\mathbf{T}}$ will be smaller. Theorem 3.35 presents an upper bound on the values in any tableau if every column of $\mathbf{A}$ contains at most s nonzero elements.

Theorem 3.35. *Given an LP in standard form, where the bit size of the coefficients* $\mathbf{A}$, $\mathbf{b}$, *and* $\mathbf{c}$ *is bounded by some positive integer* N. *Suppose furthermore that every column of* $\mathbf{A}$ *contains at most* γ *nonzero entries. Let* $\mathbf{x}$ *be a basic feasible solution* $\mathbf{x}$ *with basis* $\mathbf{s}$ *and basis matrix* $\mathbf{B}$. *Then its corresponding tableau* $\mathbf{T}$ *satisfies*

$$|\det(\mathbf{B})t_{ij}| < 2^L,$$

where

$$L = \begin{cases} (m+1)N + \frac{m}{2}\log(\gamma) + \log(m+1), & \text{if } \gamma \geq \frac{m^3}{(m+1)^2}, \\ (m+1)N + \frac{m-1}{2}\log(\gamma) + \frac{3}{2}\log(m), & \text{otherwise.} \end{cases} \tag{3.38}$$

Proof. We consider again the four parts of the tableau

$$\mathbf{T} = \begin{pmatrix} \mathbf{A}' & \mathbf{b}' \\ \overline{\mathbf{c}} & -z \end{pmatrix},$$

where $\mathbf{A}' = \mathbf{B}^{-1}\mathbf{A}$, $\mathbf{b}' = \mathbf{B}^{-1}\mathbf{b}$, $\bar{c} = \mathbf{c} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}$ and $z = \mathbf{c_s}\mathbf{B}^{-1}\mathbf{b}$.

Let $\widetilde{\mathbf{T}} = |\det(\mathbf{B})|\mathbf{T}$.

$$\begin{aligned} |\widetilde{a'}_{ij}| &= |\det(\mathbf{B})||a'_{ij}| \\ &\leq \|\mathbf{A}_j\| \prod_{k=1,k\neq i}^{m} \|\mathbf{A}_{s_k}\| \\ &= \gamma^{m/2}2^{mN}. \end{aligned} \tag{3.39}$$

Similarly,

$$\begin{aligned} |\widetilde{b'}_i| &= |\det(\mathbf{B})||b_{ij}| \\ &\leq \|\mathbf{b}\| \prod_{k=1,k\neq i}^{m} \|\mathbf{A}_{s_i}\| \\ &\leq m^{1/2}\gamma^{(m-1)/2}2^{mN}. \end{aligned} \tag{3.40}$$

Furthermore, $|\det(\mathbf{B})| \leq \gamma^{m/2}2^{mN}$ by Eq. (3.39). Hence

$$\begin{aligned} |\widetilde{\bar{c}_j}| &= |\det(\mathbf{B})||\bar{c}_j| \\ &= \left|\det(\mathbf{B})\mathbf{c}_j - \mathbf{c_s}\left(\det(\mathbf{B})\mathbf{B}^{-1}\mathbf{A}_j\right)\right| \\ &\leq \gamma^{m/2}2^{(m+1)N} + m2^N\left(\gamma^{m/2}2^{mN}\right) \\ &= (m+1)\gamma^{m/2}(2^{(m+1)N}), \end{aligned} \tag{3.41}$$

and, similarly, by Eq. (3.40) we have

$$\begin{aligned} |\widetilde{t}_{(m+1)(n+1)}| &= \left|-\mathbf{c_s}\det(\mathbf{B})\mathbf{B}^{-1}\mathbf{b}\right| \\ &\leq m^{3/2}\gamma^{(m-1)/2}2^{(m+1)N}. \end{aligned} \tag{3.42}$$

An upper bound on all values of $\widetilde{\mathbf{T}}$ is given by either Eq. (3.41) or Eq. (3.42). Note that this bound is given by Eq. (3.41) if and only if

$$(m+1)\gamma^{m/2}(2^{(m+1)N}) \geq m^{3/2}\gamma^{(m-1)/2}2^{(m+1)N},$$

i.e.,

$$m+1 \geq \sqrt{\frac{m}{\gamma}}m,$$

so,

$$\gamma \geq \left(\frac{m}{m+1}\right)^2 m.$$

Hence, all entries in $\widetilde{\mathbf{T}}$ satisfy Eq. (3.38). □

Algorithm 3.1, and Algorithm 3.4 when integer computations are required, are known as *large tableau simplex (LT)* algorithms. Algorithm 3.1 runs the simplex algorithm on the large tableau simplex with rational pivoting and is called LT-RP, while Algorithm 3.4 runs simplex on the large tableau with integer pivoting and is called LT-IP.

Remark 3.36. One would expect that the absolute value of the numbers with respect to integer pivoting will be larger than the absolute value of the numbers with respect to integer pivoting, since during the tableau updates the division with the pivot element is postponed to the next iteration.

However, below we provide an LP, where there exists a basis $\mathbf{s}$ such that the tableaus with respect to both integer pivoting and rational pivoting are equally large. Furthermore it has an entry t_{ij} , where $\log(|t_{ij}|)$ is close to L, the bound on the bit size of the numbers with respect to integer pivoting.

Consider the following LP

$$\begin{array}{rccccccc} \min & 7x_1 & - & 7x_2 & + & 7x_3, & & \\ \text{subject to} & 5x_1 & + & 7x_2 & - & 7x_3 & \leq & 7, \\ & 2x_1 & - & 7x_2 & - & 3x_3 & \leq & 1, \\ \text{and} & x_1 \geq 0, & & x_2 \geq 0, & & , x_3 \geq 0. & & \end{array}$$

Note that the absolute value of each coefficient is smaller than $8 = 2^3$. It follows that

$$L = (m+1)N + m/2 \log m + \log(m+1) = 9 + 1 + \log 3.$$

Let $\mathbf{s} = (1, 3)$, then

$$\mathbf{B} = \mathbf{A}_\mathbf{s} = \begin{pmatrix} 5 & -7 \\ 2 & -3 \end{pmatrix}.$$

Observe that

$$\mathbf{B}^{-1} = \begin{pmatrix} 3 & -7 \\ 2 & -5 \end{pmatrix}.$$

The tableau $\mathbf{T}$ with respect to $\mathbf{s}$ is given by

$$\begin{aligned} \mathbf{ZT}^0 &= \begin{pmatrix} 3 & -7 & 0 \\ 2 & -5 & 0 \\ -35 & 84 & 1 \end{pmatrix} \begin{pmatrix} 5 & 7 & -7 & 1 & 0 & 7 \\ 2 & -7 & -3 & 0 & 1 & 1 \\ -7 & -7 & 7 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 70 & 0 & 3 & -7 & 14 \\ 0 & 49 & 1 & 2 & -5 & 9 \\ 0 & -840 & 0 & -35 & 84 & -161 \end{pmatrix}. \end{aligned}$$

Since $\det(\mathbf{B}) = -1$ it holds that $\widetilde{\mathbf{T}} = \mathbf{T}$, where we have for the largest value

$$\lceil \log(|-840|) \rceil = 10.$$

◇

The next sections will show how to derive from Propositions 3.29 and 3.31 variants, where the tableaus will be smaller.

3.2.2 Small Tableau Simplex

The *small tableau simplex (ST)* algorithms improve the efficiency of the large tableau simplex algorithms by using the following properties of $\mathbf{T}$.

Lemma 3.37. *Given an LP in standard form, let* $\mathbf{T}$ *be as defined in Definition 3.28 corresponding to basis* $\mathbf{s}$*. Then*

(i)

$$\mathbf{T_s} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & 1 \\ 0 & 0 & \dots & 0 \end{pmatrix}. \tag{3.43}$$

(ii) Let $\mathbf{T}'$ *be the tableau after one iteration of the simplex method with basis* $\mathbf{s}' = (s_1, \dots, s_{k-1}, \ell, s_{k+1}, \dots, s_m)$ *using rational pivoting. Then*

$$\mathbf{T'_s} = \begin{pmatrix} 1 & 0 & \dots & -\frac{t_{1\ell}}{t_{k\ell}} & \dots & 0 \\ 0 & 1 & \dots & -\frac{t_{2\ell}}{t_{k\ell}} & \dots & 0 \\ & & \ddots & & & \\ & & & \frac{1}{t_{k\ell}} & & \\ & & & & \ddots & \\ 0 & 0 & \dots & -\frac{t_{m\ell}}{t_{k\ell}} & \dots & 1 \\ 0 & 0 & \dots & -\frac{t_{(m+1)\ell}}{t_{k\ell}} & \dots & 0 \end{pmatrix}. \tag{3.44}$$

(iii) Let $\widetilde{\mathbf{T}'}$ *be the tableau after one iteration of the simplex method with basis* $\mathbf{s}' = (s_1, \dots, s_{k-1}, \ell, s_{k+1}, \dots, s_m)$ *using integer pivoting. Then,*

$$\widetilde{\mathbf{T}'}_{\mathbf{s}} = \begin{pmatrix} \widetilde{t}_{k\ell} & 0 & \dots & -\widetilde{t}_{1\ell} & \dots & 0 \\ 0 & \widetilde{t}_{k\ell} & \dots & -\widetilde{t}_{2\ell} & \dots & 0 \\ & & \ddots & & & \\ & & & |\det(\mathbf{A_s})| & & \\ & & & & \ddots & \\ 0 & 0 & \dots & -\widetilde{t}_{m\ell} & \dots & \widetilde{t}_{k\ell} \\ 0 & 0 & \dots & -\widetilde{t}_{(m+1)\ell} & \dots & 0 \end{pmatrix}. \tag{3.45}$$

Proof. (i) Let $\mathbf{B} = \mathbf{A_s}$ be the basis matrix. By definition

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix}.$$

Hence

$$\mathbf{T_s} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A_s} \\ \mathbf{c_s} \end{pmatrix} = \begin{pmatrix} \mathbf{B}^{-1}\mathbf{B} \\ \mathbf{c_s} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{B} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_m \\ \mathbf{0} \end{pmatrix}.$$

(ii) Theorem 3.29 implies

$$\mathbf{T'}_{\mathbf{s}} = (\mathbf{QT})_{\mathbf{s}} = \mathbf{QT}_{\mathbf{s}}.$$

Applying (i) to $\mathbf{T}_{\mathbf{s}}$ yields

$$\mathbf{T'}_{\mathbf{s}} = \mathbf{Q}\begin{pmatrix} \mathbf{I}_m \\ \mathbf{0} \end{pmatrix} = (\mathbf{Q}_1 \dots \mathbf{Q}_m)\,.$$

Then Eq. (3.20) in the proof Theorem 3.29 implies Eq. (3.44).

(iii) Theorem 3.31 implies that

$$\widetilde{\mathbf{T}'} = \det(\mathbf{A}_{\mathbf{s}'})\mathbf{T}' = \widetilde{t}_{k\ell}\mathbf{T}'.$$

Hence (ii) applied to $\mathbf{T}'$ provides Eq. (3.45).

□

It follows that $(\mathbf{t}_{\mathbf{s}})_{m+1} = \overline{\mathbf{c}}_{\mathbf{s}} = 0$ and, therefore, no $\ell \in \mathbf{s}$ will be selected for pivoting. Furthermore, when a pivot element $t_{k\ell}$ is selected, then pivoting yields $\mathbf{T}'_{s_i} = \mathbf{T}_{s_i}$, $\mathbf{T}'_{\ell} = \mathbf{T}_{s_k}$, and $\mathbf{T}'_{s_k} = \mathbf{Q}'_k$, which can be computed independently from $\mathbf{T}_{\mathbf{s}}$. Hence one can remove the columns $\mathbf{T}_{\mathbf{s}}$ from consideration. Similarly, in the next iteration the columns $\mathbf{T'}_{\mathbf{s}'}$ can be removed from consideration.

Definition 3.38. *Let* $\mathbf{T}$ *be the tableau corresponding to an LP in standard form and basis* $\mathbf{s}$*. Let* $\mathbf{u}$ *be the co-basis. Then*

$$\mathbf{T}_{(\mathbf{u},n+1)} = \left(\mathbf{T}_{u_1} \dots \mathbf{T}_{u_{n-m}} \mathbf{T}_{n+1}\right) \tag{3.46}$$

is called the condensed tableau *or* small tableau*.*

Small tableau simplex iterates using Algorithm 3.1 or Algorithm 3.4 on tableau $\mathbf{T}_{(\mathbf{u},n+1)}$ with the following two additions: (*i*) that after each pivoting the ℓ-th column of $\mathbf{T}_{(\mathbf{u},n+1)}$ is replaced by $\mathbf{Q}'_k$ and (*ii*) the basis and co-basis are updated by swapping s_k with u_ℓ. Small tableau simplex using rational pivoting is called ST-RP and small tableau simplex using integer pivoting is called .

Algorithm 3.6 gives a precise description of the small tableau simplex iterations.

3.2.3 Revised Simplex

The *revised simplex* is based on the observation that by Definition 3.28 any tableau $\mathbf{T}$, with respect to basis $\mathbf{s}$, can be written as the product of two matrices, where one is invariant during all iterations.

Lemma 3.39. *Let an LP in standard form be given. Let*

$$\mathbf{T}^0 = \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix}$$

and

$$\mathbf{D} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}_{\mathbf{s}}\mathbf{B}^{-1} & 1 \end{pmatrix}.$$

Algorithm 3.6: $(\mathbf{T}, \mathbf{s}, \text{pred}, \mathbf{u}, q) \leftarrow \mathsf{Iterate}_{\mathsf{ST},\mathsf{VAR}}(\mathbf{T}, \mathbf{s}, \mathbf{u}, q)$

Input: $\mathbf{T}$, $\mathbf{s}$, $\mathbf{u}$, q.
Output: $\mathbf{T}$, $\mathbf{s}$, $\mathbf{u}$, q, pred.

1 $(\ell, k) \leftarrow \mathsf{FindPivotElement}(\mathbf{T})$;
2 **if** $\ell = 0$ **then**
3 **return** $(\mathbf{T}, \mathbf{s}, \mathsf{Optimal}, \mathbf{u}, q)$;
4 **else if** $k = 0$ **then**
5 **return** $(\mathbf{T}, \mathbf{s}, \mathsf{UnboundedLP}, \mathbf{u}, q)$;
6 $(\mathbf{T}', q') \leftarrow \mathsf{Pivot}_{\mathsf{VAR}}(\mathbf{T}, \ell, k, q)$;

$\mathsf{VAR} = \mathsf{RP}$:
7a **foreach** $i \in \{1, \ldots, m+1\}$ **do**
8a $t'_{i\ell} = -\frac{t_{i\ell}}{t_{k\ell}}$;
9a $t'_{k\ell} = \frac{1}{t_\ell}$;

$\mathsf{VAR} = \mathsf{IP}$:
7b **foreach** $i \in \{1, \ldots, m+1\}$ **do**
8b $t'_{i\ell} = -t_{i\ell}$;
9b $t'_{k\ell} = q$;

10 $s_k \leftrightarrow u_\ell$;
11 **return** $\mathsf{IterateST}_{\mathsf{VAR}}(\mathbf{T}', \mathbf{s}, \mathbf{u}, q')$;

Suppose that $\mathbf{T} = \mathbf{D}\mathbf{T}^0$ *is the corresponding tableau with respect to basis* $\mathbf{s}$ *at the beginning of an iteration of the simplex algorithm and that the basis is updated, where* ℓ *enters the basis and* s_k *leaves the basis. Let* $\mathbf{T}' = \mathbf{Q}\mathbf{T}$ *be the tableau with respect to basis* $\mathbf{s}' = (s_1, \ldots, s_{k-1}, \ell, s_{k+1}, \ldots, s_m)$*. Then,* $\mathbf{T}' = \mathbf{D}'\mathbf{T}^0$*, where*

$$\mathbf{D}' = \mathbf{Q}\mathbf{D}. \tag{3.47}$$

Proof. From Theorem 3.29 it follows that the tableau with respect to basis $\mathbf{s}'$ is given by $\mathbf{T}' = \mathbf{Q}\mathbf{T}$, so

$$\mathbf{T}' = \mathbf{Q}\mathbf{D}\mathbf{T}^0 = \mathbf{D}'\mathbf{T}^0.$$

Next, let $\mathbf{z} = (z_1, \ldots, z_{m+1})$ be such that $(\mathbf{T}^0)_\mathbf{z}$ is invertible, then

$$\mathbf{T}'_\mathbf{z} = \mathbf{Q}\mathbf{D}(\mathbf{T}^0)_\mathbf{z} = \mathbf{D}'(\mathbf{T}^0)_\mathbf{z}$$

and thus that multiplying by the inverse of $(\mathbf{T}^0)_\mathbf{z}$ results in Eq. (3.47).

Such $\mathbf{z}$ exists if $\mathbf{T}^0$ has full row rank. So suppose that $\mathbf{T}^0$ has not full row rank. Hence the rows of $\mathbf{T}^0$ are linearly dependent. Since a basis exists it follows that $\mathbf{A}$ has full row rank. Hence, there exist $(\lambda_1, \ldots, \lambda_m) \in \mathbb{R}^m$ such that

$$\sum_{i=1}^{m} \lambda_i \mathbf{a}_i = \mathbf{c}$$

and

$$\sum_{i=1}^{m} \lambda_i b_i = 0.$$

Thus, for all feasible $\mathbf{x}$ it holds that

$$\mathbf{cx} = \sum_{i=1}^{m} \lambda_i \mathbf{a}_i \mathbf{x} = \sum_{i=1}^{m} \lambda_i b_i = 0.$$

□

The revised simplex updates $\mathbf{D}$ only, instead of tableau $\mathbf{T}$. By remark 3.2 it follows that $m \leq n$. In practice, $n \gg m$ and, therefore, updating $\mathbf{D}$ is more efficient than updating $\mathbf{T}$ instead. On the other hand, values required each simplex iteration such as $\overline{\mathbf{c}}$ need to be computed each iteration from $\mathbf{D}$ and $\mathbf{T}^0$.

The revised simplex algorithm using rational pivoting is called RS-RP and the revised simplex algorithm using integer pivoting is called RS-IP. The following algorithms specify the revised simplex iterations.

Algorithm 3.7: $\mathbf{D}' \leftarrow \mathsf{Pivot}_{\mathsf{RS,RP}}(\mathbf{D}, k, \ell, \mathbf{T}^0)$

Input: $\mathbf{D}$, $\mathbf{v}$, ℓ, k.
Output: $\mathbf{D}'$.

$\mathbf{v} \leftarrow \mathbf{D}\mathbf{T}^0_\ell$;
foreach $i \in \{1, \ldots, m+1\}$ **do**
  foreach $j \in \{1, \ldots, m+1\}$ **do**
    if $i \neq k$ **then**
      $d'_{ij} \leftarrow d_{ij} - \frac{d_{kj} v_i}{v_k}$;
    else
      $d'_{kj} \leftarrow \frac{d_{kj}}{v_k}$;
return $\mathbf{D}'$;

Algorithm 3.8: $(\mathbf{D}, \mathbf{s}, \mathrm{pred}, q) \leftarrow \mathsf{Iterate}_{\mathsf{RS,VAR}}(\mathbf{D}, \mathbf{s}, \mathbf{T}^0, q)$

Input: $\mathbf{D}, \mathbf{T}^0, \mathbf{s}$, q.
Output: $\mathbf{D}$, $\mathbf{s}$, pred.

$(\ell, k, \mathbf{v}) \leftarrow \mathsf{FindPivotElement}_{\mathsf{RS}}(\mathbf{D}, \mathbf{T}^0)$;
if $\ell = 0$ **then**
  return $(\mathbf{D}, \mathbf{s}, \mathsf{Optimal})$;
else if $k = 0$ **then**
  return $(\mathbf{D}, \mathbf{s}, \mathsf{UnboundedLP})$;
$\mathsf{VAR} = \mathsf{RP}$:
  $\mathbf{D} \leftarrow \mathsf{Pivot}_{\mathsf{RS,RP}}(\mathbf{D}, \ell, k, \mathbf{T}^0)$;
$\mathsf{VAR} = \mathsf{IP}$:
  $(\mathbf{D}, q) \leftarrow \mathsf{Pivot}_{\mathsf{RS,IP}}(\mathbf{D}, \ell, k, \mathbf{T}^0 q)$;
$s_k \leftarrow \ell$;
return $\mathsf{IterateRS}_{\mathsf{VAR}}(\mathbf{D}, \mathbf{s}, \mathbf{T}^0, q)$;

Algorithm 3.9: $(\ell, k, \mathbf{v}) \leftarrow \mathsf{FindPivotElement}_{\mathsf{RS}}(\mathbf{D}, \mathbf{T}^0)$

Input: $\mathbf{D}$, $\mathbf{T}^0$.
Output: ℓ, k, $\mathbf{v}$.

$\mathbf{c} \leftarrow \mathbf{d}_{m+1} \cdot \mathbf{T}^0$;
$\ell \leftarrow \operatorname{argmin}\{c_i | \, i \in \{1, \ldots, n\}\}$;
if $c_\ell \geq 0$ **then**
    return $(0, 0)$;
$\mathbf{v} \leftarrow \mathbf{D}\mathbf{T}^0_\ell$;
$\mathbf{b} \leftarrow \mathbf{D}\mathbf{T}^0_{n+1}$;
foreach $i \in \{1, \ldots, m\}$ **do**
    if $v_i > 0$ **then** $r_i \leftarrow \frac{b_i}{v_i}$;
    else $r_i \leftarrow \infty$;
$\mathbf{R} \leftarrow (r_1, \ldots, r_m)$;
$k \leftarrow \operatorname{argmin}(\mathbf{R})$;
if $r_k = \infty$ **then**
    return $(\ell, 0)$;
return (ℓ, k);

Algorithm 3.10: $(\mathbf{D}, q) \leftarrow \mathsf{Pivot}_{\mathsf{RS},\mathsf{IP}}(\mathbf{D}, k, \ell, \mathbf{T}^0, q)$

Input: $\mathbf{D}$, $\mathbf{v}$, ℓ, k, q.
Output: $\mathbf{D}$, q.

$\mathbf{v} \leftarrow \mathbf{D}\mathbf{T}^0_\ell$;
foreach $i \in \{1, \ldots, m+1\}$ **do**
    foreach $j \in \{1, \ldots, m+1\}$ **do**
        if $i \neq k$ **then**
            $d'_{ij} \leftarrow \frac{d_{ij}v_k - d_{kj}v_i}{q}$;
        else
            $d'_{kj} \leftarrow v_j$;
$q = v_k$;
return $(\mathbf{D}', q)$;

3.3 Implementations of the Simplex Initializations

The previous section presented how to implement the simplex iterations. This section presents how to initialize the iterations of the simplex algorithm.

In Section 3.1.3 we showed how to find an initial feasible solution to a given LP in standard form. The simplex algorithm, however, requires the solution to be basic. This section presents the following algorithms to find an initial basic feasible solution given any LP in standard form, the *two-phase simplex algorithm* and the *big-M* method.

The two-phase simplex algorithm runs in two-phases. In the first phase it solves an artificial LP from which it initializes the second phase to solve the original LP. Special attention will be given to the initialization of the two-phases, since the iterations can be done by the algorithms from the previous sections.

The big-M method, on the other hand, merges the artificial LP and the original LP into a new linear program where the optimum is either a proof that the given LP is infeasible, or it is optimal the given LP as well. We show that the iterations can in some cases also be done by the algorithms described in previous sections, or they require some extension.

3.3.1 Standard two-phase Simplex

Suppose that an LP is given in standard form and suppose that $\mathbf{b} \geq \mathbf{0}$. In this section we consider the following artificial LP:

$$\begin{array}{lrcl} \min & \mathbf{c}'\mathbf{x} = \sum_{i=1}^{m} x_{n+i}, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix} \mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{3.48}$$

where $x_{n+1}, \ldots, x_{n+m}$ are the *artificial variables*.

The *two-phase simplex algorithm* is as follows:

Phase I: Given any LP in standard form solve the corresponding artificial LP. Decide whether the given LP is feasible; if not return "infeasible LP" else goto phase II.

Phase II: Given the optimal solution and basis of the artificial LP, compute a basic feasible solution to the original LP and a corresponding basis. Then solve the original LP using simplex.

With respect to phase I, we will show how to initialize any simplex algorithm of the previous sections to solve the artificial LP. Then, if the original LP is feasible, we show how to use the result of phase I to initialize phase II. The latter consists of transforming the optimal basis for the artificial LP into a basis corresponding to a feasible solution for the original LP and computing a corresponding tableau.

Note that in the previous sections we required that $\mathbf{A}$ has full row rank m. The artificial LP has by definition full row rank m since the last columns form $\mathbf{I}_m$. Hence we could drop the assumption on $\mathbf{A}$ to have full row rank. We show how to find and remove redundant constraints of the original LP from the result of phase I and compute a consistent tableau corresponding the LP where those constraints are removed.

More precisely, we need to show how to transform the tableau $\mathbf{T}$ returned by phase I into a tableau $\mathbf{T}'$ that corresponds to a basic feasible solution for the original LP. We show that the basis transformations can be done by pivot operations. Next, we show how to modify $\mathbf{T}$ so that the redundant constraints are removed. Then we show how to delete the columns of $\mathbf{T}'$ corresponding the artificial variables. Lastly, we show how to compute the last row of $\mathbf{T}'$ so that in the end $\mathbf{T}'$ is a tableau for the original LP by Definition 3.28, which will, together with the modified basis $\mathbf{s}'$, be the input for phase II.

Initializing Phase I

To initialize phase I, observe that $\mathbf{s} = (n+1, \ldots, n+m)$ is a basis and $\mathbf{u} = (1, \ldots, n)$ is a co-basis. Hence the vector $\mathbf{x}$ satisfying

$$\mathbf{x}_\mathbf{s} = (\mathbf{I}_m)^{-1}\mathbf{b} = \mathbf{b}$$

and $\mathbf{x}_\mathbf{u} = \mathbf{0}$ is a basic feasible solution with respect to basis $\mathbf{s}$.

Algorithm 3.11: $(\mathbf{x}, \text{pred}) \leftarrow \mathsf{TwoPhaseSimplex}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{A}, \mathbf{b}, \mathbf{c})$

Input: $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$
Output: $\mathbf{x}$, pred
1 $(\mathbf{T}, \mathbf{T}^0, \mathbf{s}, \mathbf{u}, q) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{A}, \mathbf{b})$;
2 $(\mathbf{T}, \mathbf{s}, \text{pred}, \mathbf{u}, q) \leftarrow \mathsf{Iterate}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{s}, \mathbf{T}^0, \mathbf{u}, q)$;
$\mathsf{VAR}_1 = \mathsf{LT}, \mathsf{ST}$:
3a $t \leftarrow t_{m+1,n+1}$;
$\mathsf{VAR}_1 = \mathsf{RS}$:
3b $t \leftarrow \mathbf{t}_{m+1}\mathbf{T}^0_{n+1}$;
4 **if** $t < 0$ **then**
5 **return** $(\mathbf{0}, \text{pred})$;
6 $(\mathbf{T}, \mathbf{T}^0, \mathbf{s}, \mathbf{u}, q) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{s}, \mathbf{c}, \mathbf{T}^0, \mathbf{u}, q)$;
7 $(\mathbf{T}, \mathbf{s}, \text{pred}, \mathbf{u}, q) \leftarrow \mathsf{Iterate}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{s}, \mathbf{T}^0, \mathbf{u}, q)$;
8 **return** $(\mathsf{GetSolution}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{s}, \mathbf{T}^0, q), \text{pred})$

With respect to the costs, observe that $\mathbf{c}'_{\mathbf{s}} = \mathbf{1}$, the all-one vector. Hence

$$\mathbf{T} = \begin{pmatrix} \mathbf{I}_m & \mathbf{0} \\ -\mathbf{1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{0} & \mathbf{1} & 0 \end{pmatrix}$$

is the tableau corresponding to basis $\mathbf{s}$, from which any simplex variant of Section 3.2 can be initialized. Algorithm 3.12 presents how to initialize phase I.

Basis Transformations

If cycling is avoided, then by Theorem 3.19 a basic optimal solution $\mathbf{x}$ and a basis $\mathbf{s}$ is returned by phase I. If $\mathbf{c}'\mathbf{x} = 0$, the solution $x_1, \ldots, x_n$ is feasible to the original LP. However, $\mathbf{s}$ may be not a basis to the original LP. We show how to compute a new basis $\mathbf{s}'$ that corresponds to $\mathbf{x}$ and is also a basis to the original LP using Theorem 3.42.

Remark 3.40. Note that by construction, the artificial LP (3.48) has full rank. Hence, two-phase simplex will be able to solve any feasible and bounded LP even if $\mathbf{A}$ is not of full rank.

Lemma 3.41. *Let $\mathbf{B} \in \mathbb{R}^{m\times m}$ be an invertible matrix. If $\mathbf{B}_k = \mathbf{e}_j$ then $(\mathbf{B}^{-1})_j = \mathbf{e}_k$.*

Proof. Suppose that $\mathbf{B}_k = \mathbf{e}_j$. From $\mathbf{B}^{-1}\mathbf{B} = \mathbf{I}_m$ it follows that

$$(\mathbf{B}^{-1})_j = \mathbf{B}^{-1}\mathbf{e}_j = \mathbf{B}^{-1}\mathbf{B}_k = \mathbf{e}_k.$$

□

Theorem 3.42. *Consider an LP in standard form, where $\mathbf{b} \geq \mathbf{0}$, and its corresponding artificial LP. Suppose that phase I returns a basic optimal solution $\mathbf{x}$ with basis $\mathbf{s}$, where $\mathbf{c}'\mathbf{x} = 0$. Let $\mathcal{D}$ be the set of indices in the basis of value larger than n, i.e.,*

$$\mathcal{D} = \{n+1, \ldots, n+m\} \cap \{s_1, \ldots, s_m\}.$$

If $\mathcal{D} = \emptyset$, then $\hat{\mathbf{x}} = (x_1, \ldots, x_n)$ is basic to the original LP with basis $\mathbf{s}$. If $\mathcal{D} \neq \emptyset$, then for any $s_k \in \mathcal{D}$ either

Algorithm 3.12: $(\mathbf{T}, \mathbf{s}, \mathbf{T}^0, \mathbf{u}, q) \leftarrow \mathsf{InitializePhasel}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{A}, \mathbf{b})$

Input: $\mathbf{A}$, $\mathbf{b}$
Output: $\mathbf{T}$, $\mathbf{s}$, $\mathbf{T}^0$, $\mathbf{u}$, q.

1 $\mathbf{T}^0 \leftarrow \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{0} & \mathbf{1} & 0 \end{pmatrix}$;

2 $\mathbf{D} \leftarrow \begin{pmatrix} \mathbf{I}_m & \mathbf{0} \\ -\mathbf{1} & 1 \end{pmatrix}$;

3 $\mathbf{s} \leftarrow (n+1, \ldots, n+m)$;

$\mathsf{VAR}_2 = \mathsf{IP}$:

4a $q = 1$;

$\mathsf{VAR}_1 = \mathsf{LT}$:

5a $\mathbf{T} \leftarrow \mathbf{D}\mathbf{T}^0$;

6a **return** $(\mathbf{T}, \mathbf{s}, q)$;

$\mathsf{VAR}_1 = \mathsf{ST}$:

5b $\mathbf{u} \leftarrow (1, \ldots, n)$;

6b $\mathbf{T} \leftarrow \mathbf{D}\mathbf{T}^0_{(\mathbf{u}, n+m+1)}$;

7b **return** $(\mathbf{T}, \mathbf{s}, \mathbf{u}, q)$;

$\mathsf{VAR}_1 = \mathsf{RS}$:

5c $\mathbf{T} \leftarrow \mathbf{D}$;

6c **return** $(\mathbf{T}, \mathbf{s}, \mathbf{T}^0, q)$;

(i) there exists an $\ell \in \{1, \ldots, n\}$ such that $\mathbf{s}' = (s_1, \ldots, s_{k-1}, \ell, s_{k+1}, \ldots, s_m)$ is also a basis for $\mathbf{x}$, or

(ii) the $(s_k - n)$'th constraint is redundant in the original LP.

Proof. Let $\hat{\mathbf{x}} = (x_1, \ldots, x_n)$. Observe that $\mathbf{x}$ satisfies

$$\mathbf{A}'\mathbf{x} = \mathbf{b},$$

where

$$\mathbf{A}' = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}.$$

If $\mathcal{D} = \emptyset$, then

$$\mathbf{A}'\mathbf{x} = \mathbf{A}'_{\mathbf{s}}\mathbf{x}_{\mathbf{s}} = \mathbf{A}_{\mathbf{s}}\hat{\mathbf{x}}_{\mathbf{s}}.$$

Hence $\hat{\mathbf{x}}$ is a basic feasible solution with basis $\mathbf{s} \in \{1, \ldots, n\}^m$ to the original LP.

If $\mathcal{D} \neq \emptyset$, then let $s_k \in \mathcal{D}$ and $\mathbf{B} = \mathbf{A}'_{\mathbf{s}}$.

(i) Suppose that $(\mathbf{B}^{-1}\mathbf{A})_{k\ell} \neq 0$ for some $\ell \in \{1, \ldots, n\}$. Then,

$$\mathbf{s}' = (s_1, \ldots, s_{k-1}, \ell, s_{k+1}, \ldots, s_m)$$

is a basis. Indeed from

$$\mathbf{B}^{-1}\mathbf{A}'_{\mathbf{s}'} = \begin{pmatrix} \mathbf{e}_1 & \ldots & \mathbf{e}_{k-1} & \mathbf{B}^{-1}\mathbf{A}_\ell & \mathbf{e}_{k+1} & \ldots & \mathbf{e}_m \end{pmatrix} \tag{3.49}$$

it follows that $\mathbf{A}'_{\mathbf{s}'}$ is invertible.

Let $\mathbf{T}$ be the tableau corresponding basis $\mathbf{s}$ and $\mathbf{T}'$ be the tableau corresponding to basis $\mathbf{s}'$. Then by Theorem 3.29 $\mathbf{T}' = \mathbf{QT}$ and

$$\mathbf{A}'^{-1}_{\mathbf{s}'} = \mathbf{Q}'\mathbf{B}^{-1},$$

where $\mathbf{Q}'$ is obtained from $\mathbf{Q}$ by removing the last row and column.

Since $\mathbf{x}$ is optimal to the artificial LP and $\mathbf{c}'\mathbf{x} = 0$, it follows that $\sum_{j=1}^{m} x_{n+j} = 0$. Hence $\mathbf{x} \geq \mathbf{0}$ implies that $x_{s_k} = 0$.

Recall that $\mathbf{Q}'_i = \mathbf{e}_i$ if $i \neq k$. Let $\mathbf{x}'$ be the basic feasible solution corresponding to basis $\mathbf{s}'$. Then

$$\mathbf{x}'_{\mathbf{s}'} = \mathbf{A}'^{-1}_{\mathbf{s}'}\mathbf{b} = \mathbf{Q}'\mathbf{B}^{-1}\mathbf{b} = \mathbf{Q}'\mathbf{x}_{\mathbf{s}} = \mathbf{x}_{\mathbf{s}} + \mathbf{Q}'_k x_{s_k} = \mathbf{x}_{\mathbf{s}}.$$

Hence, $\mathbf{x}' = \mathbf{x}$ and $\mathbf{s}'$ is a basis for $\mathbf{x}$.

(ii) If $(\mathbf{B}^{-1}\mathbf{A})_{k\ell} = 0$ for all $\ell \in \{1, \ldots, n\}$, then the k'th row of $\mathbf{B}^{-1}\mathbf{A} = \mathbf{0}$. Let $j = s_k - n$. Then

$$\mathbf{B}_k = \mathbf{A}'_{s_k} = \mathbf{A}_{n+j} = \mathbf{e}_j.$$

By Lemma 3.41 it follows that the j-th column of $\mathbf{B}^{-1}$ is equal to $\mathbf{e}_k$. Let $\boldsymbol{\lambda}$ be the k-th row of $\mathbf{B}^{-1}$. It follows that $\lambda_j = 1$. From $\boldsymbol{\lambda}\mathbf{A} = \mathbf{0}$ and $\boldsymbol{\lambda}\mathbf{b} = x'_{s_k} = 0$ it follows that the constraint $\mathbf{a}_j\mathbf{x} = b_j$ is linearly dependent of the other constraints $\mathbf{a}_i\mathbf{x} = b_i$ or, simply, redundant. □

Algorithm 3.13 presents how to transform the result of phase I into a tableau and basis initializing phase II using Theorem 3.42. Note that these basis transformations can include pivoting on a negative entry. By Theorem 3.29 this can be implemented by a simple pivot operation when the standard pivoting operations are applied (RP variants).

However for the integer pivoting simplex variants, we need to be careful. It follows from Theorem 3.31 that if the pivot element is negative after the pivot operation the tableau is multiplied by -1. Theorem 3.43 shows that to avoid this one could ensure that the whole tableau is multiplied by the sign of the pivot element. This can be done simply by multiplying the pivot row with the sign of the pivot element before pivoting.

Indeed, consider integer pivoting in tableau $\mathbf{T}$ on element $t_{k\ell}$. Let α be the sign of $t_{k\ell}$. Then integer pivoting in $\mathbf{T}$ on $t_{k\ell}$ implies that every entry t_{ij} outside the pivot row is updated by $(t_{ij}t_{k\ell} - t_{i\ell}t_{kj})/q$. However, if row k is multiplied with α first, then updating t_{ij} becomes

$$\frac{t_{ij}\alpha t_{k\ell} - t_{i\ell}\alpha t_{kj}}{q} = \alpha\frac{t_{ij}t_{k\ell} - t_{i\ell}t_{kj}}{q}.$$

Theorem 3.43. *Suppose that an LP in standard form is given, where all coefficients are integer. Let* $\mathbf{T}$ *be the tableau corresponding to basis* $\mathbf{s}$*. Consider basis*

$$\mathbf{s}' = (s_1, \ldots, s_{k-1}, \ell, s_{k+1}, \ldots, s_m).$$

Let tableau $\mathbf{T}'$ *be the tableau corresponding to basis* $\mathbf{s}'$*. Let* $\widetilde{\mathbf{T}} = |\det(\mathbf{B})|\mathbf{T}$ *and* $\widetilde{\mathbf{T}'} = |\det(\mathbf{B}')|\mathbf{T}'$*, where* $\mathbf{B} = \mathbf{A}_{\mathbf{s}}$ *and* $\mathbf{B}' = \mathbf{A}_{\mathbf{s}'}$ *and let* $\alpha = \operatorname{sgn}(\widetilde{t_{k\ell}})$*. Then*

$$\widetilde{t}'_{ij} = \alpha\frac{\widetilde{t}_{\ell k}\widetilde{t}_{ij} - \widetilde{t}_{i\ell}\widetilde{t}_{kj}}{|\det(\mathbf{B})|}, \qquad \text{if } i \neq k, \tag{3.50}$$

$$\widetilde{t}'_{kj} = \alpha\widetilde{t}_{kj}, \tag{3.51}$$

Algorithm 3.13: $(\mathbf{T}, \mathbf{s}, \mathbf{T}^0, \mathbf{u}, q) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{s}, \mathbf{c}, \mathbf{T}^0, \mathbf{u}, q)$

Input: $\mathbf{T}$, $\mathbf{c}$, $\mathbf{T}^0$, $\mathbf{s}$, $\mathbf{u}$, q
Output: $\mathbf{T}$, $\mathbf{T}^0$, $\mathbf{s}$, $\mathbf{u}$, q

1 **for** $k = m$ **down to** 1 **do**
  if $s_k > n$ **then**
2 $\ell \leftarrow 0$;
3 $\alpha \leftarrow 1$;
4 **repeat**
5 $\ell \leftarrow \ell + 1$;
  $\mathsf{Var}_1 = \mathsf{ST}$:
6a $\alpha \leftarrow u_\ell \leq n$;
  $\mathsf{VAR}_1 = \mathsf{RS}$:
6b $t_{k\ell} \leftarrow \mathbf{t}_k \mathbf{T}^0_\ell$;
7 $t \leftarrow \alpha t_{k\ell}$;
  until $t \neq 0$ *or* $\ell = n$;
8 **if** $\ell = 0$ **then**
9 $(\mathbf{T}, \mathbf{s}, \mathbf{T}^0) \leftarrow \mathsf{DeleteRowAndColumn}_{\mathsf{Var}_1}(\mathbf{T}, \mathbf{s}, k, \mathbf{T}^0)$;
10 **else**
  $\mathsf{VAR}_2 = \mathsf{IP}$*:*
11 $\mathbf{t}_k \leftarrow \mathrm{sgn}(t_{k\ell})\mathbf{t}_k$;
12 $(\mathbf{T}, \mathbf{s}, \mathbf{u}, q) \leftarrow \mathsf{Pivot}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{s}, \mathbf{T}^0, \mathbf{u}, q)$;
13 $(\mathbf{T}, \mathbf{T}^0, \mathbf{u}) \leftarrow \mathsf{DeleteColumn}_{\mathsf{VAR}_1}(\mathbf{T}, s_k, \mathbf{T}^0, \mathbf{u})$;
14 $(\mathbf{T}, \mathbf{T}^0) \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{c}, \mathbf{T}^0, \mathbf{s}, \mathbf{u})$;
return $\mathbf{T}, \mathbf{s}, \mathbf{T}^0, \mathbf{u}, q$;

Proof. Let $\alpha := \operatorname{sgn}(\widetilde{t_{k\ell}})$.

Recall from Eq. (3.33) in the proof of Theorem 3.31 that

$$\widetilde{t_{k\ell}} = \alpha' \det(\mathbf{B}') \neq 0,$$

where $\alpha' = \operatorname{sgn}(\det(\mathbf{B}))$. Observe that

$$\alpha = \alpha' \operatorname{sgn}(\det(\mathbf{B}')).$$

Hence applying Eq. (3.50) and Eq. (3.51) to $\widetilde{\mathbf{T}}$ and using the result of Theorem 3.29 to $\mathbf{T}$ and $\mathbf{T}'$ we get

$$\begin{aligned}
\widetilde{t'}_{ij} &= \alpha \frac{\widetilde{t}_{ij}\widetilde{t}_{k\ell} - \widetilde{t}_{i\ell}\widetilde{t}_{kj}}{|\det(\mathbf{B})|} \\
&= \alpha \widetilde{t}_{k\ell} \left(t_{ij} - \frac{t_{i\ell} t_{kj}}{t_{k\ell}} \right) \\
&= \alpha' \operatorname{sgn}(\det(\mathbf{B}')) \alpha' \det(\mathbf{B}') t'_{ij} \\
&= |\det(\mathbf{B}')| t'_{ij},
\end{aligned}$$

for all $i \neq k$ and

$$\begin{aligned}
\widetilde{t'}_{kj} &= \alpha \widetilde{t}_{kj} \\
&= \alpha \widetilde{t}_{k\ell} \frac{t_{ij}}{t_{k\ell}} \\
&= |\det(\mathbf{B}')| t'_{kj}.
\end{aligned}$$

Hence $\widetilde{\mathbf{T}}' = |\det(\mathbf{B}')| \mathbf{T}'$ as required. □

Removing Redundant Constraints

Theorem 3.44 shows how deletion of redundant constraints translates in deletion of rows and columns in tableau $\mathbf{T}$ and entries in $\mathbf{s}$. The resulting tableau $\mathbf{T}'$ will be a tableau for the equivalent linear program, where the redundant constraints are removed, corresponding to basis $\mathbf{s}'$.

Theorem 3.44. *Suppose that the given LP in standard form is redundant and feasible. Let $\mathbf{T}$ be a tableau for the corresponding artificial LP with respect to basis $\mathbf{s}$ and optimal solution $\mathbf{x}$. Let $\mathbf{s}$ be such that a minimal number of artificial variables are basic. Then*

(i) $\mathbf{t}_j = \mathbf{e}_{n+j}$ *for some* $j \in \{1, \ldots, m\}$,

(ii) $n + k = s_j$ *for some* $k \in \{1, \ldots, m\}$, *and*

(iii) if $\mathbf{T}'$ is equal to $\mathbf{T}$ where row j and column $n + k$ are removed and $\mathbf{s}'$ is equal to $\mathbf{s}$ where $n + k$ is removed, then $\mathbf{T}'$ is a tableau corresponding to the artificial LP where the j-th constraint is removed with respect to basis $\mathbf{s}'$.

Proof. Let $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{c}$ be the coefficients of the given LP in standard form. Suppose that $\mathbf{A}$ has rank $v < m$.

(i) Let $\mathbf{B}$ be the basis matrix corresponding $\mathbf{T}$. Observe that since $\mathbf{B}$ is invertible it contains at most v columns of $\mathbf{A}$. Hence by minimality of the number of basic artificial variables, it contains precisely v columns of $\mathbf{A}$. Assume without loss of generality that $\mathbf{s} = (1, \ldots, v, n+v+1, \ldots, n+m)$. Hence

$$\mathbf{B} = \begin{pmatrix} \mathbf{A}_1 & \cdots & \mathbf{A}_v & \mathbf{e}_{v+1} & \cdots & \mathbf{e}_m \end{pmatrix}.$$

For $i = 1, \ldots, m-v$ let $\boldsymbol{\lambda}_i$ be the $(v+i)$-th row of $\mathbf{B}^{-1}$. Then by $\mathbf{B}\mathbf{B}^{-1} = \mathbf{I}_m$ for all i it follows that $\boldsymbol{\lambda}_i \mathbf{A}_j = 0$ for all $j = 1, \ldots, v$. The rank of the first v columns of $\mathbf{A}$ is equal to the rank of $\mathbf{A}$. Hence, for all i

$$\boldsymbol{\lambda}_i \mathbf{A} = \mathbf{0}. \tag{3.52}$$

Since the original LP is feasible $\mathbf{c}'_{\mathbf{s}}\mathbf{x}_{\mathbf{s}} = 0$, where $\mathbf{c}'$ denotes the cost vector of the artificial LP. Hence

$$\mathbf{c}'_{\mathbf{s}}\mathbf{B}^{-1}\mathbf{b} = \sum_{i=1}^{m-v} \boldsymbol{\lambda}_i b_i = 0. \tag{3.53}$$

By $\mathbf{x} \geq \mathbf{0}$ we have $\mathbf{B}^{-1}\mathbf{b} \geq \mathbf{0}$. So, by Eq. (3.53) $\boldsymbol{\lambda}_i \mathbf{b} = 0$ for all $i = 1, \ldots, m-v$.

Let $\mathbf{T}$ be the tableau corresponding to basis $\mathbf{s}$ with respect to the artificial LP. Then

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}'_{\mathbf{s}}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{0} & \mathbf{1} & 0 \end{pmatrix}.$$

Hence for $i = v+1, \ldots, m$ each row i satisfies $\mathbf{t}_i = \mathbf{e}_{n+i}$. Observe that in addition $n+i \in \mathbf{s}$, so (ii) follows.

(iii) Let

$$\mathbf{D} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}_{\mathbf{s}}\mathbf{B}^{-1} & 1 \end{pmatrix}$$

and

$$\mathbf{T}^0 = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{0} & \mathbf{1} & 0 \end{pmatrix}.$$

Let furthermore $\mathbf{D}'$ be equal to $\mathbf{B}$ where the m-th row and column is removed and $\mathbf{T}'^0$ be equal to $\mathbf{T}^0$ where the m-th column is removed. Then $\mathbf{T}' = \mathbf{D}'\mathbf{T}'^0$ is equal to $\mathbf{T}$ where the m-th row and $(n+m)$-th column is removed. Furthermore it is a tableau for the following linear program

$$\begin{array}{lrcl} \min & \sum_{i=n+1}^{n+m-1} x_i, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A}' & \mathbf{I}_{m-1} \end{pmatrix} \mathbf{x} & = & \mathbf{b}', \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{3.54}$$

where $\mathbf{A}'$ is the matrix obtained from $\mathbf{A}$ by deleting row m and $\mathbf{b}'$ is obtained from $\mathbf{b}$ by deleting entry m. The corresponding basis is $\mathbf{s}' = (1, \ldots, v, n+v+1, \ldots, n+m-1)$. □

Algorithm 3.14 shows how to delete a redundant constraint based on Theorem 3.44.

Finishing the Transformation

If there are no redundant constraints and phase I returns a tableau $\mathbf{T}$ and a basis corresponding to a basic feasible solution to the original LP, the Lemma 3.45 shows how to transform $\mathbf{T}$ to a tableau corresponding to the original LP.

Algorithm 3.14: $(\mathbf{T}, \mathbf{s}, \mathbf{T}^0) \leftarrow \mathsf{DeleteRowAndColumn}_{\mathsf{Var}_1}(\mathbf{T}, \mathbf{s}, k, \mathbf{T}^0)$

Input: $\mathbf{T}$, $\mathbf{s}$, k, $\mathbf{T}^0$
Output: $\mathbf{T}$, $\mathbf{s}$, $\mathbf{T}^0$

1 $\mathbf{s}' \leftarrow (s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_{m'})$;
2 $j \leftarrow s_k - n$;

$\mathsf{VAR}_1 = \mathsf{LT}$:
3a $\mathbf{T}' \leftarrow \mathsf{DeleteRow}(\mathbf{T}, k)$;
4a $\mathbf{T}'' \leftarrow \mathsf{DeleteColumn}(\mathbf{T}', n + j)$;

$\mathsf{VAR}_1 = \mathsf{ST}$:
3b $\mathbf{T}' \leftarrow \mathsf{DeleteRow}(\mathbf{T}, k)$;

$\mathsf{VAR}_1 = \mathsf{RS}$:
3c $\mathbf{T}' \leftarrow \mathsf{DeleteRow}(\mathbf{T}, k)$;
4c $\mathbf{T}'' \leftarrow \mathsf{DeleteColumn}(\mathbf{T}', j)$;
5c $\mathbf{T}^{0\prime} \leftarrow \mathsf{DeleteRow}(\mathbf{T}^0, j)$;
6c $\mathbf{T}^{0\prime\prime} \leftarrow \mathsf{DeleteColumn}(\mathbf{T}^{0\prime}, n + j)$;

3 **return** $(\mathbf{T}'', \mathbf{s}', \mathbf{T}^{0\prime\prime})$

Lemma 3.45. *Consider the artificial LP given a feasible LP in standard form having coefficients* $\mathbf{A}, \mathbf{b}$ *and* $\mathbf{c}$*. Let* $\mathbf{T}$ *be the tableau with respect to basis* $\mathbf{s}$ *corresponding to an optimal solution* $\mathbf{x}$ *where no artificial variable is basic. Let* $\mathbf{B} = \mathbf{A}_\mathbf{s}$ *be the basis matrix. Then* $\mathbf{T}'$ *is a tableau corresponding to* $\mathbf{s}$ *for the original LP if* $\mathbf{T}'$ *is equal to* $\mathbf{T}$ *where*

(i) the columns $n + 1, \ldots, n + m$ *are removed and*

(ii) the last row is replaced with $\mathbf{t}$ *where* $t_i = \overline{c}_i$ *and* $t_{m+1} = -\mathbf{c}_\mathbf{s}\mathbf{B}^{-1}\mathbf{b}$.

Proof. Observe that

$$\mathbf{T} = \begin{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}_\mathbf{s}^{-1} & \mathbf{0} \\ -\mathbf{c}'_\mathbf{s} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}_\mathbf{s}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{0} & \mathbf{1} & 0 \end{pmatrix}$$

is the tableau corresponding to basis $\mathbf{s}$ of the artificial LP with cost vector $\mathbf{c}'$. If no artificial variable is basic, then Theorem 3.42 implies that $\mathbf{s}$ is a basis for the original LP as well. The corresponding basis matrices are the same since

$$\begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}_\mathbf{s} = \mathbf{A}_\mathbf{s} = \mathbf{B}.$$

Hence,

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}'_\mathbf{s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{0} & \mathbf{1} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{B}^{-1}\mathbf{A} & \mathbf{B}^{-1} & \mathbf{B}^{-1}\mathbf{b} \\ -\mathbf{c}'_\mathbf{s}\mathbf{B}^{-1}\mathbf{A} & \mathbf{1} - \mathbf{c}'_\mathbf{s}\mathbf{B}^{-1} & 0 \end{pmatrix}.$$

Furthermore,

$$\mathbf{T}' = \begin{pmatrix} \mathbf{A}_\mathbf{s}^{-1} & \mathbf{0} \\ -\mathbf{c}_\mathbf{s}\mathbf{A}_\mathbf{s}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{B}^{-1}\mathbf{A} & \mathbf{B}^{-1}\mathbf{b} \\ \mathbf{c} - \mathbf{c}_\mathbf{s}\mathbf{B}^{-1}\mathbf{A} & -\mathbf{c}'_\mathbf{s}\mathbf{B}^{-1}\mathbf{b} \end{pmatrix}$$

is a tableau corresponding to basis $\mathbf{s}$ for the original LP. Hence $\mathbf{T}'$ is equal to $\mathbf{T}$ where the columns $n+1, \ldots, n+m$ are removed and where the last row is replaced according to (ii). □

Algorithm 3.15 shows how to remove a non-basic column corresponding to an artificial variable based on Lemma 3.45.

Finally, Algorithm 3.16 shows how to compute the last row of the tableau corresponding to the costs of the original LP.

Algorithm 3.15: $(\mathbf{T}, \mathbf{T}^0, \mathbf{u}) \leftarrow \mathsf{DeleteColumn}_{\mathsf{Var}_1}(\mathbf{T}, k, \mathbf{T}^0, \mathbf{u})$

Input: $\mathbf{T}$, k, $\mathbf{T}^0$, $\mathbf{u}$
Output: $\mathbf{T}$, $\mathbf{T}^0$, $\mathbf{u}$

$\mathsf{VAR}_1 = \mathsf{LT}$:
1a $\quad \mathbf{T}' \leftarrow \mathsf{DeleteColumn}(\mathbf{T}, k)$;

$\mathsf{VAR}_1 = \mathsf{RS}$:
1b $\quad \mathbf{T}^{0\prime} \leftarrow \mathsf{DeleteColumn}(\mathbf{T}^0, k)$;

$\mathsf{VAR}_1 = \mathsf{ST}$:
1c $\quad \mathsf{findj} : u_j = k$;
2c $\quad \mathbf{u} = (u_1, \ldots, u_{j-1}, u_{j+1}, u_{n'-m'})$;
3c $\quad \mathbf{T}' \leftarrow \mathsf{DeleteColumn}(\mathbf{T}, j)$;

4 **return** $(\mathbf{T}', \mathbf{T}^{0\prime}, \mathbf{u}')$

Algorithm 3.16: $(\mathbf{T}, \mathbf{T}^0) \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{T}, \mathbf{c}, \mathbf{T}^0, \mathbf{s}, \mathbf{u}, q)$

Input: $\mathbf{T}$, $\mathbf{c}$, $\mathbf{T}^0$, $\mathbf{s}$, $\mathbf{u}$, q
Output: $\mathbf{T}$, $\mathbf{T}^0$

1 $\mathbf{v} \leftarrow \mathbf{c_s}$;
$\mathsf{VAR}_2 = \mathsf{IP}$:
2 $\quad \mathbf{c} \leftarrow q\mathbf{c}$;
3 $\mathbf{T} \leftarrow \mathsf{DeleteRow}(\mathbf{T}, m+1)$;

$\mathsf{VAR}_1 = \mathsf{LT}$:
4a $\quad$ **foreach** $i \in \{1, \ldots, n+1\}$ **do**
5a $\quad\quad t_{(m+1)i} \leftarrow c_i - \mathbf{v}\mathbf{T}_i$;

$\mathsf{VAR}_1 = \mathsf{ST}$:
4b $\quad$ **foreach** $i \in \{1, \ldots, n-m+1\}$ **do**
5b $\quad\quad t_{(m+1)i} \leftarrow c_{u_i} - \mathbf{v}\mathbf{T}_i$;

$\mathsf{VAR}_1 = \mathsf{RS}$:
4c $\quad \mathbf{t}^0_{m+1} \leftarrow (\mathbf{c}, 0)$;
5c $\quad$ **foreach** $i \in \{1, \ldots, m\}$ **do**
6c $\quad\quad t_{(m+1)i} \leftarrow -\mathbf{v}\mathbf{T}_i$;
7c $\quad t_{(m+1)(m+1)} \leftarrow 1$;

8 **return** $(\mathbf{T}, \mathbf{T}^0)$;

3.3.2 Two-Phase Simplex with One Artificial Variable

In order to improve the performance of the two-phase simplex algorithm we consider a different well known artificial LP using only one artificial variable. This way the corresponding tableau has only one extra column compared to the tableau corresponding to

the original LP and initializing phase II has to take care of just one artificial variable.

Given an LP in standard form, this section considers the following artificial LP.

$$\begin{array}{rlcl} \min & \mathbf{c}'\mathbf{x} = x_{n+1}, & & \\ \text{subject to} & \begin{pmatrix} & -1 \\ \mathbf{A} & \vdots \\ & -1 \end{pmatrix} \mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}. \end{array} \tag{3.55}$$

Observe that in contrast with the standard two-phase simplex algorithm, a basic feasible solution to this artificial LP is not obvious. We show that given a basic solution to the original LP, one can derive a basic feasible solution for this artificial LP easily. However, a basic solution to this artificial LP exists only if we require that $\mathbf{A}$ of the original LP has full row rank.

With respect to phase II, we note that, except with respect to redundancy, the theorems of previous section also apply to this artificial LP, and thus that initializing phase II proceeds as discussed in the previous section. Since $\mathbf{A}$ has full row rank, redundancy is not an issue.

In the next lemma we show that since $\mathbf{A}$ has full row rank we may assume without loss af generality that the original LP is in *canonical* form, i.e.,

$$\begin{array}{rlcl} \min & \mathbf{c}\mathbf{x}, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A}' & \mathbf{I}_m \end{pmatrix} \mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{3.56}$$

where $\mathbf{A}'$ is an $m \times (n-m)$ matrix.

Lemma 3.46. *Consider an LP in standard form. Suppose that* $\mathbf{A}$ *is of rank* m *and that* $\mathbf{s} \in \{1, \ldots, n\}^m$ *is a basis and* $\mathbf{u}$ *a co-basis. Then, the given LP is equivalent to LP* (3.56)*, where* $\mathbf{A}' = \mathbf{A}_\mathbf{s}^{-1}\mathbf{A}_\mathbf{u}$ *and* $\mathbf{b}' = \mathbf{A}_\mathbf{s}^{-1}\mathbf{b}$*.*

Proof. Without loss of generality $\mathbf{s} = (n-m+1, \ldots, n)$ and $\mathbf{u} = (1, \ldots, n-m)$. From

$$\mathbf{A}_\mathbf{s}^{-1}\mathbf{A}\mathbf{x} = \mathbf{A}_\mathbf{s}^{-1}\mathbf{b}$$

we have

$$\mathbf{A}_\mathbf{s}^{-1}\mathbf{A} = \mathbf{A}_\mathbf{s}^{-1} \begin{pmatrix} \mathbf{A}_\mathbf{u} & \mathbf{A}_\mathbf{s} \end{pmatrix} = \begin{pmatrix} \mathbf{A}' & \mathbf{I}_m \end{pmatrix},$$

and thus we conclude that the constraints are equivalent. Hence, the given LP is equivalent to LP (3.56). □

The following theorem shows how to initialize phase I when the original LP is in canonical form.

Theorem 3.47. *Consider an LP in canonical form and the corresponding artificial LP. Let*

$$\beta := \operatorname{argmin}(\mathbf{b}').$$

If $b_\beta \geq 0$*, then the basic solution* $\mathbf{x}'$ *corresponding to basis* $\mathbf{s} = (n-m+1, \ldots, n)$ *is a basic feasible solution to the original LP. Otherwise, if* $b_\beta < 0$ *then* $\mathbf{x}$ *satisfying*

$$x_i = \begin{cases} b_j - b_\beta, & \text{if } i = n-m+j, \\ -b_\beta & \text{if } i = n+1, \\ 0 & \text{otherwise,} \end{cases} \tag{3.57}$$

for $i = 1, \ldots, n+1$, is a basic feasible solution to the artificial LP with respect to basis $\mathbf{s}' = (s_1, \ldots, s_{\beta-1}, n+1, s_{\beta+1}, \ldots, s_m)$.

Proof. Note that by construction $\mathbf{s} = (n-m+1, \ldots, n)$ is a basis for the original LP, since it is in canonical form. Hence $\mathbf{A}_\mathbf{s} = \mathbf{I}_m$.

Let $\mathbf{x}'$ be the basic solution corresponding to basis $\mathbf{s}$.

If $b_\beta \geq 0$ it follows that $\mathbf{b}' = \mathbf{x}'_\mathbf{s} \geq \mathbf{0}$ and thus that $\mathbf{x}'$ is basic feasible to the original LP.

Suppose $b_\beta < 0$. Then $\mathbf{s}' = (s_1, \ldots, s_{\beta-1}, n+1, s_{\beta+1}, \ldots, s_m)$ is a basis, since

$$\mathbf{B} = \begin{pmatrix} & & -1 \\ \mathbf{A}' & \mathbf{I}_m & \vdots \\ & & -1 \end{pmatrix}_{\mathbf{s}'} = (\mathbf{e}_1, \ldots, \mathbf{e}_{\beta-1}, -\mathbf{1}, \mathbf{e}_{\beta+1}, \ldots, \mathbf{e}_m)$$

has an inverse

$$\mathbf{B}^{-1} = \begin{pmatrix} \mathbf{e}_1 - \mathbf{e}_\beta \\ \vdots \\ \mathbf{e}_{\beta-1} - \mathbf{e}_\beta \\ -\mathbf{e}_\beta \\ \mathbf{e}_{\beta+1} - \mathbf{e}_\beta \\ \vdots \\ \mathbf{e}_m - \mathbf{e}_\beta \end{pmatrix}.$$

The corresponding basic solution satisfies

$$\mathbf{x}_{\mathbf{s}'} = \mathbf{B}^{-1}\mathbf{b}' = \begin{pmatrix} b'_1 - b'_\beta \\ \vdots \\ b'_{\beta-1} - b'_\beta \\ -b'_\beta \\ b'_{\beta+1} - b'_\beta \\ \vdots \\ b'_m - b'_\beta \end{pmatrix}.$$

Observe that $b_\beta = \min(\mathbf{b})$ and, therefore, $\mathbf{x} \geq \mathbf{0}$. Hence it is a basic feasible solution to the artificial LP. □

Algorithm 3.17 shows ho to initialize phase I with respect to the alternative artificial LP (3.55).

3.3.3 Big-M Method

Instead of solving an LP in standard form in two-phases, the *Big-M method* solves both LPs simultaneously by solving similar to the standard two-phase simplex algorithm either

$$\begin{array}{lrcl} \min & \mathbf{cx} + M\sum_{i=1}^{m} x_{n+i}, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}\mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{3.58}$$

Algorithm 3.17: $(\mathbf{T}, \mathbf{s}, \text{pred}, \mathbf{T}^0, \mathbf{u}, q) \leftarrow \mathsf{InitializePhaseI1Art}_{\mathsf{VAR}_1,\mathsf{VAR}_2}(\mathbf{A}, \mathbf{b})$

Input: $\mathbf{A}$, $\mathbf{b}$
Output: $\mathbf{T}$, $\mathbf{s}$, pred, $\mathbf{T}^0$, $\mathbf{u}$, q

1 $\mathbf{s} \leftarrow (n-m+1, \ldots, n)$;
2 $\mathbf{T}^0 \leftarrow \begin{pmatrix} \mathbf{A} & -\mathbf{1} & \mathbf{b} \\ \mathbf{0} & 1 & 0 \end{pmatrix}$;
3 $\mathbf{D} \leftarrow \begin{pmatrix} \mathbf{I}_m & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}$;
4 $\alpha \leftarrow 1$;

$\mathsf{VAR}_2 = \mathsf{IP}$:
5a $q \leftarrow 1$;
6a $\alpha \leftarrow -1$;

7 $\beta := \text{argmin}(\mathbf{b})$;

$\mathsf{VAR}_1 = \mathsf{LT}$:
8a $\mathbf{T} \leftarrow \mathbf{D}\mathbf{T}^0$;
9a **if** $b_\beta \geq 0$ **then return** $(\mathbf{T}, \mathbf{s}, \mathsf{FeasibleO}, q)$;
10a $\mathbf{t}_\beta \leftarrow \alpha \mathbf{t}_\beta$;
11a $(\mathbf{T}, q) \leftarrow \mathsf{Pivot}_{\mathsf{LT},\mathsf{VAR}_2}(\mathbf{T}, \beta, n+1, q)$;
12a $s_\beta \leftarrow n+1$;
13a **return** $(\mathbf{T}, \mathbf{s}, \mathsf{FeasibleA}, q)$;

$\mathsf{VAR}_1 = \mathsf{ST}$:
8b $\mathbf{u} \leftarrow (1, \ldots, n-m, n+1)$;
9b $\mathbf{T} \leftarrow \mathbf{D}\mathbf{T}^0_{(\mathbf{u}, n+2)}$;
10b **if** $b_\beta \geq 0$ **then return** $(\mathbf{T}, \mathbf{s}, \mathsf{FeasibleO}, \mathbf{u}, q)$;
11b $\mathbf{t}_\beta \leftarrow \alpha \mathbf{t}_\beta$;
12b $(\mathbf{T}, q) \leftarrow \mathsf{Pivot}_{\mathsf{ST},\mathsf{VAR}_2}(\mathbf{T}, \beta, n-m+1, q)$;
13b $u_{n-m+1} \leftrightarrow s_\beta$;
14b **return** $(\mathbf{T}, \mathbf{s}, \mathsf{FeasibleA}, \mathbf{u}, q)$;

$\mathsf{VAR}_1 = \mathsf{RS}$:
8c $\mathbf{T} \leftarrow \mathbf{D}$;
9c **if** $b_\beta \geq 0$ **then return** $(\mathbf{T}, \mathbf{s}, \mathsf{FeasibleO}, \mathbf{T}^0, q)$;
10c $\mathbf{t}_\beta \leftarrow \alpha \mathbf{t}_\beta$;
11c $(\mathbf{T}, q) \leftarrow \mathsf{Pivot}_{\mathsf{RS},\mathsf{VAR}_2}(\mathbf{T}, \beta, n+1, \mathbf{T}^0, q)$;
12c $s_\beta \leftarrow n+1$;
13c **return** $(\mathbf{T}, \mathbf{s}, \mathsf{FeasibleA}, \mathbf{T}^0, q)$;

or, similar to the two-phase simplex algorithm with one artificial variable,

$$\begin{array}{lrcl} \min & \mathbf{c}\mathbf{x} + Mx_{n+1}, & & \\ \text{subject to} & \begin{pmatrix} & -1 \\ \mathbf{A} & \vdots \\ & -1 \end{pmatrix} \mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{3.59}$$

where M is a "very big number".

Suppose that $M = \infty$, where

$$\infty x = \begin{cases} \infty, & \text{if } x > 0, \\ 0, & \text{if } x = 0, \\ -\infty, & \text{if } x < 0. \end{cases}$$

Initializing any simplex variant to solve these linear programs can be done using the algorithms initializing phase I, where the cost vectors of the corresponding artificial LPs are replaced by respectively $(\mathbf{c}, M\mathbf{1})$ with respect to the standard two-phase simplex algorithm or $(\mathbf{c}, M)$ with respect to the two-phase simplex algorithm with one artificial variable.

We will show that the resulting optimum is either the optimum of the original LP or a proof that the original LP is infeasible. While Lemma 3.48 proves this fact for LP (3.58), we note that following the same lines of the proof the lemma also holds for LP (3.59). It follows that running phase I suffices to solve the original LP.

Lemma 3.48. *Let* $\mathbf{x}$ *be an optimal basic feasible solution to the artificial LP* (3.58)*. If the corresponding costs are equal to* ∞*, then the original LP is infeasible. But if the corresponding costs are less than* ∞*, then* $x_1, \cdots, x_n$ *is optimal to the original LP.*

Proof. Firstly, suppose that $\mathbf{x}$ has cost ∞. Suppose furthermore that $\mathbf{x}'$ is a feasible solution to the original LP. It follows that $(\mathbf{x}', \mathbf{0})$ is feasible to LP (3.58) having cost $\mathbf{c}\mathbf{x}' < \infty$, contradicting the optimality of $\mathbf{x}$.

Next, suppose that $\mathbf{x}$ has cost less than ∞. It follows that all artificial values are set to zero. Hence $\hat{\mathbf{x}} = (x_1, \ldots, x_n)$ is (basic) feasible to the original LP. Let $\mathbf{x}'$ be a feasible solution to the original LP such that $\mathbf{c}\mathbf{x}' < \mathbf{c}\hat{\mathbf{x}}$. It follows that $(\mathbf{x}', \mathbf{0})$ is feasible to LP (3.58) having cost $\mathbf{c}\mathbf{x}' < \mathbf{c}\hat{\mathbf{x}}$, contradicting the optimality of $\mathbf{x}$. Hence $\hat{\mathbf{x}}$ is optimal to the original LP. □

When implementing the big-M method one has to choose a finite number M so that any feasible solution to the original LP has lower cost than any feasible solution to the corresponding artificial problem where one artificial variable is nonzero. Lemma 3.49 shows how to choose M given that the coefficients of the original LP are bounded.

Lemma 3.49. *Consider an LP in standard form, where the absolute value of all coefficients is bounded. If* $M > 2^{2L+1}$*, where* L *satisfies Theorem 3.34, then the optimal costs of both LP* (3.58) *and* (3.59) *are larger than* 2^L *if and only if the original LP is infeasible.*

Proof. By Theorem 3.34 it follows that the size of the numbers in the tableau $\mathbf{T}$ are represented by integers bounded by 2^L. Suppose that $\mathbf{T}$ is a tableau corresponding to an optimal solution $\mathbf{x}$ to LP (3.58) or to LP (3.59). Let the basis be $\mathbf{s}$, and let $q = |\det(\mathbf{A}_\mathbf{s})|$.

Then by Theorem 3.34

$$|\mathbf{cx}| = |t_{(n+1)(n+m+1)}| < 2^L. \tag{3.60}$$

Theorem 3.34 shows that for any nonzero x_i one has $1 \leq |\widetilde{x}_i| = q|x_i| < 2^L$, where $1 \leq q < 2^L$. It follows that $\frac{1}{2^L} < |x_i| < 2^L$.

If at least one artificial variable is nonzero, then

$$|M \sum_{i=1}^{m} x_{n+i}| > 2^{2L+1} \frac{1}{2^L} = 2^{L+1}$$

and this it follows from Eq. (3.60) that

$$\mathbf{cx} + M \sum_{i=1}^{m} x_{n+i} > -2^L + 2^{L+1} = 2^L. \tag{3.61}$$

On the other hand if all artificial variables are equal to zero, then

$$\mathbf{cx} + M \sum_{i=1}^{m} x_{n+i} = \mathbf{cx} < 2^L.$$

□

Remark 3.50. Instead of making the numbers larger to accommodate M, we could also represent the numbers in tuples. For example, let $a = a_1 + a_2 M$, where $2^{-L} < a_i < 2^L$. Note that M only appears in the costs, and therefore, in the last row of any tableau $\mathbf{T}$ only. Indeed a pivot operation is never done on the last row, so all updates are independent to M, except for the last row.

Thus, only the last row of the tableau would contain numbers as tuples. This means that the existence of M only affects the column selection and updating the last row, since those are the only parts where the last row of $\mathbf{T}$ is touched. Let $\mathbf{t}$ denote the last row of $\mathbf{T}$. Suppose that the i-th entry of the last row is given by $(t_i, t_i^{(M)})$ and represents the value $t_i + Mt_i^{(M)}$.

With respect to the column selection note that by the definition of M

$$t_i + Mt_i^{(M)} < 0 \Leftrightarrow t_i^{(M)} < 0 \vee \left(t_i^{(M)} = 0 \wedge t_i < 0\right).$$

And with respect to pivoting on $t_{k\ell}$ note that

$$t'_i + Mt_i^{\prime(M)} = t_i + Mt_i^{(M)} - \frac{(t_\ell + Mt_\ell^{(M)})t_{ki}}{t_{k\ell}}$$

and thus

$$t'_i = t_i - \frac{t_\ell t_{ki}}{t_{k\ell}}$$

and

$$t_i^{\prime(M)} = t_i^{(M)} - \frac{t_\ell^{(M)} t_{ki}}{t_{k\ell}}.$$

◇

Remark 3.51. One would expect that the big-M method requires less iterations in total, since it runs a simplex variant only once. Nabli provides in [Nab09] experimental results in terms of the number of iterations of the two-phase simplex and the big-M method. It seems that there is no big difference between the total number of iterations.

Therefore, the Big-M method seems to be less efficient than the two-phase simplex method since every iteration of the big-M method is more expensive due to either Lemma 3.49 or Remark 3.50. Moreover, while initializing phase II, the two-phase simplex switches to a smaller tableau without the artificial variables, while the Big-M has the larger tableau including columns with respect to the artificial variables in all its iterations. ◇

Chapter 4

Building Blocks for Secure Linear Programming

In this chapter we review and build protocols for the basic operations used by the linear programming protocols in Chapter 5.

In the first section we review the security properties of the protocols presented in this chapter. The protocols are given as arithmetic circuits. Therefore, they will be statistically secure by Theorem 2.13. The statistical security is due to the fact that random numbers dependent on some secret values are opened during protocol execution. Basic results on statistical distance are discussed which are used to prove statistical security.

The second section exploits features of Shamir's secret sharing scheme to be able to non-interactively generate random numbers, following [CDI05, DT08]. In addition, we provide a protocol, following [BGW88, CDI05], that securely computes an inner product that has the same complexity with respect to communication as the multiplication protocol.

The third section shows how to build efficient arithmetic circuits for evaluating k-ary and prefix operations for any binary associative operator $\odot$, which are defined on any $\mathbf{x}$ by

- *k-ary operation:* $y = x_1 \odot \cdots \odot x_k = \odot_{i=1}^{k} x_i$, and
- *prefix operation:* $y_j = \odot_{i=1}^{j} x_i$ for each $j = 1, \ldots, k$.

For important tools with respect to linear programming, we compare the circuits of logarithmic depth with circuits of constant depth from the literature. In some cases we improve the performance of the protocols in the literature by slight modifications.

The fourth section shows how to apply those building blocks to build efficient circuits for integer comparison following the approach of [ST06, GSV07]. With respect to comparisons of the form $x \leq y$, we compare the logarithmic depth circuit of [GSV07] with the constant depth circuit of [Rei09]. We present a new circuit for comparisons of the form $x = y$ that has $\log^*$ depth, where $\log^*(k) = \min\{i | \log^i(k) \leq 1\}$.

To accommodate computation in $\mathbb{Q}$, we present in the fifth section protocols for fixed point arithmetic following [CS10, CH10b]. Special attention will be paid to an improved protocol for division [CH10b].

The last section shows techniques to hide entries in a matrix by means of secret indexing following [Tof09]. We present protocols that securely modifies matrices based on secret indices that will be used in the linear programming protocols.

Efficiency Measures

We focus on *communication* complexity, i.e., the total number of communicated bits by each party, and *round* complexity. The latter counts the total number of interactive rounds, where in each successive round parties are sending messages that are dependent on received messages in earlier rounds.

We call the amount of data send by each party in a multiplication protocol (see Protocol 2.9) an *invocation*, which is abbreviated as inv. We will also count the number of interactive rounds, which is abbreviated as rnd.

4.1 Statistical Security

The protocols in this chapter will satisfy the requirements of the composition theorem of [Can00] and, therefore, any modular composition of those protocol will be secure (see Theorem 2.13).

However, during some of the protocols the parties need to reveal some non-uniform random number $x + U$ that depends on some secret x, where U is a random variable. By the following lemmas it follows that statistically, the opened values, in the protocols in this chapter, reveal nothing about the secret. Precisely, it follows that for any secret x and random variable U, the distribution of U will be such that the statistical difference between $x + U$ and U will be negligible.

We first show that if U is uniform on some finite set then the statistical distance between $X + U$ and U, where X is some random variable of unknown distribution, can be bounded by the size of the domain of U.

Lemma 4.1. *Let M and K be positive integers with $M \leq K$. Let $X \in \{0, \ldots, M-1\}$ and $U \in \{0, \ldots, K-1\}$ be random variables, where U is uniformly distributed. Then $\Delta(U; X + U) \leq (M-1)/K$ and this bound is tight.*

Proof. This is Lemma 1 in [ST06, Appendix A]. □

In Theorem 4.4 we show that this holds even if U is not uniform, but a sum of uniform distributions. For this we will use the following lemmas. See for example [Sho05] for their proofs.

Lemma 4.2. *Let X and Y be random variable taking values in some finite set V and let $f : V \to V'$ be some function mapping to some finite set V'. Then*

$$\Delta(f(X); f(Y)) \leq \Delta(X; Y). \tag{4.1}$$

Lemma 4.3. *Let X,Y and Z be random values, where X and Z are independent and Y and Z are independent. Then*

$$\Delta((X, Z); (Y, Z)) = \Delta(X; Y). \tag{4.2}$$

Theorem 4.4. *Let $X \in \{0, \ldots, 2^k - 1\}$ and U be random variables, where $U = \sum_{i=1}^{n} U_i$ for some finite n, where each U_i is independent and uniform in $\{0, \ldots, 2^{k+\kappa} - 1\}$. Then:*

$$\Delta(X + U; U) < 2^{-\kappa}. \tag{4.3}$$

Proof. Let f be defined by $f(x, y) := x + y$. It follows that

$$\begin{aligned}
\Delta(X+U;U) &= \Delta(X+\sum_{i=1}^{n} U_i; \sum_{i=1}^{n} U_i) \\
&= \Delta(X+\sum_{i=1}^{n-1} U_i + U_n; \sum_{i=1}^{n-1} U_i + U_n) \\
&= \Delta(f(\sum_{i=1}^{n-1} U_i, X+U_n); f(\sum_{i=1}^{n-1} U_i, U_n)) \\
&\overset{\text{Lemma 4.2}}{\leq} \Delta((\sum_{i=1}^{n-1} U_i, X+U_n); (\sum_{i=1}^{n-1} U_i, U_n)) \\
&\overset{\text{Lemma 4.3}}{=} \Delta(X+U_n; U_n) \\
&\overset{\text{Lemma 4.1}}{\leq} \frac{2^k-1}{2^{k+\kappa}} < 2^{-\kappa}.
\end{aligned}$$

□

Theorem 4.5. *Let $X \in \{0, \ldots, 2^k - 1\}$ and U be random variables and let $U = U' + 2^k \sum_{i=1}^{n} U'_i$, where U' is a uniform random variable in $\{0, \ldots, 2^k - 1\}$ and each U'_i is uniform and independent in $\{0, \ldots, 2^\kappa - 1\}$. Then*

$$\Delta(X+U;U) < 2^{-\kappa}. \tag{4.4}$$

Proof. Let f be defined by $f(x, y) := x + y$. Using the same method as in the proof of Theorem 4.4 we obtain:

$$\begin{aligned}
\Delta(X+U;U) &= \Delta(X+\sum_{i=1}^{n} U_i; \sum_{i=1}^{n} U_i) \\
&= \Delta(X+\sum_{i=1}^{n-1} U_i + U_n; \sum_{i=1}^{n-1} U_i + U_n) \\
&= \Delta(f(\sum_{i=1}^{n-1} U_i, X+U_n); f(\sum_{i=1}^{n-1} U_i, U_n)) \\
&\overset{\text{Lemma 4.2}}{\leq} \Delta((\sum_{i=1}^{n-1} U_i, X+U_n); (\sum_{i=1}^{n-1} U_i, U_n))
\end{aligned}$$

$$\begin{aligned}
&\overset{\text{Lemma 4.3}}{=} \Delta(X+U_n; U_n) \\
&= \Delta(X+U'+2^k U'_n; U'+2^k U'_n) \\
&\overset{\text{Lemma 4.1}}{\leq} \frac{2^k-1}{2^{k+\kappa}} < 2^{-\kappa}.
\end{aligned}$$

□

4.2 Efficient Primitives for Shamir Secret Sharing

We consider (t, n)-Shamir secret sharing, where we require $t < n/2$. Recall that this scheme is defined over some finite field $\mathbb{F}_q$. In our protocols we will take $\mathbb{F}_q = \mathbb{Z}_q$, where q is some prime number. We let κ denote the security parameter with respect to the statistical distances.

The simplex algorithm that performs integer pivoting is defined over $\mathbb{Z}$ if the inputs are from $\mathbb{Z}$. Hence, we need some transformations to be able to use Shamir secret sharing to perform the evaluations.

4.2.1 Encoding Signed Integers as Prime Field Elements

Consider the set of k-bit signed integers

$$\mathbb{Z}_{\langle k \rangle} = \left\{ x \in \mathbb{Z} \,\middle|\, -2^{k-1} < x \leq 2^{k-1} \right\}.$$

Let $\odot \in \{+, -, \cdot\}$. We wish to find q and an injective map $\phi : \mathbb{Z}_{\langle k \rangle} \to \mathbb{Z}_q$ such that $\phi(x) \odot \phi(y) = \phi(z)$ for all $x, y \in \mathbb{Z}_{\langle k \rangle}$, where $x \odot y = z \in \mathbb{Z}_{\langle k \rangle}$.

The classical solution is to take $q > 2^k$ and $\phi : \mathbb{Z}_{\langle k \rangle} \to \mathbb{Z}_q$, where

$$\phi(x) := x \mod q.$$

Its inverse map $\phi^{-1} : \mathbb{Z}_q \to \mathbb{Z}_{\langle k \rangle}$ is given by

$$\phi^{-1}(x) := \begin{cases} x, & \text{if } x < q/2, \\ x - q, & \text{otherwise,} \end{cases}$$

where we view $x \in \mathbb{Z}_q$ as an integer being the least nonnegative representative of $x \mod q$.

Lemma 4.6. *Let $\odot \in \{+, -, \cdot\}$. If $x \in \mathbb{Z}_{\langle k \rangle}$, $y \in \mathbb{Z}_{\langle k \rangle}$ and $x \odot y \in \mathbb{Z}_{\langle k \rangle}$. Then*

$$x \odot y = \phi^{-1}(\phi(x) \odot \phi(y)).$$

Moreover, if y divides x then

$$\frac{x}{y} = \phi^{-1}\left(\phi(x) \odot \phi(y)^{-1}\right).$$

Proof. First, note that for all $x \in \mathbb{Z}_{\langle k \rangle}$ it follows that $\phi^{-1}(\phi(x)) = x$. Suppose that $x \in \mathbb{Z}_{\langle k \rangle}$, $y \in \mathbb{Z}_{\langle k \rangle}$ and $x \odot y \in \mathbb{Z}_{\langle k \rangle}$. Then,

$$\begin{aligned} \phi^{-1}\left(\phi(x) \odot \phi(y)\right) &= \phi^{-1}\left((x + \alpha_1 q) \odot (y + \alpha_2 q)\right) \\ &= \phi^{-1}(x \odot y + \beta q) \equiv \phi^{-1}(\phi(x \odot y)) \\ &= x \odot y, \end{aligned}$$

where

$$\beta = \begin{cases} \alpha_1 + \alpha_2, & \text{if } \odot = +, \\ \alpha_1 - \alpha_2, & \text{if } \odot = -, \\ x\alpha_1 + y\alpha_2 + \alpha_1\alpha_2 q, & \text{if } \odot = \cdot. \end{cases}$$

If y divides x, then $\frac{x}{y} \mod q = xy^{-1} \mod q$, where $y^{-1} \in \mathbb{Z}_q$ denotes the multiplicative inverse of $y \mod q$. Hence for all $x \in \mathbb{Z}_{\langle k \rangle}$ and $y \in \mathbb{Z}_{\langle k \rangle}$ where $y|x$, we have $\frac{x}{y} \in \mathbb{Z}_{\langle k \rangle}$ and

$$\frac{x}{y} = \phi^{-1}\left(\phi(x) \odot \phi(y)^{-1}\right).$$

□

Unless stated otherwise, the numbers x in the remainder of this chapter are $k+1$ signed bit numbers, i.e., $x \in \mathbb{Z}_{\langle k+1 \rangle}$. We say that y is a k bit number if $0 \leq y < 2^k$.

4.2.2 Noninteractive Random Number Generation

In this section we review some basic techniques to efficiently generate Shamir shares of random field elements. The protocols that we will discuss are efficient since they require one interactive setup protocol. In the setup protocol each party secretly shares one random field element. Given those shares the parties can generate fresh random shares by local computation only.

We apply the result of [CDI05] to use replicated secret sharing to noninteractively generate Shamir shares of uniformly random field elements. We remark here that since replicated secret sharing is used with threshold $t \approx n/2$, in which $\binom{n}{t}$ shares are computed, this noninteractive generation is only efficient if n is small.

Given replicated shares $[r]^R$ of some $r \in \mathbb{F}$, we will show how the parties can locally evaluate some function on their shares to get a consistent replicated sharing of some new secret r'. This r' will be uniformly random. We show how to convert the replicated shares into consistent Shamir shares.

Let $\mathcal{T} = \{T_1, \ldots, T_w\}$ denote the set of all size t subsets of $\{1, \ldots, n\}$.

Setup:

The parties generate a replicated sharing of some uniformly random $k \in \mathbb{F}$ as follows. First every party P_i draws $k_i \in_R \mathbb{F}$ and shares it among all parties using replicated secret sharing with threshold t. Then, all parties locally compute a consistent replicated sharing of $k = \sum_{i=1}^{n} k_i$. Precisely, each party P_i locally computes $[k]_j^R = \sum_{i=1}^{n} [k_i]_j$, where j is such that $i \notin T_j$.

The local addition of replicated shares $[x]^R$ and $[y]^R$ resulting in replicated shares $[x+y]^R$ is denoted simply by $[x]^R + [y]^R$.

Protocol 4.1: $[k]^R \leftarrow \mathsf{SetupRandRSS}(\mathbb{F})$

```
1 foreach party i = 1, ..., n do
2     pick k_i ∈_R F;
3     [k_i]^R ← RShare(k_i, n, t);
4 [k]^R ← Σ_{i=1}^{n} [k_i]^n;
5 return [k]^R
```

Noninteractive Pseudo-Random Share Generation:

Suppose that the parties have replicated shares of k. Let $\mathcal{H} : \mathbb{F} \times \mathbb{N} \to \mathbb{F}$ be a pseudo random function. Suppose that the parties have agreed upon a $c \in \mathbb{N}$. Then each party P_i computes $[r']_j^R = \mathcal{H}([k]_j, c)$ for all j such that $i \notin T_j$. It follows that $[r']^R$ is a consistent replicated sharing and from

$$r' = \sum_{i=1}^{w} \mathcal{H}([k]_i, c)$$

it follows that indeed r' is uniformly random.

Protocol 4.2: $[r]^R \leftarrow \mathsf{PRandRSS}(\mathbb{F})$

```
static ctr ← 0;
static [k]^R ← SetupRandRSS(F);
foreach party i = 1, ..., n do
    foreach j s.t. i ∉ T_j do
        [r]_j^R ← H([k]_j, ctr);
ctr++;
return [r]^R
```

Conversion of Replicated Shares into Shamir Shares:

The conversion of replicated shares into Shamir shares is based on the following lemma:

Lemma 4.7. *Consider the (t, n)-replicated secret sharing among parties $P_1, \ldots, P_n$ and let $[r]^R$ be the (replicated) sharing of $r \in \mathbb{F}$. Let $\mathcal{T} = \{T_1, \ldots, T_w\}$ denote the collection of all size t subsets of $\{1, \ldots, n\}$. The function $p(x) = \sum_{i=1}^{w} [r]_i^R p_i(x)$, where*

$$p_i(x) = \prod_{j \in T_i} \frac{j - x}{j}$$

is a polynomial of degree t that satisfies $p(0) = r$. Moreover each party P_i can compute $p(i)$ locally.

Proof. Observe, firstly, that $p_\ell(0) = 1$ for all $\ell = 1, \ldots, w$ and thus that

$$p(0) = \sum_{\ell=1}^{w} [r]_\ell^R p_\ell(0) = \sum_{\ell=1}^{w} [r]_\ell^R = r.$$

Secondly, since $p_\ell(j) = 0$ for all $j \in T_\ell$, the polynomial p satisfies

$$p(i) = \sum_{\ell=1, i \notin T_\ell}^{w} [r]_\ell^R p_\ell(j).$$

Hence $p(i)$ can be computed from the shares $[r]_j^R$, where $i \notin T_j$, i.e., all replicated shares held by party P_i. □

Protocol 4.3: $[r] \leftarrow \mathsf{RSSToShamir}([r]^R)$

foreach party $i = 1, \ldots, n$ **do**

$$[r]_i \leftarrow \sum_{j=1, i \notin T_j}^{w} \left([r]_j^R \prod_{\ell \in T_j} \frac{\ell - i}{\ell} \right);$$

return $[r]$

In conclusion, Protocol 4.6 generates Shamir shares of uniformly random field elements.

Protocol 4.4: $[r] \leftarrow \mathsf{PRandFld}(\mathbb{F})$

$[r]^R \leftarrow \mathsf{PRandRSS}(\mathbb{F})$;
$[r] \leftarrow \mathsf{RSSToShamir}([r]^R)$;
return $[r]$

Noninteractive Pseudo-Random Integer Share Generation:

Similarly to generating replicated shares of an uniformly random field element we change the pseudo random function to output a uniformly randomly chosen integer of fixed bit size. It follows that, repeating the procedure of above, the resulting secret is not uniformly random distributed, but its distribution is equal to the distribution of the sum of w uniform random integers.

Here, we assume that $\mathbb{F} = \mathbb{Z}_p$ is the field to represent $\mathbb{Z}_{\langle k \rangle}$.

Suppose that parties have $[k]^R$. Let $0 < \alpha < k$ and $\mathcal{H}^\alpha : \mathbb{Z}_p \times \mathbb{N} \to \{0,1\}^\alpha$ be a pseudo random function. Suppose that the parties have agreed upon a $c \in \mathbb{N}$. Then each party P_i computes $[r']_j^R = \mathcal{H}^\alpha([k]_j, c)$ for all j such that $i \notin T_j$. It follows that $[r']^R$ is a consistent replicated sharing and from

$$r' = \sum_{i=1}^{w} \mathcal{H}^\alpha([k]_i, c)$$

it follows that indeed r' has bit size $\alpha + \log(w)$ and its distribution is equal to the sum of w uniformly random numbers.

Protocol 4.5: $[r]^R \leftarrow \mathsf{PRandRISS}(\mathbb{Z}_p, \alpha)$

static $ctr \leftarrow 0$;
static $[k]^R \leftarrow \mathsf{SetupRandRSS}(\mathbb{Z}_p)$;
foreach party $i = 1, \ldots, n$ **do**
    foreach j **s.t.** $i \notin T_j$ **do**
        $[r]_j^R \leftarrow \mathcal{H}^\alpha([k]_j, ctr)$;
$ctr++$;
return $[r]^R$

Hence, to noninteractively generate a Shamir sharing of some random r' with bounded bit size, we run the following protocol.

Protocol 4.6: $[r] \leftarrow \mathsf{PRandInt}(\mathbb{Z}_p, \alpha)$

1 $[r]^R \leftarrow \mathsf{PRandRISS}(\mathbb{Z}_p, \alpha)$;
2 $[r] \leftarrow \mathsf{RSSToShamir}([r]^R)$;
3 **return** $[r]$

Noninteractive Pseudo Random Zero Sharing:

To noninteractively generate shares of zero, we apply the protocol PRZS from [CDI05]. The goal is to generate consistent shares of a uniformly random $2t$-degree polynomial $z(x)$. While we show how to noninteractively generate Shamir shares on a degree $2t$ random polynomial, we remark that instead of $2t+1$ parties any $t+1$ parties will be able to reconstruct $z(x)$ due to the underlying replicated secret sharing scheme with threshold t.

The idea is to use the polynomials $p_i(x)$ from Lemma 4.7 as follows.

Theorem 4.8. *Consider the (t,n)-replicated secret sharing among parties $P_1,\ldots,P_n$ and let $[\mathbf{r}]^R$ be a random (replicated) sharing of $\mathbf{r} \in_R \mathbb{F}^t$. Let $\mathcal{T} = \{T_1,\ldots,T_w\}$ denote the collection of all size t subsets of $\{1,\ldots,n\}$. Then $z(x)$, which is defined by*

$$z(x) = \sum_{i=1}^{w} p_i(x) \left([r_1]_i^R x + \ldots, [r_t]_i^R x^t \right),$$

is a $2t$ degree uniformly random polynomial with the restriction $z(0) = 0$. In addition, any party P_i can locally compute $z(i)$.

Proof. Similarly to Lemma 4.7 observe that from $p_i(j) = 0$ if $j \in T_i$, then

$$z(j) = \sum_{i=1, j \notin T_i}^{w} p_i(j) \left([r_1]_i^R j + \cdots + [r_t]_i^R j^t \right),$$

which can be computed by P_j since it has all replicated shares $[\mathbf{r}]_i$ when $j \notin T_i$.

Observe that $z(x)$ can be written as $z(x) = \sum_{i=1}^{2t} \alpha_i x^i$. We need to show that the vector $\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_{2t})$ is uniformly random in $\mathbb{F}^{2t}$. Note that $\boldsymbol{\alpha}$ can be computed by solving the system

$$\begin{pmatrix} z(1) \\ z(2) \\ z(3) \\ \vdots \\ z(2t) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 2 & 2^2 & 2^3 & \cdots & 2^{2t} \\ 3 & 3^2 & 3^3 & \cdots & 3^{2t} \\ & & & \ddots & \\ 2t & (2t)^2 & (2t)^3 & \cdots & (2t)^{2t} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_{2t} \end{pmatrix}. \tag{4.5}$$

Let u and v be such that $T_u = \{1,\ldots,t\}$ and $T_v = \{t+1,\ldots,2t\}$. Then let

$$\beta_k = \sum_{i=1, i \neq v, k \notin T_i}^{w} p_i(k) \left([r_1]_i^R k + \ldots, [r_t]_i^R k^t \right),$$

for all $k = 1, \ldots, t$, and

$$\gamma_k = \sum_{i=1, i\neq u, k\notin T_i}^{w} p_i(k+t)\left([r_1]_i^R(k+t) + \ldots, [r_t]_i^R(k+t)^t\right).$$

Then

$$\begin{pmatrix} z(1) \\ \vdots \\ z(t) \\ z(t+1) \\ \vdots \\ z(2t) \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_t \\ \gamma_1 \\ \vdots \\ \gamma_{2t} \end{pmatrix} + \begin{pmatrix} 1 & \cdots & 1^t & & & \\ & \ddots & & & & \\ t & \cdots & t^t & & & \\ & & & t+1 & \cdots & (t+1)^t \\ & & & & \ddots & \\ & & & 2t & \cdots & (2t)^t \end{pmatrix} \begin{pmatrix} p_v(1)[r_1]_v^R \\ \vdots \\ p_v(t)[r_t]_v^R \\ p_u(t+1)[r_1]_u^R \\ \vdots \\ p_u(2t)[r_t]_u^R \end{pmatrix}. \tag{4.6}$$

From Eq. (4.5) it follows that $\boldsymbol{\alpha}$ is uniformly random if $z(1), \ldots, z(2t)$ is uniformly random. Since $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ is independent on $[\mathbf{r}]_v^R$ and $[\mathbf{r}]_u^R$, which are uniformly random and independent, we have by Eq. (4.6) that $(z(1), \ldots, z(2t))$ is uniformly random. □

Protocol 4.7: $(z_1, \ldots, z_n) \leftarrow \mathsf{PRandZero}(\mathbb{F})$

foreach $i = 1, \ldots, t$ **do**
  $[r_i]^R \leftarrow \mathsf{PRandRSS}(\mathbb{F})$;
foreach party $j = 1, \ldots, n$ **do**
  $z_j = \sum_{i=1, j\notin T_i}^{w} p_i(j)\left([r_1]_i^R j + \cdots + [r_t]_i^R j^t\right)$;
  return z_j;

4.2.3 Efficient Arithmetic for Shamir Secret Sharing

This section provides efficient protocols for securely computing inner products. In addition we will use the noninteractive random number generation to efficiently perform a multiplication or inner product, where the result is opened.

Inner Products

Let $\mathbf{x} = x_1, \ldots, x_m$ and $\mathbf{y} = y_1, \ldots, y_m$ be vectors in $\mathbb{F}_q^m$. Let $[\mathbf{x}]$ denote the vector where each entry is (Shamir) secret shared, i.e., $[\mathbf{x}] = [x_1], \ldots, [x_m]$ and let the vector $[\mathbf{x}]_k = [x_1]_k, \ldots, [x_m]_k$ denote party P_k's shares of $\mathbf{x}$.

Naively, to compute an inner product securely one could run the multiplication protocol to compute each product $[x_i y_i]$ and compute the result by $\sum_{i=1}^m [x_i y_i]$. The following lemma

shows how to extend the multiplication protocol of [BGW88] to be able to compute any inner product by adding local computations only. Actually, we will show how to extend the multiplication protocol to be able to compute any *generalized inner product* by adding local computations only.

Lemma 4.9. *Suppose that* $[\mathbf{x}]$ *and* $[\mathbf{y}]$ *are length* m *vectors that are* (t,n)*-Shamir shared among parties* $P_1,\ldots,P_n$*. Let* $\alpha_1,\ldots,\alpha_m \in \mathbb{F}^n$ *and* $m_k = \sum_{i=1}^m \alpha_i[x_i]_k[y_i]_k$ *for* $i = 1,\ldots,m$ *and* $k = 1,\ldots,n$*. Let* $\{r_1,\ldots,r_{2t+1}\} \in \{1,\ldots,n\}^{2t}$ *and* $\lambda_1,\ldots,\lambda_{2t+1}$ *denote the Lagrange coefficients for reconstruction of parties* $P_{r_1},\ldots,P_{r_{2t+1}}$*. Then*

$$\sum_{j=1}^{2t+1} \lambda_j m_{r_j} = \sum_{i=1}^{m} \alpha_i x_i y_i.$$

Proof. Let f_i and g_i be the polynomials over $\mathbb{F}$ of degree t such that $f_i(j) = [x_i]_j$ and $g_i(j) = [y_i]_j$ for $i = 1,\ldots,m$ and $j = 1,\ldots,n$. Then

$$m_k = \sum_{i=1}^{m} \alpha_i[x_i]_k[y_i]_k = \sum_{i=1}^{m} \alpha_i f_i(k) g_i(k).$$

It follows that each (k, m_k) is on the polynomial $h = \sum_{i=1}^m \alpha_i f_i g_i$, which is a $2t$ degree polynomial. Furthermore, we have by construction that

$$h(0) = (\sum_{i=1}^{m} \alpha_i f_i g_i)(0) = \sum_{i=1}^{m} \alpha_i f_i(0) g_i(0) = \sum_{i=1}^{m} \alpha_i x_i y_i$$

and by Lagrange interpolation

$$h(0) = \sum_{j=1}^{2t+1} \lambda_j h(r_j) = \sum_{j=1}^{2t+1} \lambda_j \left(\sum_{i=1}^{m} \alpha_i f_i(r_j) g_i(r_j) \right) = \sum_{j=1}^{2t+1} \lambda_j m_{r_j}.$$

□

In conclusion, given $[\mathbf{x}]$ and $[\mathbf{y}]$, the parties securely compute shares for the generalized inner product between $\mathbf{x}$ and $\mathbf{y}$ as follows:

Protocol 4.8: $[c] \leftarrow \mathsf{Inner}([\mathbf{x}],[\mathbf{y}])$

1 **foreach party** $i = 1,\ldots,2t+1$ **do**
2 $\quad m_i \leftarrow [\mathbf{x}]_i[\mathbf{y}]_i$;
3 $\quad [m_i] \leftarrow \mathsf{SShare}(m_i, t, n)$;
4 $[c] \leftarrow \sum_{i=1}^{2t+1} \left([m_i] \prod_{j=1, j\neq i}^{2t+1} \frac{-j}{i-j} \right)$;
5 **return** $[c]$

Security is immediate by similarity to the multiplication protocol (Protocol 2.9).

Multiplication with Public Result

If the parties wish to compute and reveal ab from $[a]$ and $[b]$, then they can run $[c] \leftarrow \mathsf{Mul}([a],[b])$ followed by $ab \leftarrow \mathsf{Open}([c])$. This takes 2 interactive rounds and 2 invocations.

To remove one round of interaction, observe that in the second phase of Mul the parties secretly share their shares $[a]_i[b]_i$ and then compute shares of ab locally using the reconstruction formula. Naively, if ab may be revealed then the reconstruction can be done from the shares $[a]_i[b]_i$. However, the $2t$ shares $[a]_i[b]_i$ are not uniformly random and depend on a and b. Hence a simulator not knowing a and b may not provide indistinguishable views.

To fix this problem one noninteractively generates shares on a random $2t$-degree polynomial $z(x)$, where $z(0) = 0$ and use $m_i = [a]_i[b]_i + z(i)$ as reconstruction shares. Hence it follows that any set of $2t$ values of m_i is uniformly random and does not depend on a and b anymore. This results in the following protocol, which is proven to be secure in [CDI05] using the framework of [Can00].

Protocol 4.9: $c \leftarrow \mathsf{MulPub}([a],[b])$

$(z_1,\ldots,z_n) \leftarrow \mathsf{PRandZero}(\mathbb{F})$;
foreach party $i = 1,\ldots,2t+1$ **do**
  $m_i \leftarrow [x]_i[y]_i + z_i$;
  send m_i **to all parties** ; `// 1 rnd, 1 inv.`
$c \leftarrow \sum_{i=1}^{2t+1} \left(m_i \prod_{j=1, j\neq i}^{2t+1} \frac{-j}{i-j} \right)$;
return c

And similarly with respect to inner products the following protocol is executed.

Protocol 4.10: $c \leftarrow \mathsf{InnerPub}([\mathbf{a}],[\mathbf{b}])$

$(z_1,\ldots,z_n) \leftarrow \mathsf{PRandZero}(\mathbb{F})$;
foreach party $i = 1,\ldots,2t+1$ **do**
  $m_i \leftarrow [\mathbf{x}]_i \cdot [\mathbf{y}]_i + z_i$;
  send m_i **to all parties** ; `// 1 rnd, 1 inv.`
$c \leftarrow \sum_{i=1}^{2t+1} \left(m_i \prod_{j=1, j\neq i}^{2t+1} \frac{-j}{i-j} \right)$;
return c

Computing the Field Inverse

Given $[x]$, where $x \in \mathbb{F}^*$, suppose that the parties wish to compute $[y]$, where $y = x^{-1}$. The parties first generate $[r]$, where r is uniformly random. Then, they compute and reveal $z = xr$, which is uniformly random. If $z \neq 0$, then they compute the result locally by $z^{-1}[r]$. Otherwise, they try again.

The probability that a uniformly random $z \in \mathbb{F}$ is equal to zero is $1/|\mathbb{F}|$, which is usually negligible.

Protocol 4.11: $[y] \leftarrow \mathsf{Inv}([x])$

do
  $[r] \leftarrow \mathsf{PRandFld}(\mathbb{F})$;
  $z \leftarrow \mathsf{MulPub}([x], [r])$; `// 1 rnd, 1 inv.`
while $z = 0$;
$[y] \leftarrow z^{-1}[r]$;
return $[y]$

Random Bit Generation

Protocol 4.12 generates random bits. Here we assume that $q = 3 \bmod 4$.

Protocol 4.12: $[b] \leftarrow \mathsf{PRandBit}(\mathbb{Z}_q)$

do
  $[r] \leftarrow \mathsf{PRandFld}(\mathbb{Z}_q)$;
  $u \leftarrow \mathsf{MulPub}([r], [r])$; `// 1 rnd, 1 inv.`
while $u = 0$;
$v \leftarrow u^{-(q+1)/4} \bmod q$;
$[b] \leftarrow (v[r] + 1)(2^{-1} \bmod q)$;
return $[b]$

For convenience we specify the protocol for generating multiple bits.

Protocol 4.13: $[\mathbf{b}] \leftarrow \mathsf{PRandBits}(\mathbb{Z}_q, m)$

foreach $i = 1, \ldots, m$ **do parallel**
  $[b_i] \leftarrow \mathsf{PRandBit}(\mathbb{Z}_q)$;
return $[b]$

We will use the notation given in Table 4.1(a) to denote invocation of the corresponding protocols. Table 4.1(b) presents the efficiency and security of the protocols that are discussed in this section.

4.3 Arithmetic Circuits for Prefix and k-ary Operations

Let $\mathcal{S}$ be a set and $\odot : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ be an associative binary operator. We denote $[x] \odot [y]$ as the secure evaluation of $x \odot y$ with secret inputs and output. This section considers the following operations (note that we start counting from 0 in this section)

- *k-ary operation:* A k-ary operation computes $y = x_0 \odot \cdots \odot x_{k-1} = \odot_{i=0}^{k-1} x_i$, and
- *prefix operation:* A prefix operation computes $y_j = \odot_{i=0}^{j} x_i$ for $j = 0, \ldots, k-1$.

This section shows how to construct arithmetic circuits of logarithmic depth to perform k-ary and prefix operations in general for any associative binary operator. In addition,

Notation	Protocol
$[c] \leftarrow [x][y]$	$[c] \leftarrow \mathsf{Mul}([x], [y])$
$c \leftarrow [x][y]$	$c \leftarrow \mathsf{MulPub}([x], [y])$
$[c] \leftarrow [\mathbf{x}][\mathbf{y}]$	$[c] \leftarrow \mathsf{Inner}([\mathbf{x}], [\mathbf{y}])$
$c \leftarrow [\mathbf{x}][\mathbf{y}]$	$[c] \leftarrow \mathsf{InnerPub}([\mathbf{x}], [\mathbf{y}])$

(a) Shorthand notation

Protocol	Rounds	Invocations	Security
PRandFld	0	0	N.A.
PRandInt	0	0	N.A.
PRandBit	1	1	Perfect
Inner	1	1	Perfect
MulPub	1	1	Perfect
InnerPub	1	1	Perfect
Inv	1	1	Perfect

(b) Complexity and security

Table 4.1: Efficient protocols based on Shamir sharing

we will give optimized circuits for important building blocks such as k-ary and prefix multiplication, prefix-or, bitwise comparison and binary addition.

We aim for the best performance. To minimize network delays we minimize the number of communication rounds, which in turn is minimized by minimizing the depth of the circuit. On the other hand, the communication complexity is minimized by minimizing the number of invocations, which is done by minimizing the number of (interactive) gates in the circuit. Typically reducing the number of rounds results in more gates to be executed in parallel and vice versa. Hence we need to balance between the number of rounds and the number of gates.

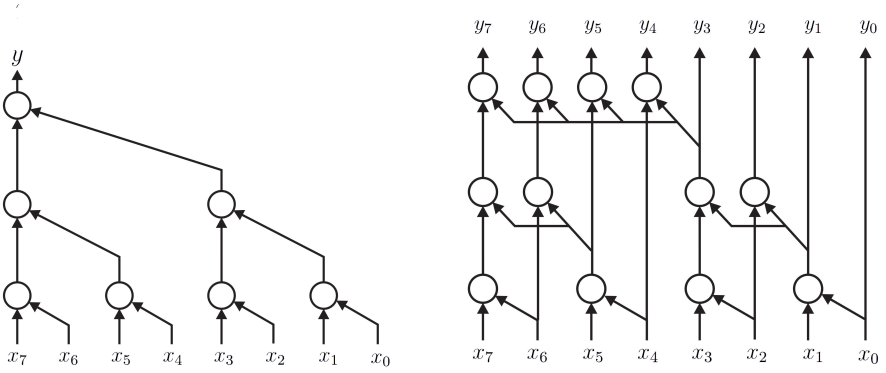
Assuming that $k-1$ is the minimal number of invocations of $\odot$, the communication complexity of both the k-ary and prefix operations is minimized by the straightforward k round circuit that computes $y_{j+1} = y_j x_{j+1}$ for $j = 0, \ldots, k-2$, where $y_0 = x_0$. However, in our applications a linear amount of rounds is undesirable.

Figure 4.1 shows arithmetic circuits with depth $\log k$, where k is a power of 2. Note that with respect to k-ary operations the circuit is optimal in the sense that only $k-1$ gates are evaluated. On the other hand with respect to prefix operations, the circuit requires more gates: $k/2\log(k)$. There are more circuits for prefix operations requiring $O(\log k)$ rounds and less gates, however with hidden constant larger than 1. For the sake of simplicity we apply the circuit given in Figure 4.1(b)

Protocol 4.14 implements the k-ary logarithmic depth circuit of Figure 4.1(a) on any $k > 1$. Suppose that op is an interactive protocol with α rounds and β invocations. Then Protocol 4.14 requires exactly $\alpha\lceil\log k\rceil$ rounds and $\beta(k-1)$ invocations.

Protocol 4.14: $[y] \leftarrow \mathsf{KOpL}([\mathbf{x}], \mathsf{op})$

1 $k \leftarrow \mathrm{Len}([\mathbf{x}])$;
2 **if** $k = 1$ **then**
3 **return** $[x]$;
4 **else**
5 $[a] \leftarrow \mathsf{KOpL}([x_0], \ldots, [x_{\lfloor k/2 \rfloor - 1}])$;
6 $[b] \leftarrow \mathsf{KOpL}([x_{\lfloor k/2 \rfloor}], \ldots, [x_{k-1}])$;
7 **return** $\mathsf{op}([a], [b])$



(a) k-ary (b) Prefix

Figure 4.1: Logarithmic circuits for k-ary and prefix operations

Protocol 4.15 implements the prefix logarithmic depth circuit of Figure 4.1(b) on any $k > 1$. The number of rounds is exactly $\alpha\lceil \log k \rceil$. Note that there is not a nice formula to count the number of gates when k is not a power of two. However, observe that an upper bound to the number of gates is equal to $\beta\lceil k/2 \rceil \lceil \log k \rceil$.

Protocol 4.15: $[y] \leftarrow \mathsf{PreOpL}([\mathbf{x}], \mathsf{op})$

```
1 k ← len([x]);
2 for i = 1, ..., ⌈log k⌉ do
3     foreach j ∈ {1, ..., ⌈k/2^i⌉} do parallel
4         ℓ1 ← 2^(i−1) + (j − 1)2^i − 1;
5         ℓ2 ← min{2^(i−1), k − ℓ1 − 1};
6         if ℓ2 > 0 then
7             foreach z ∈ {1, ..., ℓ2} do parallel
8                 [x_(ℓ1+z)] ← op([x_ℓ1], [x_(ℓ1+z)]);
9 return [x]
```

4.3.1 Multiplication

The k-ary multiplication $[y] = \prod_{i=0}^{k-1}[x_i]$ can be done via $\mathsf{KOpL}([\mathbf{x}], \mathsf{Mul})$ requiring $\log k$ rounds and $k-1$ invocations. We present a well known protocol by Bar-Ilan and Beaver [BB89] to create a circuit of constant depth for the computation of $[y]$, if x_i is invertible for all i.

The idea is to generate nonzero random values $[r_0], \ldots, [r_{k-1}]$ and compute their inverses $[r_0^{-1}], \ldots, [r_{k-1}^{-1}]$. Then, compute and open $m_0 = x_0 r_0$ and $m_i = r_{i-1}^{-1} x_i r_i$ for $i = 1, \ldots, k-1$. Since $x_i \neq 0$ and r_i are uniformly random and independent all values m_i are uniformly

random and independent. Moreover,

$$z = \prod_{i=0}^{k-1} m_i = r_{k-1} \prod_{i=0}^{k-1} x_i,$$

so that $[y] = z[r_{k-1}^{-1}]$.

Protocol 4.16: $[y] \leftarrow \mathsf{KMulC}([\mathbf{x}])$

1 **foreach** $i \in \{0, \ldots, k-1\}$ **do parallel**
2 **do**
3 $[r_i] \leftarrow \mathsf{PRandFld}(\mathbb{F})$;
4 $[s_i] \leftarrow \mathsf{PRandFld}(\mathbb{F})$;
5 $u_i \leftarrow [r_i][s_i]$; // 1 rnd, k inv
6 **while** $u_i = 0$;
7 **foreach** $i \in \{1, \ldots, k-1\}$ **do parallel** $[v_i] \leftarrow [r_i][s_{i-1}]$; // $k-1$ inv.
8 $[w_0] \leftarrow [r_0]$;
9 **foreach** $i \in \{1, \ldots, k-1\}$ **do** $[w_i] \leftarrow [v_i]u_{i-1}^{-1}$;
10 **foreach** $i \in \{0, \ldots, k-1\}$ **do parallel** $m_i \leftarrow [x_i][w_i]$; // 1 rnd, k inv.
11 $[y] \leftarrow [s_{k-1}]u_{k-1}^{-1} \prod_{j=0}^{k-1} m_j$;
12 **return** $[y]$

Protocol 4.17: $[y] \leftarrow \mathsf{PreMulC}([\mathbf{x}])$

1 **foreach** $i \in \{0, \ldots, k-1\}$ **do parallel**
2 **do**
3 $[r_i] \leftarrow \mathsf{PRandFld}(\mathbb{F})$;
4 $[s_i] \leftarrow \mathsf{PRandFld}(\mathbb{F})$;
5 $u_i \leftarrow [r_i][s_i]$; // 1 rnd, k inv
6 **while** $u_i = 0$;
7 **foreach** $i \in \{1, \ldots, k-1\}$ **do parallel** $[v_i] \leftarrow [r_i][s_{i-1}]$; // $k-1$ inv.
8 $[w_0] \leftarrow [r_0]$;
9 **foreach** $i \in \{1, \ldots, k-1\}$ **do** $[w_i] \leftarrow [v_i]u_{i-1}^{-1}$;
10 **foreach** $i \in \{0, \ldots, k-1\}$ **do parallel** $m_i \leftarrow [x_i][w_i]$; // 1 rnd, k inv.
11 $[y_0] \leftarrow [x_0]$;
12 **foreach** $i \in \{1, \ldots, k-1\}$ **do** $[y_i] \leftarrow [s_i]u_i^{-1} \prod_{j=1}^{i} m_j$;
13 **return** $[\mathbf{y}]$

Protocol 4.17 is an extension of Protocol 4.16 for computing the prefix products $[y_i] = \prod_{j=0}^{i} x_j$ for $i = 0 \ldots, k-1$, without extra communication costs. Indeed,

$$\prod_{j=0}^{i} m_j = r_i \prod_{j=0}^{i} x_j$$

for all $i = 0, \ldots, k-1$. The inversion of r_i can be computed by $s_i u_i^{-1}$.

Observe that Protocol 4.17 is very efficient and, while having constant rounds, its communication cost is very competitive to the logarithmic rounds protocol PreOpL. Indeed,

the latter requires $k/2 \log k$ invocations, while the constant rounds protocol requires $3k-1$ invocations. Hence, for vectors of length larger than $2^6 = 64$ the constant rounds solution for prefix multiplication has both less rounds and less invocations.

4.3.2 Prefix-Or

Consider $[\mathbf{b}]$, where $\mathbf{b} \in \{0,1\}^k$. We wish to compute

$$\bigvee_{j=0}^{i} [b_j],$$

for all $i = 0, \ldots, k-1$. This can be done via the logarithmic circuit of Figure 4.1(b) via $\mathsf{PreOpL}([\mathbf{b}], \mathsf{Or})$ in $\lceil \log k \rceil$ rounds and $\lceil k/2 \rceil \lceil \log k \rceil$ invocations, where

$$\mathsf{Or}([x],[y]) = [x] + [y] - [x][y].$$

Protocol 4.18 has constant rounds and is based on the observation that $x \vee y = 1$ if and only if $(1+x)(1+y)$ is even. Note that any positive integer k is even if the least significant bit of k is equal to zero, i.e., $\mathsf{LSB}(k) = 0$. The protocol for securely computing the least significant bit is described in the next section.

Protocol 4.18: $[\mathbf{x}] \leftarrow \mathsf{PreOrC}([\mathbf{b}])$

1 $([z_0], \ldots, [z_{k-1}]) \leftarrow \mathsf{PreMulC}([b_0]+1, \ldots, [b_{k-1}]+1)$; `// 2 rnd, 3k − 1 inv.`
2 $[z_0] \leftarrow [b_0]$;
3 **foreach** $i \in \{1, \ldots, k-1\}$ **do parallel**
4 $[x_i] \leftarrow 1 - \mathsf{LSB}([z_i])$; `// 1 rnd, 2k − 2 inv.`

With respect to efficiency observe that PreOpL has worse round complexity if $k > 2^4 = 16$ and worse communication complexity if $k > 2^{10} = 1024$.

4.3.3 Bitwise Comparison

Let $x \in \mathbb{Z}^+_{\langle k \rangle}$ be a positive integer. We denote the bitwise sharing of x by $[x]_B$, i.e.,

$$[x]_B = [x_{k-1}], \ldots, [x_0],$$

where the $x_i \in \{0,1\}$ for $i = 0, \ldots, k-1$ are such that $x = \sum_{i=0}^{k-1} x_i 2^i$. We also write $x = x_{k-1} \ldots x_0$.

This section shows how to efficiently compare secretly shared bitwise numbers $[x]_B$ and $[y]_B$. More precisely we will describe how to efficiently compute $[|x < y|_b]$.

An important tool for these protocols is the extraction of the least significant bit. Protocol 4.19 is from Schoenmakers and Tuyls [ST06]. Suppose that $\mathbb{F} = \mathbb{Z}_q$, where $q > 2^{k+\kappa+\log n}$.

Protocol 4.19: $[b] \leftarrow \mathsf{LSB}([x])$

1 $[r_0] \leftarrow \mathsf{PRandBit}(\mathbb{Z}_q)$; `// 1 rnd, 1 inv.`
2 $[r'] \leftarrow \mathsf{PRandInt}(\mathbb{Z}_q, k+\kappa-1)$;
3 $c \leftarrow \mathsf{Open}([x] + [r_0] + 2[r'])$; `// 1 rnd, 1 inv`
4 $[b] \leftarrow c_0 + [r_0] - 2c_0[r_0]$;
5 **return** b.

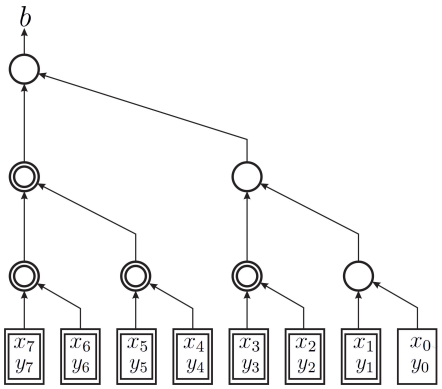


Figure 4.2: Circuit of BitLTL. Each circle corresponds to evaluating Eq. (4.7) and each double circle to evaluating in addition Eq. (4.8). Similarly, the squares correspond to evaluating Eq. (4.9) and Eq. (4.10).

4.3.3.1 Logarithmic Depth Circuit

We will present the result of [GSV07] to compute recursively the result of $[|x < y|_b]$ given shares of the bits of $[x]_B$ and $[y]_B$. Suppose that x is the concatenation $X_1||X_0$ and suppose that y is the concatenation $Y_1||Y_0$. Observe that

$$|x < y|_b = |X_1 < Y_1|_b + |X_1 = Y_1|_b \, |X_0 < Y_0|_b \tag{4.7}$$

and

$$|X = Y|_b = |X_1 = Y_1|_b \, |X_0 = Y_0|_b \, . \tag{4.8}$$

Note that when x and y are bits, then

$$|x < y|_b = y(1 - x) \tag{4.9}$$

and

$$|x = y|_b = 1 - (x - y)^2. \tag{4.10}$$

This leads to a circuit of the form of an k-ary operation given in Figure 4.1(a). Figure 4.2 shows the corresponding circuit for the case $[x]_B$ and $[y]_B$ are represented by 8 bits.

Protocol 4.20: $[z] \leftarrow \mathsf{BitLTL}([x]_B, [y]_B)$

1 $k \leftarrow \text{len}([x]_B)$;
2 **if** $k = 1$ **then**
3 $\quad [c] \leftarrow [x_0][y_0]$; `// 1 rnd, 1 inv.`
4 $\quad$ **return** $([y_0] - [c])$
5 **else**
6 $\quad k' \leftarrow \lceil k/2 \rceil$;
7 $\quad ([\ell_1], [\ell_2]) \leftarrow \mathsf{LTEQ}([x_{k-1}], \ldots, [x_{k'}], [y_{k-1}], \ldots, [y_{k'}])$;
8 $\quad [r] \leftarrow \mathsf{BitLTL}([x_{k'-1}], \ldots, [x_0], [y_{k'-1}], \ldots, [y_0])$;
9 $\quad [c_1] \leftarrow [\ell_1] + [r][\ell_2]$; `// 1 rnd, 1 inv.`
10 $\quad$ **return** $[c_1]$

Protocol 4.21: $([u], [v]) \leftarrow \mathsf{LTEQ}([x]_B, [y]_B)$

```
1  k ← len([x]_B);
2  if k = 1 then
3      [c] ← [x_0][y_0] ;                                   // 1 rnd, 1 inv.
4      return ([y_0] − [c], 1 − [x_0] − [y_0] + 2[c])
5  else
6      k' ← ⌈k/2⌉;
7      ([ℓ_1], [ℓ_2]) ← LTEQ([x_{k−1}], ..., [x_{k'}], [y_{k−1}], ..., [y_{k'}]);
8      ([r_1], [r_2]) ← LTEQ([x_{k'−1}], ..., [x_0], [y_{k'−1}], ..., [y_0]);
9      [c_1] ← [ℓ_1] + [r_1][ℓ_2] ;                         // 1 rnd, 1 inv.
10     [c_2] ← [ℓ_2][r_2] ;                                 // 1 inv.
11     return ([c_1], [c_2])
```

Remark 4.10. Notice that BitLTL requires k secure multiplications and 1 round less if x is public. Indeed, the k secure multiplications on the leafs to evaluate Eq. (4.9) and Eq. (4.10) are replaced by local multiplications. ◇

4.3.3.2 Constant Depth Circuit

Reistad provides in [Rei09] a constant depth circuit evaluating $[|x < y|_b]$ provided $[x]_B$ and $[y]_B$. The idea is to find the position of most significants bits that are different, i.e., compute $0 < i \leq k$ so that $x_i \neq y_i$ but $x_j = y_j$ for all $j = i + 1, \ldots, k - 1$. Once the position is located, the result is given by $x_i < y_i$.

Lemma 4.11. *Let*

$$e_j = y_j(1 - x_j)p_j, \tag{4.11}$$

where

$$p_i = 2^{\sum_{j=i+1}^{k-1} x_j \oplus y_j}.$$

Then $\sum_{i=0}^{k-1} e_i$ *is odd if and only if* $x < y$.

Proof. Let i be the position of the most significant differing bits. For all $j > i$ it follows that $p_j = 1$ and $y_j(1 - x_j) = 0$, so $e_j = 0$. For all $j < i$, on the other hand, p_j is a positive power of 2 and, therefore, e_j is either equal to zero or to a power of 2. Finally, since $p_i = 1$,

$$e_i = |x_i < y_i|_b = |x < y|_b .$$

Hence $E = \sum_{i=0}^{k-1} e_i$ is odd if and only if $x < y$. □

It follows that from E the result can be computed by

$$|x < y|_b = \mathsf{LSB}(E).$$

To compute p_j observe first that

$$x_i \oplus y_i = x_i + y_i - 2x_i y_i,$$

and second

$$p_i = 2^{\sum_{j=i+1}^{k-1} x_j \oplus y_j} = \prod_{j=i+1}^{k-1} (x_i \oplus y_i + 1)\,.$$

Hence $(p_{k-1}, \ldots, p_0)$ can be computed by a prefix multiplication.

As a slight optimization with respect to [Rei09] we observe that $y_i(1-x_i) = (x_i \oplus y_i)(1-x_i)$. Hence

$$e_i = (x_i \oplus y_i)(1 - x_i)p_i. \tag{4.12}$$

While this expression seems to be less efficient than Eq. (4.11) this representation saves k secure multiplications with respect to [Rei09] if x is public.

Indeed, let $d_i = x_i \oplus y_i$. Then

$$\begin{aligned} s_i &= p_i - p_{i+1} \\ &= 2^{\sum_{j=i+1}^{k} d_j} + 2^{\sum_{j=i+2}^{k} d_j} \\ &= 2^{\sum_{j=2+1}^{k} d_j} \left(2^{d_i} - 1\right) \\ &= p_{i+1} d_i. \end{aligned}$$

It follows by Eq. (4.12) that

$$e_i = s_i(1 - x_i).$$

While the expression $[e_i] \leftarrow [y_i](1 - x_i)[p_i]$ requires 1 secure multiplication we observe that $[s_i] \leftarrow [p_i] - [p_{i+1}]$ can be computed locally as well as $[e_i] \leftarrow [s_i](1 - x_i)$.

Protocol 4.22 evaluates $[|x < y|_b]$ given $[x]_B$ and $[y]_B$.

Protocol 4.22: $[b] \leftarrow \mathsf{BitLTC}([x]_B, [y]_B)$

1 **foreach** $i = 0, \ldots, k-1$ **do parallel**
2 $\quad [d_i] \leftarrow [x_i] + [y_i] - 2[x_i][y_i]$; `// 1 rnd, k inv.`
3 $([p_{k-1}], \ldots, [p_0]) \leftarrow \mathsf{PreMulC}([d_{k-1}] + 1, \ldots, [d_0] + 1)$; `// 2 rnd, 3k − 1 inv.`
4 **foreach** $i = 0, \ldots, k-2$ **do** $[s_i] \leftarrow [p_i] - [p_{i+1}]$;
5 $[s_{k-1}] \leftarrow [p_{k-1}] - 1$;
6 $[E] \leftarrow \sum_{i=0}^{k-1} [s_i](1 - [x_i])$; `// 1 rnd, k inv.`
7 $[b] \leftarrow \mathsf{LSB}([E])$; `// 1 rnd, 2 inv.`
8 **return** $[b]$.

4.3.4 Bitwise Addition

Given $[x]_B$ and $[y]_B$ we wish to compute $[s]_B = [x + y]_B$ using a binary addition circuit. We will use the same ideas from [GSV07] to build a circuit of logarithmic depth similar to Figure 4.1(b).

An addition circuit computes the bits s_i by

$$s_i = x_i + y_i + c_{i-1} - 2c_i, \tag{4.13}$$

where bits c_i, known as the *carry* bits, satisfy

$$c_i = x_i y_i + c_{i-1}\,(x_i + y + i - 2x_i y_i) \tag{4.14}$$

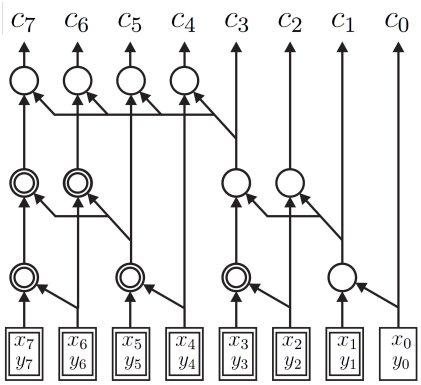


Figure 4.3: Circuit of PreCarry. Each circle corresponds to evaluating Eq. (4.15) and each double circle to evaluating in addition Eq. (4.16). Similarly the squares correspond to evaluating Eq. (4.17) and Eq. (4.18).

and $c_{-1} = 0$.

Let $c_k = \mathsf{carry}(x_{k-1} \dots x_0, y_{k-1} \dots y_0)$ denote the final carry when computing $x + y$. Suppose that x is the concatenation $X_1||X_0$ and suppose that y is the concatenation $Y_1||Y_0$, where X_1 and Y_1 are ℓ bits. Then

$$\mathsf{carry}(x, y) = \mathsf{carry}(X_1, Y_1) + \left|X_1 + Y_1 = 2^\ell - 1\right|_b \mathsf{carry}(X_0, Y_0) \tag{4.15}$$

and

$$\left|X + Y = 2^k - 1\right|_b = \left|X_1 + Y_1 = 2^\ell - 1\right|_b \left|X_0 + Y_0 = 2^{k-\ell} - 1\right|_b . \tag{4.16}$$

Note that when x and y are bits, then

$$\mathsf{carry}(x, y) = xy \tag{4.17}$$

and

$$|x + y = 1|_b = x \oplus y = x + y - 2xy. \tag{4.18}$$

Observe that

$$\left|a + b = 2^{k-1}\right|_b = \prod_{i=0}^{k-1} |a_i \oplus b_i|_b ,$$

for any k bits a and b, i.e., $a + b = 2^k - 1$ if and only if they differ in all bits.

A circuit of the form of a prefix operation given in Figure 4.1(b) is used to compute all carries $c_i = \mathsf{carry}(x_i \dots x_0, y_i \dots y_0)$ for all $i = 0 \dots k-1$. Figure 4.3 shows the corresponding circuit for the case $[x]_B$ and $[y]_B$ are represented by 8 bits. This circuit is slightly more optimal than the one suggested in [CH10a], saving k invocations.

Remark 4.12. Notice that AddBitwise requires k invocations and 1 round less if x is public. Indeed, the k invocations on the leafs to evaluate Eq. (4.13) and Eq. (4.14) are replaced by local computations. ◇

As an application we will use the addition circuit to decompose $[x]$ into its binary representation $[x]_B$.

Protocol 4.23: $[y] \leftarrow \mathsf{PreCarry}([x]_B, [y_B])$

$k \leftarrow \text{len}([x_B])$;
foreach $i = 0, \ldots, k-1$ **do**
  $[c_i] \leftarrow [x_i][y_i]$; // 1 rnd, k inv.
  $[d_i] \leftarrow [x_i] + [y_i] - 2[c_i]$;
for $i = 1, \ldots, \lceil \log k \rceil$ **do**
  foreach $j \in \{1, \ldots, \lceil k/2^i \rceil\}$ **do parallel**
    $\ell_1 \leftarrow 2^{i-1} + (j-1)2^i - 1$;
    $\ell_2 \leftarrow \min\{2^{i-1}, k - \ell_1 - 1\}$;
    if $\ell_2 > 0$ **then**
      foreach $z \in \{1, \ldots, \ell_2\}$ **do parallel**
        $[c_{\ell_1+z}] \leftarrow [c_{\ell_1+z}] + [d_{\ell_1+z}][c_{\ell_1}]$; // 1 rnd, ℓ_2 inv.
        if $j \neq 1$ **then** $[d_{\ell_1+z}] \leftarrow [d_{\ell_1+z}][d_{\ell_1}]$; // ℓ_2 inv.
return $([c_k], \ldots, [c_0])$

Protocol 4.24: $[s] \leftarrow \mathsf{AddBitwise}([x]_B, [y]_B)$

$c_{-1} \leftarrow 0$;
$([c_k], \ldots, [c_0]) \leftarrow \mathsf{PreCarry}([x]_B, [y]_B)$; // $\lceil \log k \rceil$ rnd, $k(\lceil \log k \rceil - 1)$ inv.
foreach $i \in \{0, \ldots, k-1\}$ **do** $[s_i] \leftarrow [x_i] + [y_i] + [c_{i-1}] - 2[c_i]$;
$[s_k] \leftarrow [c_k]$;
return $[s]_B$

4.3.5 Bit Decomposition

We use the result of [ST06] to apply the LSBs protocol to decompose $[x]$ into $[x]_B$. This protocol requires $q > 2^{k+\kappa+\log n}$ to provide statical security.

The LSBs gate compute the bits of x as follows. Let $r_0, \ldots, r_{k-1}$ be k uniformly random bits and let $r' \in_R \{0, \ldots, 2^{\kappa+\log n} - 1\}$. Let $c = 2^{k+1} + x - r + 2^k r'$. Then $0 < c < q$ and $c' = c \bmod 2^k = x - r \bmod 2^k$. It follows that $c' = x - r + \alpha 2^k$, where $\alpha \in \{0, 1\}$. Hence $c' + r = x + \alpha 2^k$ is a $k+1$ bit number where the k least significant bits are the bits of x.

Protocol 4.25: $[x]_B \leftarrow \mathsf{BitDec}([x], k)$

foreach $i \in \{0, \ldots, k-1\}$ **do parallel**
  $[r_i] \leftarrow \mathsf{PRandBit}(\mathbb{Z}_q)$; // 1 rnd, k inv.
$[r'] \leftarrow \mathsf{PRandInt}(\mathbb{Z}_q, \kappa)$;
$c \leftarrow \mathsf{Open}(2^{k+1} + [x] - \sum_{i=0}^{k-1} [r_i] 2^i + 2^k [r'])$; // 1 rnd, 1 inv.
$[x]_B \leftarrow \mathsf{AddBitwise}((c_{k-1}, \ldots, c_0), ([r_{k-1}], \ldots, [r_0]))$;
  // $\lceil \log k \rceil$ rnd, $k(\lceil \log k \rceil - 1)$ inv.
return $[x]_B$

Remark 4.13. Reistad and Toft give in [RT09b] a constant rounds bit decomposition protocol requiring 12 rounds and over $39.5k$ invocations. Hence their solution is more efficient with respect to the number of rounds if $k > 2^{12} = 4096$ and with respect to the number of invocations if $k > 2^{81}$. It follows that their bit-decomposition protocol may be more efficient for values of bit size of at least 4096.

Protocol	Rounds	Invocations	Security
$\mathsf{Op}([x],[y])$	α	β	Perfect
$\mathsf{KOpL}([\mathbf{x}],\mathsf{Op})$	$\alpha\log(k)$	$\beta(k-1)$	Perfect
$\mathsf{PerOpL}([\mathbf{x}],\mathsf{Op})$	$\alpha\log(k)$	$\beta k/2\log(k)$	Perfect
$\mathsf{KMulC}([\mathbf{x}])$	2	$3k-2$	Perfect
after preproc.	1	$2k-2$	
$\mathsf{PreMulC}([\mathbf{x}])$	2	$3k-1$	Perfect
after preproc.	1	k	
$\mathsf{LSB}([x])$	2	2	Statistical
after preproc.	1	1	
$\mathsf{PreOrC}([\mathbf{x}])$	3	$5k-3$	Statistical
after preproc.	2	$3k-2$	
$\mathsf{BitLTL}([\mathbf{x}],[\mathbf{y}])$	$\log(k)+1$	$3k-\log k-2$	Perfect
$\mathsf{BitLTL}(\mathbf{x},[\mathbf{y}])$	$\log k$	$2k-\log k-2$	Perfect
$\mathsf{BitLTC}([\mathbf{x}],[\mathbf{y}])$	4	$5k+1$	Statistical
after preproc.	3	$3k+2$	
$\mathsf{BitLTC}(\mathbf{x},[\mathbf{y}])$	3	$3k+1$	Statistical
after preproc.	2	$k+2$	
$\mathsf{AddBitwise}([x]_B,[y]_B)$	$\log(k)+1$	$k\log(k)$	Perfect
$\mathsf{AddBitwise}(x,[y]_B)$	$\log(k)$	$k(\log(k)-1)$	Perfect
$\mathsf{BitDec}([x],k)$	$\log(k)+2$	$k\log(k)+1$	Statistical
after preproc.	$\log(k)+1$	$k(\log(k)-1)+1$	

Table 4.2: Complexity and security of the k-ary and prefix protocols

Since we need bit decomposition of numbers that are much smaller, the presented logarithmic protocol is in our applications more efficient in both round and communication complexity. ◇

Table 4.2 list the protocols with their complexities and security. Note that the interactive generation of random bits can be preprocessed.

4.4 Integer Comparison

This section shows how to apply the previously discussed protocols to compute the result of comparing $[x]$ and $[y]$. Actually we will discuss protocols that compare a secret shared number with zero. This is sufficient as $x-y\leq 0$ if and only if $x\leq y$.

Overall the idea is from [ST06] where $x\in\mathbb{Z}_{\langle k+1\rangle}$ is masked with some large random integer r so that $0<x+2^k+r<q$ is statistically close to the distribution of r. Then one can use the bit representation of $[r]$ and $c=x+r$ to complete the computation.

4.4.1 Equality Tests

This section discusses two variants of equality comparison protocols. The first protocol will securely evaluate $[|x=0|_b]$, while the second protocol evaluates $|x=0|_b$ in the clear.

4.4.1.1 Equality with Secret Result

To compute $[|x = 0|_b]$ we present a new $\log^*(k)$ rounds protocol. The idea is basically to compute an k-ary Or on the bits of x, i.e., $|x = 0|_b = 1 - \bigvee_{i=0}^{k-1} x_i$. Viewing the bits as integers, observe that

$$\bigvee_{i=0}^{k-1} x_i = 0 \Leftrightarrow \sum_{i=0}^{k-1} x_i = 0,$$

which is again an integer consisting of at most $\log k$ bits. Thus, we could add those bits again resulting in an integer consisting of at most $\log \log k$ bits. If we continue, then at some stage we will be left with just one bit, which will be equal to $|x = 0|_b$.

Lemma 4.14. *Let x be a k-bit integer. Consider the sequence*

$$d^{(i)} = \sum_{j=0}^{\lceil \log d^{(i-1)} \rceil - 1} d_j^{(i)},$$

where $d_j^{(i)}$ denotes the j-th bit of $d^{(i)}$ and

$$d^{(1)} = \sum_{j=0}^{\lceil \log k \rceil - 1} x_j.$$

Then $|x = 0|_b = d^{(\ell)}$, where $\ell = \log^*(k)$.

Proof. Notice that

$$x = 0 \Leftrightarrow d^{(1)} = 0 \Leftrightarrow \sum_{i=0}^{\lceil \log(k) \rceil - 1} d_i^{(1)} = 0.$$

Let $l(x) = \lceil \log(x) \rceil$ and let $l^i(x)$ denote that l is i times applied on x. For example, $l^2(x) = l(l(x))$. Let $d^{(i)} = \sum_{j=0}^{l^i(k)} d_j^{(i-1)}$. If

$$\ell = \min\{\ell \in \mathbb{N} | l^i(k) = 1\},$$

then

$$|x = 0|_b = \left| d^{(1)} = 0 \right|_b = \cdots = d^{(\ell)}.$$

□

Let $c = x + 2^k + r$, then $x = 0$ if and only if $c_i = r_i$ for all $i = 0, \ldots k-1$ or, equivalently, $\sum_{j=0}^{k-1} (c_i \oplus r_i) = 0$.

With respect to the round complexity, note that all the random bits can be generated in one round. Hence EQZ requires $\log^*(k) + 1$ rounds. We require $\log^*(k) + 1$ openings and less than $\log^*(k) \log(k)$ secure multiplications for the random bit generation.

Protocol 4.26: $[b] \leftarrow \mathsf{EQZ}([x], k)$

1 **if** $k = 1$ **then return** $[x]$ **else**
2 $([r_0], \ldots, [r_{k-1}]) \leftarrow \mathsf{PRandBits}(\mathbb{Z}_q, k)$; `// 1 rnd, k inv.`
3 $[r'] \leftarrow \mathsf{PRandInt}(\mathbb{F}, \kappa + 1)$;
4 $c \leftarrow \mathsf{Open}([x] + \sum_{i=0}^{k-1} [r_i] 2^i + 2^k [r'])$; `// 1 rnd, 1 inv.`
5 $[d] \leftarrow \sum_{i=0}^{k-1} (c_i + [r_i] - 2c_i[r_i])$;
6 **return** $\mathsf{EQZ}([d], \lceil \log k \rceil)$

4.4.1.2 Equality with Public Result

If the result of the comparison is public we apply the comparison protocol by Franklin and Haber in [FH96]. The idea is simply to observe that $x = 0$ if and only if $xr = 0$, where $r \neq 0$. If r is uniformly random in $\mathbb{Z}_q^*$ then xr is uniformly random in $\mathbb{Z}_q$ which hides x perfectly if it is nonzero.

Protocol 4.27: $b \leftarrow \mathsf{EQZPub}([x])$

1 **do**
2 $[r] \leftarrow \mathsf{PRandFld}(\mathbb{Z}_q)$;
3 $[s] \leftarrow \mathsf{PRandFld}(\mathbb{Z}_q)$;
4 $z \leftarrow [r][s]$; `// 1 rnd, 1 inv`
5 **while** $z = 0$;
6 $u \leftarrow [r][x]$; `// 1 rnd, 1 inv.`
7 **return** $|u = 0|_b$

4.4.2 Less Than Zero Tests

To compute $|x < 0|_b$, we use the ideas from [ST06] and [RT09b]. Consider

$$b = \left((x \mod 2^k) - x \right) 2^{-k}.$$

Observe that if $x < 0$, then $(x \mod 2^k) = x + 2^k$. Hence $b = |x < 0|_b$.

Let $c = 2^k + x + r$. Then $0 < c < q$ and $x \bmod 2^k$ satisfies

$$x \bmod 2^k = (c - r) \bmod 2^k = (c \bmod 2^k) - (r \bmod 2^k) + 2^i \left| (c \bmod 2^k) < (r \bmod 2^k) \right|_b .$$

To compute $\left|(c \bmod 2^i) < (r \bmod 2^i)\right|_b$ a binary circuit of Section 4.3.3 is used.

Protocol 4.28: $[b] \leftarrow \mathsf{LTZ}([x])$

1 $[z] \leftarrow \mathsf{Mod2m}([x], k, k)$; `// Protocol 4.29`
2 $[b] \leftarrow ([z] - [x])2^k)$;
3 **return** $[b]$

Notation	Protocol
$[c] \leftarrow [x] < [y]$	$[c] \leftarrow \mathsf{LTZ}([x] - [y])$
$[c] \leftarrow [x] \leq [y]$	$[c] \leftarrow 1 - \mathsf{LTZ}([y] - [x])$
$[c] \leftarrow [x] > [y]$	$[c] \leftarrow 1 - \mathsf{LTZ}([x] - [y])$
$[c] \leftarrow [x] \geq [y]$	$[c] \leftarrow \mathsf{LTZ}([y] - [x])$
$[c] \leftarrow [x] = [y]$	$[c] \leftarrow \mathsf{EQZ}([x] - [y])$
$c \leftarrow [x] = [y]$	$[c] \leftarrow \mathsf{EQZPub}([x] - [y])$

(a) Shorthand notation

Protocol	Rounds	Invocations	Security
$\mathsf{EQZ}([x])$	$\log^*(k) + 1$	$< \log(k) \log^*(k)$	Statistical
after preproc.	$\log^*(k)$	$\log^*(k)$	
$\mathsf{EQZPub}([x])$	2	2	Perfect
after preproc.	1	1	
$\mathsf{LTZ}([x])$ Log:	$\log(k) + 3$	$4k - \log k - 1$	Statistical
after preproc.	$\log(k) + 2$	$3k - \log k - 1$	
$\mathsf{LTZ}([x])$ Const:	4	$4k + 2$	Statistical
after preproc.	3	$k + 3$	

(b) Complexity and security

Table 4.3: Integer comparison protocols

Protocol 4.29: $[b] \leftarrow \mathsf{Mod2m}([x], k, m)$

1 $[\mathbf{r}] \leftarrow \mathsf{PRandBits}(\mathbb{Z}_q, m)$; `// 1 rnd,` m `inv.`
2 $[r'] \leftarrow \mathsf{PRandInt}(\mathbb{Z}_q, \kappa + k - m)$;
3 $c \leftarrow \mathsf{Open}\left(2^k + [x] + \sum_{i=1}^{m} [r_i] 2^{i-1} + 2^m [r']\right)$; `// 1 rnd, 1 inv.`
4 $c' \leftarrow c \mod 2^m$;
5 $[b] \leftarrow \mathsf{BitLT}(c', [r_m], \ldots, [r_1])$; `// Protocol 4.20 or Protocol 4.22.`
6 $[x'] \leftarrow c' - \sum_{i=1}^{m} [r_i] 2^{i-1} + 2^m [b]$;
7 **return** $[x']$

We will use the notation given in Table 4.3(a) to denote invocation of the corresponding protocols. Table 4.3(b) presents the efficiency and security of the protocols that are discussed in this section.

From Table 4.3(b) it follows that the constant rounds solution for inequality comparison is almost as efficient with respect to communication complexity as the logarithmic version. Furthermore the constant round inequality allows more efficiency gain by preprocessing.

4.5 Fixed Point Arithmetic

In this section we consider the following set of numbers:

$$\mathbb{Q}_{\langle k,f \rangle} = \left\{ x \in \mathbb{Q} \,\middle|\, x = \bar{x} 2^{-f}, \bar{x} \in \mathbb{Z}_{\langle k \rangle} \right\},$$

where f is called the *resolution* and $e = k - f$ is called the *range* of the fixed point representation.

For all $y \in \mathbb{Q}_{\langle k,f \rangle}$ note that $2^f y \in \mathbb{Z}_{\langle k \rangle}$. Therefore, we can use the mapping $\phi : \mathbb{Z}_{\langle k \rangle} \to \mathbb{Z}_Q$ from Section 4.2.1 to map elements from $\mathbb{Q}_{\langle k,f \rangle}$ into elements of $\mathbb{Z}_q$ in such a way that the arithmetic operations of the simplex algorithm are preserved.

We will show how to apply arithmetic over $\mathbb{Z}_q$ over representations of both $\mathbb{Z}_{\langle k \rangle}$ and $\mathbb{Q}_{\langle k,f \rangle}$.

Addition and subtraction of fixed point numbers Let $x, y \in \mathbb{Q}_{\langle k,f \rangle}$. Then

$$z = \phi(2^f x) \pm \phi(2^f y) = \phi(2^f (x \pm y)),$$

so $2^{-f}\phi^{-1}(z) = x \pm y$.

Multiplication of a fixed point number with an integer Let $x \in \mathbb{Q}_{lrak,f}$ and $y \in \mathbb{Z}_{\langle k \rangle}$. Then

$$z = 2^f \phi(x)\phi(y) = \phi(2^f xy),$$

so $2^{-f}\phi^{-1}(z) = xy$.

Addition and subtraction of a fixed point number with an integer Let $x \in \mathbb{Q}_{\langle k,f \rangle}$ and $y \in \mathbb{Z}_{\langle k \rangle}$. Then,

$$z = \phi(2^f x) \pm \phi(2^f y) = \phi(2^f x \pm 2^f y),$$

so $2^{-f}\phi^{-1}(z) = x \pm y$.

Multiplication of fixed point numbers Multiplication of fixed point numbers is not straightforward. Indeed, let $x, y \in \mathbb{Q}_{\langle k,f \rangle}$, then

$$z = \phi(2^f x)\phi(2^f y) = \phi(2^{2f} xy),$$

so $a = 2^{-f}\phi^{-1}(z) = 2^f xy$. The absolute error of a satisfies

$$|a - xy| < 2^{-f}.$$

The multiplication of x and y is performed as follows. First, compute

$$z = \phi(2^f x)\phi(2^f y) = \phi(2^{2f} xy).$$

Second, compute

$$a = \left\lfloor \frac{\phi^{-1}(z)}{2^f} \right\rfloor + u,$$

where $u \in [0, 1]$ is chosen depending on the rounding.

For any $x \in \mathbb{Q}_{\langle k,f \rangle}$ we denote with $[x]$ the Shamir share of the value $2^f x \in \mathbb{Z}_{\langle k \rangle}$.

Protocol 4.30: $[z] \leftarrow \mathsf{MulFP}([x], [y])$

1 $[z] \leftarrow [x][y]$; // 1 rnd, 1 inv.
2 $[z] \leftarrow \mathsf{TruncPr}([z], k + f, f)$; // 2 rnd, f inv.
3 **return** $[z]$

Table 4.4 presents the complexity and absolute error of protocols for fixed point arithmetic.

x	y	Operation	Protocol	complexity	error
$\mathbb{Q}_{\langle k,f\rangle}$	$\mathbb{Q}_{\langle k,f\rangle}$	$+$	$[x]+[y]$	N.A	0
$\mathbb{Q}_{\langle k,f\rangle}$	$\mathbb{Z}_{\langle k\rangle}$	$+$	$[x]+2^f[y]$	N.A.	0
$\mathbb{Q}_{\langle k,f\rangle}$	$\mathbb{Z}_{\langle k\rangle}$	$\cdot$	$[x][y]$	N.A.	0
$\mathbb{Q}_{\langle k,f\rangle}$	$\mathbb{Q}_{\langle k,f\rangle}$	$\cdot$	$\mathsf{FPMul}([x],[y])$	2 rnd, 2 inv.	$\delta < 2^{-f}$

Table 4.4: Complexity and error of basic protocols for fixed point arithmetic

4.5.1 Truncation

To truncate the f least significant bits of x one computes $d = x \bmod 2^f$ so that $x - d$ is a multiple of 2^f. Moreover $(x-d)2^{-f} = \lfloor 2^{-f}x \rfloor$. This shows correctness of Protocol 4.31.

Protocol 4.31: $[y] \leftarrow \mathsf{Trunc}([x], k, f)$

1 $[d] \leftarrow \mathsf{Mod2m}([x], k, f)$; `// 4 rnd, 4f + 2 inv.`
2 $[y] \leftarrow ([x] - [d])2^{-f}$;
3 **return** $[y]$.

Protocol 4.32 improves the efficiency by allowing probabilistic rounding by removing the call to BitLT in Mod2m. Let r be a uniformly random f bit value and r' a random value in $\{0, \ldots, 2^{\kappa+k+\log(n)-f}\}$ and $c = 2^{k-1} + x + r + 2^f r'$. Then

$$c' = c \bmod 2^f = x + r \bmod 2^f = x \bmod 2^f + r - u2^f,$$

where $u = \left|(x \bmod 2^f) + r \geq 2^f\right|_b$.

Hence

$$(x - c' + r)2^{-f} = (x - (x \bmod 2^f))2^{-f} + u = \lfloor 2^{-f}x \rfloor + u.$$

Note that the value of $u \in \{0, 1\}$ depends on r and is, therefore, random. It satisfies

$$\mathbb{P}[u = 1] = \mathbb{P}\left[(x \bmod 2^f) + r \geq 2^f\right] = \mathbb{P}\left[r \geq 2^f - (x \bmod 2^f)\right].$$

Protocol 4.32: $[y] \leftarrow \mathsf{TruncPr}([x], k, f)$

1 $[\mathbf{r}] \leftarrow \mathsf{PRandBits}(\mathbb{Z}_q, f)$; `// 1 rnd, f inv.`
2 $[r'] \leftarrow \mathsf{PRandInt}(\mathbb{Z}_q, \kappa + k + 1 - f)$;
3 $c \leftarrow \mathsf{Open}\left(2^k + [x] + \sum_{i=0}^{f-1}([r_i]2^i) + 2^f[r']\right)$; `// 1 rnd, 1 inv.`
4 $c' \leftarrow c \mod 2^f$;
5 $[d] \leftarrow ([x] - c' + \sum_{i=1}^{m}[r_i]2^{i-1})2^{-f}$;
6 **return** $[d]$

4.5.2 Division

This section shows how to compute the division of x and y using the Newton-Raphson method. We will first discuss the Newton-Raphson method. Then, we will provide and analyze protocols that securely computes $a \approx x/y$.

Newton-Raphson

Given a differentiable function $f : \mathbb{R} \to \mathbb{R}$ the Newton-Raphson method iteratively approximates a zero z of f as follows. Let z_0 be an initial approximation of z, then the sequence

$$z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)}$$

is computed. If the initial approximation z_0 is close enough to z then z_i converges to z quadratically.

We will first provide a function f that has $1/y$ as a zero and show that the Newton-Raphson method converges quadratically to $1/y$. Then, we will show how to initialize the Newton-Raphson method.

To compute $z = 1/y$ consider the function $f(z) = y - 1/z$. Indeed $f(z) = 0$ implies that $y = 1/z$ or, equivalently, $z = 1/y$. From $f'(z) = 1/z^2$ it follows that the recurrence relation becomes

$$z_{i+1} = z_i - \frac{y - 1/z_i}{1/z_i^2} = z_i(2 - z_i y). \tag{4.19}$$

Let $\epsilon_i = z - z_i$ be the absolute error for $i = 0, 1, 2, \ldots$. Then from Eq. (4.19) we get that

$$z_{i+1} = z - \epsilon_{i+1} = (z - \epsilon_i)(2 - (z - \epsilon_i)y) = z - \epsilon_i^2.$$

Hence $\epsilon_{i+1} = \epsilon_i^2 = \epsilon_0^{2^i}$, implying that the error decreases quadratically if $|\epsilon_0| < 1$.

The hard part is to find a good initial approximation. The classical approach is to normalize y into c so that $c \in [1/2, 1)$. It follows that $c = y2^{-\lfloor \log y \rfloor - 1}$ and $1/c \in (1, 2]$. Therefore, $z_0 = 3/2$ has error $\epsilon_0 \leq 1/2$ (1 exact bit) so after iteration i the relative error is at most $\epsilon_0^{2^i} < 2^{-2^i}$ (implying 2^i exact bits). Hence, after $\theta = \lceil \log(k+1) \rceil$ iterations we have $k + 1$ exact bits of $1/c$.

A better initial approximation is suggested by [EL03], which sets $z_0 = 2.9142 - 2c$, with relative error $\epsilon_0 < 0.08578$ (at least $\log(\epsilon_0) = 3.5$ exact bits). Hence, $k+1$ bits are obtained after $\theta = \lceil \log \frac{k+1}{3.5} \rceil$ iterations.

Computing x/y

Protocol 4.33 computes the reciprocal of $[x]$, where $x \in \mathbb{Q}_{\langle k+1,k \rangle}$ and $1/2 \leq x < 1$. It uses the Newton-Raphson method being initialized following [EL03]. The number of iterations θ is computed so that the absolute error is bounded by 2^{-k}.

Since we use integer arithmetic and all values are in $\mathbb{Q}_{\langle k+1,k \rangle}$, all values are multiplied by 2^k. Hence after each iteration we need to truncate 2^{2k} bits.

Protocol 4.33: $[y] \leftarrow \mathsf{RecItNR}([x], k)$

1 $\theta \leftarrow \lceil \log \frac{k+1}{3.5} \rceil$;
2 $[y] \leftarrow 2.9142 \cdot 2^k - 2[x]$;
3 **foreach** $i = 1, \ldots, \theta$ **do**
4 $\quad [y] \leftarrow [y](2 \cdot 2^{2k} - [y][x])$; `// 2 rnd, 2 inv.`
5 $\quad [y] \leftarrow \mathsf{TruncPr}([y], 3k + 1, 2k)$; `// 2 rnd, 2k+1 inv.`
6 **return** $[z]$;

Let $x \in \mathbb{Q}_{\langle k-f,f\rangle}$ and $y \in \mathbb{Q}_{\langle k,f\rangle}$. In the following we will show how to compute $a \approx x/y$. Then we will compute the absolute error.

To compute $a \approx x/y$ one computes $a \approx 1/y$ first. To use the Newton-Raphson method we need to normalize y, i.e., compute v so that $1/2 \leq v'y < 1$. Let m be such that $2^{m-1} \leq 2^f y < 2^m$. Then $v' = 2^{f-m}$. Observe that $2^{k-1} \leq 2^{k+f-m}y < 2^k$ is integer. Let $c = v'y$. Then $c \in \mathbb{Q}_{\langle k+1,k\rangle}$ and $1/2 \leq c < 1$.

Let v be such that $2^k c = 2^f yv$. Then v is called the *normalization factor*. Note that in this case $v = 2^{k-m}$.

Let $a \approx 1/c$. So $a \approx 1/(v'y)$ and, therefore, $v'a = 2^{f-m}a \approx 1/y$. Hence, $ax2^{f-m} \approx x/y$.

With respect to the truncation in Protocol 4.35, we have from $a \in \mathbb{Q}_{\langle k+1,k\rangle}$ and $x \in \mathbb{Q}_{\langle k-f,f\rangle}$ that

$$ax2^{f-m}2^f = 2^k a 2^f x 2^{k-m} 2^{-(2k-f)}$$

and $2^k a 2^f xv < 2^{3k}$. Hence $axv \in \mathbb{Q}_{\langle 3k,2k-f\rangle}$.

With respect to the error, let $1/y - a = \delta < 2^{-k}$. We compute a bound on the absolute error by

$$x/y - ax2^{f-m} = (1/y - a)x2^{f-m} = \delta x 2^{f-m} < 2^{-k+k-2f+f-m} < 2^{-f-m} < 2^{-f}.$$

Protocol 4.34: $[z] \leftarrow \mathsf{DivNR}([x],[y],k,f)$

1 $[c] \leftarrow \mathsf{Rec}([y],k)$;
2 $[z] \leftarrow [x][c]$;
3 $[z] \leftarrow \mathsf{TruncPr}([z],3k,2k-f)$;
4 **return** $[z]$;

Protocol 4.35: $[x] \leftarrow \mathsf{Rec}([x],k)$

1 $([c],[v]) \leftarrow \mathsf{Norm}([x],k)$;
2 $[a] \leftarrow \mathsf{RecItNR}([c],k)$;
3 $[z] \leftarrow [a][v]$;
4 **return** $[z]$

Normalization

To normalize $y \in \mathbb{Z}_{\langle k\rangle}$ we compute c and v so that $2^{k-1} \leq c < 2^k$ and $c = yv$. Note that

$$v = 2^{k-(\lfloor \log y \rfloor + 1)}.$$

Let y be decomposed in k bits $y_{k-1} \ldots y_0$. Then v is equal to 2^{k-i}, where i is the largest index such that $y_i \neq 0$.

Protocol	Rounds	Invocations	Security	error
$\mathsf{TrunPr}([x], f)$	2	$f+1$	Statistical	$< 2^{-f}$
after preproc.	1	1		
$\mathsf{Rec}([y], k)$	$O(\log(k))$	$O(k \log k)$	Statistical	$< 2^{-k+m}$
$\mathsf{Div}([x], [y], k, f)$	$O(\log(k))$	$O(k \log k)$	Statistical	$< 2^{-f}$

Table 4.5: Complexity, security and error of protocols for truncation and division

Protocol 4.36: $([c], [v]) \leftarrow \mathsf{Norm}([x], k)$

1 $[x]_B \leftarrow \mathsf{BitDec}([x], k)$; // $\lceil \log k \rceil$ rnd, $k(\lceil \log k \rceil) + 1$ inv.
2 $([d_{k-1}], \ldots, [d_0]) \leftarrow \mathsf{PreOrC}([x]_B)$; // 3 rnd, $5k - 3$ inv.
3 $[b_{k-1}] \leftarrow [d_{k-1}]$;
4 **foreach** $i = 0, \ldots, k-2$ **do**
5 $\quad [b_i] \leftarrow [d_i] - [d_{i+1}]$;
6 $[v] \leftarrow \sum_{i=0}^{k-1} [b_{k-1-i}] 2^i$;
7 $[c] \leftarrow [x][v]$; // 1 rnd, 1 inv.
8 **return** $([c], [v])$

Table 4.5 lists the protocols with respect to fixed point arithmetic with their complexity, security and absolute errors, where m denotes the smallest integer such that $2^{k-1} \leq 2^m x < 2^k$.

4.6 Secret Indexing

In this section we discuss methods to obliviously perform operations in multiple dimensional arrays. Basically, the idea is to represent an index as an unary array of the right length, where every entry is shared [Tof09]. Let $\mathbf{i}$ denote the i-th length n unary array, i.e., $i_j = |i = j|_b$ for all $j = 1, \ldots, n$. We call $[\mathbf{i}]$ the *secret index representing* i.

Let $[\mathbf{i}]$ be a secret index representing i. Then, from the shared list $[\mathbf{x}] = [x_1], \ldots, [x_n]$ the i-th entry is obliviously selected by computing the inner product $[x_i] = [\mathbf{x}][\mathbf{i}]$.

To write a shared value $[\alpha]$ at position $[\mathbf{i}]$ in array $[\mathbf{x}]$ we execute Protocol 4.37 that for each entry computes

$$[x_j] \leftarrow [i_j][\alpha] + (1 - [i_j])[x_j]. \tag{4.20}$$

Indeed, since $\mathbf{i}$ is a secret index representing i Eq. (4.20) implies that $[x_j] \leftarrow [x_j]$ for all $j \neq i$ and $[x_i] \leftarrow [\alpha]$.

Protocol 4.37: $[\mathbf{x}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{x}], [\mathbf{i}], [\alpha])$

1 **foreach** $j = 1, \ldots, n$ **do**
2 $\quad [x_j] \leftarrow [x_j] + ([\alpha] - [x_j])[i_j]$;
3 **return** $[\mathbf{x}]$.

A more advanced operation is to obliviously add or delete an element from or to a shared list $[\mathbf{x}]$. Let $[\mathbf{T}]$ be a matrix and suppose we wish to securely delete column k being

represented by the secret index $[\mathbf{k}]$. Let $\mathbf{T}'$ be the resulting tableau. Then

$$\mathbf{T}'_j = \begin{cases} \mathbf{T}_j, & \text{if } j < k, \\ \mathbf{T}_{j+1}, & \text{if } j \geq k. \end{cases}$$

Protocol 4.38 shows how to securely delete a column from $\mathbf{T}$ given secret index $\mathbf{k}$ using the observation that $|j \geq k|_b = \sum_{i=1}^{j} k_j$.

Protocol 4.38: $[\mathbf{T}] \leftarrow \mathsf{DelCol}([\mathbf{T}], [\mathbf{k}])$

$[x]_1 \leftarrow [k_1]$;
for $i = 2, \ldots, n+1$ **do**
    $[x_i] \leftarrow [x_{i-1}] + [k]_i$;
foreach $i = 1, \ldots, n$ **do parallel**
    $[\mathbf{T}'_i] \leftarrow (1 - [x]_i)[\mathbf{T}_i] + [x_i][\mathbf{T}_{i+1}]$; // 1 rnd, $n(m+1)$ inv.
return $[\mathbf{T}']$.

To remove m columns one could apply Protocol 4.38 m times successively. However, this would lead to m interactive rounds. We propose an alternative solution, where the m columns are deleted in $\log^*$ rounds (Protocol 4.39)

Protocol 4.39: $[\mathbf{T}] \leftarrow \mathsf{DelCols}([\mathbf{T}], [\mathbf{w}])$

Input: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{m+1 \times m+n+1}$, $[\mathbf{w}] \in \{0,1\}^{n+m+1}$.
Output: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+1}$.

$[\mathbf{v}] \leftarrow \mathbf{1} - [\mathbf{w}]$;
$[d_0] \leftarrow [0]$;
for $i = 1, \ldots, n+m$ **do**
    $[d_i] \leftarrow [d_{i-1}] + [v_i]$;
foreach $i \in \{1, \ldots, n+m\}$ **do parallel**
    $[x_i] \leftarrow [v_i][d_i]$; // 1 rnd, $n+m+1$ inv.
foreach $i \in \{1, \ldots, m\}$ **do parallel**
    foreach $j \in \{1, \ldots, n+m\}$ **do parallel**
        $[y_{ij}] \leftarrow [x_j = i]$; // $\log^*(m)$ rnd, $m^2(n+m)\log m$ inv.
    $[\mathbf{T}'_i] = [\mathbf{T}]\mathbf{y_i}$; // 1 rnd, m inv.
return $\mathbf{T}'$

To convert a shared number $[i]$ into a secret index $[\mathbf{i}]$ of length n, we execute Protocol 4.40. The idea to locally compute coefficients of a $n-1$ degree polynomial $p_i : \{1, \ldots, n\} \to \{0,1\}$, where

$$p_i(j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Note that for all $i = 1, \ldots, n$

$$\begin{aligned} p_i(x) &= \prod_{j=1, j\neq i}^{n} \frac{x-j}{i-j} \\ &= a_{i0} + \sum_{j=1}^{n-1} a_{ij} x^j \\ &= \mathbf{a}_i \cdot (1, x, x^2, \ldots, x^{n-1}), \end{aligned}$$

where the coefficients $\mathbf{a}_i$ can be computed using local computations only.

Protocol 4.40: $[\mathbf{i}] \leftarrow \mathsf{ConvertUnary}([i], n)$

1 $[\mathbf{I}] \leftarrow (1, [i], \ldots, [i])$;
2 $[\mathbf{v}] = (1, i, i^2, \ldots, i^{n-1}) \leftarrow \mathsf{PreMulC}([\mathbf{I}])$; `// 2 rnd, 3n − 2 inv.`
3 **foreach** $j = 1, \ldots, n$ **do**
4 $\quad [i_j] \leftarrow \mathbf{a}_j[\mathbf{v}]$;
5 **return** $[\mathbf{v}]$

Chapter 5

Secure Linear Programming

This chapter discusses secure implementations of the simplex algorithm following the approach of [Tof07, CH10b]. We provide a secure protocol for each simplex variant presented in Chapter 3. The building blocks of the previous chapter will be used extensively. We follow the same structure as in Chapter 3.

The first section discusses the secure implementation of the simplex iterations. We provide a complete overview of efficient secure protocols implementing each variant that is presented in Section 3.2. To this end, we present several ideas to efficiently select the pivot column and pivot row, and how to update the tableaus. We show the balances between security and efficiency. For example, hiding the number of iterations of the simplex algorithm has the consequence that the secure implementations have a worst case number of iterations. Hence these implementations would require an exponential number of iterations.

The second section discusses the secure implementation of the simplex initialization. We provide a complete overview of efficient secure protocols implementing each variant that is presented in Section 3.3. We address issues that arise when securely implementing the initialization of phase II in the two-phase simplex algorithm. In addition, we show that to solve security issues we have to change the artificial linear program with one artificial variable.

The third section presents the secure validation of the results returned by the simplex algorithm. We show how to extract a certificate of either optimality, non-feasibility, or unboundedness from the simplex tableau and (co-)basis returned by the simplex algorithm. Then, we show how to securely validate the certificates.

Finally, the last section provides a comparison between all variants with respect to security and efficiency. We show that there is no variant that is best with respect to both security and efficiency.

This chapter will present descriptions of how to build the secure protocols. The detailed protocols are presented in Appendix A.

Representing the Linear Program

In Chapter 3 we considered linear programs in standard form:

$$\begin{array}{rlcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \mathbf{Ax} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}. \end{array} \tag{5.1}$$

The simplex algorithm requires that $\mathbf{A}$ has full row rank. We showed that this requirement may be dropped if one applies the standard two-phase simplex algorithm, or the corresponding big-M method.

To avoid issues due to the rank of $\mathbf{A}$ we consider in this chapter linear programs in the following form

$$\begin{array}{rrcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \mathbf{A'x} & \leq & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{5.2}$$

where $\mathbf{A'} \in \mathbb{Z}^{m \times n}$, $\mathbf{c} \in \mathbb{Z}^n$ and $\mathbf{b} \in \mathbb{Z}^m$, and $\mathbf{x} \in \mathbb{Q}^n$. Indeed, any pair of numbers x, y, satisfies that $x \leq y$ if and only if there exists some $a \geq 0$ such that $x = y + a$. Hence, LP (5.2) can be written in standard form by

$$\begin{array}{rrcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A'} & \mathbf{I}_m \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{x}_s \end{pmatrix} & = & \mathbf{b}, \\ & \mathbf{x}, \mathbf{x}_s & \geq & \mathbf{0}, \end{array} \tag{5.3}$$

where $\mathbf{x}_s$ are called the *slack variables* and

$$\mathbf{A} := \begin{pmatrix} \mathbf{A'} & \mathbf{I}_m \end{pmatrix}$$

has rank m. By construction $\mathbf{A}$ is in *canonical form* and the algorithms with respect to solving the artificial LP with one artificial variable can be applied (see Section 3.3.2).

In this chapter a linear program is in *standard* form if it is of the form Eq. (5.3).

5.1 Secure Simplex Iterations

Consider an LP in standard form. Let $\mathbf{T}$ be a tableau corresponding to basis $\mathbf{s}$. Recall that the basis matrix $\mathbf{B} = \mathbf{A_s}$ is invertible and $\mathbf{T}$ can be written as

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix}.$$

A corresponding co-basis is equal to $\mathbf{u} = (u_1, \ldots, u_n)$, where $u_i \notin \mathbf{s}$ for each i. A basic feasible solution $\mathbf{x}$ satisfies $\mathbf{x_u} = \mathbf{0}$ and $\mathbf{x_s} = \mathbf{B}^{-1}\mathbf{b}$.

Recall furthermore that with respect to integer pivoting the tableau $\mathbf{T}$ can be written as

$$\mathbf{T} = |\det(\mathbf{B})| \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix}.$$

With respect to security, a secure implementation of the simplex algorithm should hide the fact whether the current solution is optimal at the beginning of every iteration. Unfortunately, we showed in Chapter 3 that the simplex algorithm requires exponentially many iterations in the worst case. Therefore, revealing no data about the number of iterations implies that the protocol should have worst case running time implying exponentially many iterations. This makes the protocol infeasible in practice. To avoid exponentially many iteration we choose to reveal the number of iterations.

5.1.1 Large Tableau Simplex

In Section 3.2 we showed that the large tableau simplex algorithm consists of the following three steps in each iteration, where $q = \det(\mathbf{B})$:

1. *Column step:* Find an $\ell \in \{1, \ldots, n+m\}$ such that $t_{m+1,j} < 0$. If no such ℓ exists, then output the current solution being the optimum.

2. *Row step:* Find a $k \in \{1, \ldots, m\}$ such that

$$k = \text{argmin} \left\{ \frac{t_{i,n+m+1}}{t_{i,\ell}} \middle| t_{i,\ell} > 0, \text{ and } i \in \{1, \ldots, m\} \right\}.$$

   If $t_{k\ell} \leq 0$ for all $k \in \{1, \ldots, m\}$, then stop and report that the LP is unbounded.

3. *Update tableau and basis:* In case of rational pivoting, pivot on $t_{k\ell}$ by updating $\mathbf{T}$ according to:

$$t_{ij} = \begin{cases} t_{ij} - t_{i\ell} t_{kj} / t_{k\ell}, & \text{if } i \neq k, \\ t_{ij} / t_{k\ell}, & \text{if } i = k. \end{cases}$$

   Otherwise, in case of integer pivoting, pivot on $t_{k\ell}$ by updating $\mathbf{T}$ according to:

$$t_{ij} = \begin{cases} \left(t_{ij} t_{k\ell} - t_{i\ell} t_{kj}\right) / q, & \text{if } i \neq k, \\ t_{ij}, & \text{if } i = k. \end{cases}$$

   The basis is updated by replacing s_k with ℓ.

We will describe how to efficiently and securely implement each step. The detailed protocols can be found in Appendix A.1.1.

5.1.1.1 Column Step

Given tableau $\mathbf{T}$ this step selects the pivot column $\mathbf{T}_\ell$, where $\ell \in \{1, \ldots, n+m\}$ for which $t_{(m+1)\ell} < 0$. If no such ℓ exists, then the simplex algorithm stops reporting that the current solution corresponding to $\mathbf{T}$ is optimal.

We describe protocols for both Dantzig's original pivoting rule as Bland's pivoting rule.

Dantzig's original pivoting rule is to take $\ell \in \{1, \ldots, m+n\}$ such that

$$t_{(m+1)\ell} = \min \left\{ t_{(m+1)j} | t_{(m+1)j} < 0 \text{ and } j \in \{1, \ldots, n+m\} \right\}.$$

Consider Protocol 5.1, which is due to Toft in [Tof07]. It returns the secret index $[\boldsymbol{\ell}]$ and the corresponding minimal entry $[\mu] = [t_{(m+1)\ell}]$ on input $[t_{(m+1)1}], \ldots, [t_{(m+1)(m+n)}]$ and $\mathsf{g} = \mathsf{LTZ}$.

Protocol 5.1: $([m], [\mathbf{i}]) \leftarrow \mathsf{FindMin}(\mathbf{X}, \mathsf{g})$

Input: $[\mathbf{X}] \in \mathbb{Z}_{\langle k \rangle}^{p \times n}$, $\mathsf{g} \in \{\mathsf{LTZ}, \mathsf{FracLTZ}, \mathsf{BlandFracLTZ}\}$.
Output: $[m] \in \mathbb{Z}_q$, $[\mathbf{i}] \in \{0,1\}^n$

1 **if** $n = 1$ **then**
2 **return** $([\mathbf{x}_1], [1])$;
3 **foreach** $j \in \{1, \ldots, \lfloor n/2 \rfloor\}$ **do parallel**
4 $[t_j] \leftarrow \mathsf{g}([\mathbf{x}_{2j}], [\mathbf{x}_{2j-1}])$;
5 $[\mathbf{x}'_j] \leftarrow [\mathbf{x}_{2j-1}] + [t_j]([\mathbf{x}_{2j}] - [\mathbf{x}_{2j-1}])$; `// 1 rnd,` $p\lfloor n/2 \rfloor$ `inv`
6 **if** n is odd **then**
7 $[\mathbf{x}'_{(n+1)/2}] \leftarrow [\mathbf{x}_n]$;
8 $([m], [\mathbf{i}']) \leftarrow \mathsf{FindMin}(\mathbf{x}', \mathsf{g})$;
9 **foreach** $j \in \{1, \ldots, \lfloor n/2 \rfloor\}$ **do parallel**
10 $[s] \leftarrow [t_j][i'_j]$; `// 1 rnd,` $\lfloor n/2 \rfloor$ `inv`
11 $[i_{2j-1}] \leftarrow [s]$;
12 $[i_{2j}] \leftarrow [i'_j] - [s]$;
13 **if** n is odd **then**
14 $[i_n] \leftarrow [i'_{(n+1)/2}]$;
15 **return** $([m], [\mathbf{i}])$

According to Bland's rule ℓ is computed by

$$\ell = \min \left\{ i \in \{1, \ldots, n+m\} \,|\, t_{(m+1)i} < 0 \right\}.$$

Consider Protocol 5.2. By construction $\mathsf{FirstNeg}$ returns $[\mathbf{0}]$ if no $\ell \in \{1, \ldots, n+m\}$ exists such that $t_{(m+1)\ell} < 0$. Hence, without interaction, one can compute the bit $[d]$ by

$$[d] \leftarrow \sum_{i=1}^{n} [\ell_i].$$

Protocol 5.2: $[\boldsymbol{\ell}] \leftarrow \mathsf{FirstNeg}([\mathbf{t}])$

Input: $[\mathbf{t}] \in \mathbb{Z}_{\langle k \rangle}^{n}$
Output: $[\boldsymbol{\ell}] \in \{0,1\}^n$

1 **foreach** $i \in \{1, \ldots, n\}$ **do parallel**
2 $[z_i] \leftarrow [t_i] < 0$; `// 4 rnd,` $n(4k+2)$ `inv`
3 $[\mathbf{v}] \leftarrow \mathsf{PrefixOr}([\mathbf{z}])$; `// 3 rnd,` $5n-1$ `inv`
4 $[\ell_1] \leftarrow [v_1]$;
5 **foreach** $i \in \{2, \ldots, n\}$ **do**
6 $[\ell_i] \leftarrow [v_i] - [v_{i-1}]$;
7 **return** $([\boldsymbol{\ell}])$

In conclusion the pivot column is securely selected as follows:

1. Let $[\mathbf{t}] = ([t_{(m+1)1}], \ldots, [t_{(m+1)(n+m)}])$.

   Dantzig' Pivoting Rule: Compute $([min], [\boldsymbol{\ell}]) = \mathsf{FindMin}([\mathbf{t}], \mathsf{LTZ})$ and compute $[d] = [min < 0]$.

Bland's Pivoting Rule: Compute $[\boldsymbol{\ell}] = \mathsf{FirstNeg}([\mathbf{t}])$ and $[d] = \sum_{i=1}^{n+m}[\ell_i]$.

2. Reveal the bit d.
3. If $d = 1$, then return the pivot column $[\mathbf{p^c}] = [\mathbf{T}][\boldsymbol{\ell}]$ and the index $[\boldsymbol{\ell}]$. If $d = 0$, then output the current solution being the optimal.

5.1.1.2 Row Step

Given tableau $\mathbf{T}$ and the index of the pivot column ℓ, this step selects the pivot row $\mathbf{t}_k$, where $k \in \{1, \ldots, m\}$ for which $t_{k(n+m+1)}/t_{k\ell} = \min(\mathcal{W})$, where

$$\mathcal{W} = \left\{ \frac{t_{i(n+m+1)}}{t_{i\ell}} \middle| \, t_{i\ell} > 0, \text{ and } i \in \{1, \ldots, m\} \right\}. \tag{5.4}$$

If $t_{k\ell} \leq 0$ for all $k \in \{1, \ldots, m\}$, then the simplex algorithm stops and reports that the LP is unbounded.

In Chapter 3 we showed that k is the minimum of the list $\mathcal{W}^* = \{w_1^*, \ldots, w_m^*\}$, where

$$w_i^* = \begin{cases} \frac{t_{i(n+m+1)}}{t_{i\ell}}, & \text{if } t_{i\ell} > 0, \\ \infty, & \text{otherwise.} \end{cases}$$

Observe that $\min(\mathcal{W}^*) = \infty$ if and only if $t_{k\ell} \leq 0$ for all $k \in \{1, \ldots, m\}$ and that $\text{argmin}(\mathcal{W}^*) = \text{argmin}(\mathcal{W})$ otherwise.

We encounter two problems for a secure implementation: (i) $\infty \notin \mathbb{Z}_{\langle k \rangle}$ and (ii) $\frac{t_{i(n+1)}}{t_{i\ell}} \notin \mathbb{Z}_{\langle k \rangle}$.

In [Tof09] one replaces ∞ by $2^{k-1} - 1$. Furthermore, it is observed that if $c > 0$ and $d > 0$ then

$$\frac{a}{c} \leq \frac{b}{d} \Leftrightarrow ad \leq bc, \tag{5.5}$$

and, therefore, comparing the fractions in $\mathcal{W}$ (and $\mathcal{W}^*$, where $\infty = \frac{\infty}{1}$) can be replaced by comparing integers. Indeed, by construction, the entries in $\mathcal{W}$ and $\mathcal{W}^*$ have strictly positive denominators.

To securely compute the pivot row one proceeds as follows.

1. For $i = 1, \ldots, m$ do
   (a) $[\beta_i] \leftarrow [t_{i\ell} > 0]$.
   (b) $[w_{i1}] \leftarrow 2^{k-1} - 1 + [\beta_i]([t_{(n+m+1)i}] - 2^{k-1} - 1)$.
   (c) $[w_{i2}] \leftarrow 1 + [\beta_i]([t_{\ell i}] - 1)$.
   (d) $[w_i] \leftarrow (w_{i1}, w_{i2})$.
2. $([min], [\mathbf{k}]) \leftarrow \mathsf{FindMin}([\mathbf{W}], \mathsf{FracLTZ})$.
3. If $min = 2^{k-1} - 1$ stop and return unbounded LP.

If Bland's rule is applied, one should select k such that $k = \text{argmin}(\mathcal{W})$ and in addition $s_k < s_i$ for all $i = \text{argmin}(\mathcal{W})$, see Protocol 5.4. To select the pivot row using Bland's rule one proceeds as before, where $w_i = (w_{i1}, w_{i2}, w_{i3})$ and where w_{i1} and w_{i2} are computed as before and $w_{i3} = s_i$. Finally, $([m], [\mathbf{k}])$ is computed by $\mathsf{FindMin}([\mathbf{w}], \mathsf{BlandFracLTZ})$.

Protocol 5.3: $[b] \leftarrow \mathsf{FracLTZ}([\mathbf{a}], [\mathbf{b}])$

Input: $[\mathbf{a}] \in \mathbb{Z}^2_{\langle k \rangle}$, $[\mathbf{b}] \in \mathbb{Z}^2_{\langle k \rangle}$.
Output: $[b] \in \{0, 1\}$.

1 $[a'] \leftarrow [a_1][b_2]$; // 1 rnd, 1 inv
2 $[b]' \leftarrow [a_2][b_1]$; // 1 inv
3 $[\beta] \leftarrow [a'] \leq [b']$; // 4 rnd, $(4k+2)$ inv
4 **return** $[\beta]$;

Protocol 5.4: $[b] \leftarrow \mathsf{BlandFracLTZ}([\mathbf{a}], [\mathbf{b}])$

Input: $[\mathbf{a}] \in \mathbb{Z}^3_{\langle k \rangle}$, $[\mathbf{b}] \in \mathbb{Z}^3_{\langle k \rangle}$.
Output: $[b] \in \{0, 1\}$.

1 $[a'] \leftarrow [a_1][b_2]$; // 1 rnd, 1 inv.
2 $[b]' \leftarrow [a_2][b_1]$; // 1 inv.
3 $[\gamma] \leftarrow [a'] = [b']$; // $\max\{\log^*(k), 4\}$ rnd, $\log^*(k) \log k$ inv.
4 $[\beta] \leftarrow [a'] \leq [b']$; // $(4k+2)$ inv.
5 $[\alpha] \leftarrow [a_3] \leq [b_3]$; // $(4k+2)$ inv.
6 $[\delta] \leftarrow \beta + \gamma(\alpha - \beta)$; // 1 rnd, 1 inv.
7 **return** $[\delta]$;

We will present a more efficient solution by exploiting the nonnegativity constraints. The following solution saves m secure multiplications and one equality comparison. This is based on the following lemma.

Lemma 5.1. *Consider a tableau* $\mathbf{T}$. *Let* $\beta_i = |t_{i\ell} \leq 0|_b$ *and*

$$\mathcal{W}^+ = \left\{ \frac{t_{i(n+m+1)} + \beta_i}{t_{i\ell}} \middle| \, i \in \{1, \ldots, m\} \right\}. \tag{5.6}$$

Suppose $\frac{v_1}{v_2} \in \mathcal{W}^+$ *and* $\frac{w_1}{w_2} \in \mathcal{W}^+$ *and*

$$\frac{v_1}{v_2} \sqsubset \frac{w_1}{w_2} \Leftrightarrow v_1 w_2 \leq w_1 v_2.$$

Then, $\mathcal{W} \neq \emptyset$ *implies that* $\operatorname{argmin}(\mathcal{W}^+) = \operatorname{argmin}(\mathcal{W})$, *and* $\mathcal{W} = \emptyset$ *implies that* $d = \sum_{i=1}^{m}(1 - \beta_i) = 0$.

Proof. Consider integers a, b, c, d, where $a \geq 0$ and $b \geq 0$. If $c > 0$ and $d > 0$, then

$$ad \leq bc \Leftrightarrow \frac{a}{c} \leq \frac{b}{d},$$

but if $d \leq 0$ and $c > 0$ then $|ad \leq (b+1)c|_b = 1$. If $d > 0$, $c \leq 0$ and $a \neq 0$, then $|a(d+1) \leq bc|_b = 0$.

We show that these three rules imply that the ordering $\sqsubset$ on $\mathcal{W}^+$ yields the desired result.

Let $\mathbf{T}$ be a tableau with respect to some basis $\mathbf{s}$. From the nonnegativity constraints we have $\mathbf{x}_\mathbf{s} = \mathbf{T}_{n+m+1} \geq \mathbf{0}$. Hence $t_{i(n+m+1)} + \beta_i > 0$ if $t_{i\ell} \leq 0$.

The above three rules imply

$$\left|(t_{i(n+m+1)}+\beta_i)t_{j\ell} \leq (t_{j(n+m+1)}+\beta_j)t_{i\ell}\right|_b = \begin{cases} \left|\frac{t_{i(n+m+1)}}{t_{i\ell}} \leq \frac{t_{j(n+m+1)}}{t_{j\ell}}\right|_b, & \text{if } t_{i\ell}>0 \text{ and } t_{j\ell}>0, \\ 1, & \text{if } t_{i\ell}>0 \text{ and } t_{j\ell}\leq 0, \\ 0, & \text{if } t_{i\ell}\leq 0 \text{ and } t_{j\ell}>0, \\ \gamma, & \text{otherwise,} \end{cases}$$

for some $\gamma \in \{0,1\}$, which is not relevant in the following.

It follows that $\sqsubset$ orders $\mathcal{W}^+$ in such a way that all entries with a positive denominator are considered smaller than entries with nonpositive denominators and, moreover, the entries with a positive denominator are ordered normally, i.e., according to $<$.

Hence if $\mathcal{W} \neq \emptyset$, then $\operatorname{argmin}(\mathcal{W}^+) = \operatorname{argmin}(\mathcal{W})$, but if $\mathcal{W} = \emptyset$ then $d = \sum_{i=1}^{m}(1-\beta_i) = 0$. □

In conclusion, the pivot row is securely selected as follows given pivot column $[\mathbf{p}^c]$ and pivot column index $[\boldsymbol{\ell}]$:

1. Compute the bits $[\beta_i] = [p_i^c \leq 0]$ for all $i \in \{1,\ldots,m\}$.
2. Reveal the bit $d = \sum_{i=1}^{m}(1-[\beta_i])$. If $d = 0$ stop and report that the LP is unbounded.

   Dantzig' Pivoting Rule: (a) Compute the list $[\mathbf{W}]$ where

   $$([t_{i(n+m+1)}] + [\beta_i], [p_i^c]) = [\mathbf{w}_i]$$

   for all $i = 1,\ldots,m$.

   (b) Compute $([min], [\mathbf{k}]) = \mathsf{FindMin}([\mathbf{W}], \mathsf{FracLTZ})$.

   Bland's Pivoting Rule: (a) Compute the list $[\mathbf{W}]$ where

   $$([t_{i(n+m+1)}] + [\beta_i], [p_i^c], [s_i]) = [\mathbf{w}_i],$$

   for all $i = 1,\ldots,m$.

   (b) Compute $([min], [\mathbf{k}]) = \mathsf{FindMin}([\mathbf{W}], \mathsf{BlandFracLTZ})$.
3. Compute the pivot row $[\mathbf{p}^r] \leftarrow [\mathbf{k}][\mathbf{T}]$.

5.1.1.3 Update Tableau and Basis

This step computes the new tableau $\mathbf{T}'$ and basis $\mathbf{s}'$, given tableau $\mathbf{T}$, basis $\mathbf{s}$, pivot column $\mathbf{p}^c$ with index ℓ, pivot row $\mathbf{p}^r$ with index k.

In Chapter 3 we showed that $\mathbf{T}$ is updated as follows

$$\begin{aligned} t'_{ij} &= \frac{t_{ij}t_{k\ell} - t_{i\ell}t_{kj}}{q_1}, \quad \text{if } i \neq k, \\ t'_{kj} &= \frac{t_{kj}}{q_2}, \end{aligned} \tag{5.7}$$

where (q_1, q_2) is equal to either $(t_{k\ell}, t_{k\ell})$ if rational pivoting is applied, or to $(q, 1)$ if integer pivoting is applied.

Note that the computation of t'_{ij} depends on the value of i. To hide the value for i we introduce two vectors $\mathbf{v} \in \mathbb{Z}_{\langle k\rangle}^{m+1}$ and $\mathbf{w} \in \mathbb{Z}_{\langle k\rangle}^{m+1}$ so that Eq. (5.7) is equivalent to

$$t'_{ij} = t_{ij}v_i - p_j^r w_i,$$

which is independent to i. For example, $\mathbf{v}$ defined by $v_i = \frac{t_{k\ell}}{q_1}$ for all $i \neq k$ and $v_k = \frac{1}{q_2}$ and $\mathbf{w}$ defined by $w_i = \frac{t_{i\ell}}{q_1}$ for all $i \neq k$ and $w_k = 0$ suffices.

Let the pivot column $[\mathbf{p}^c] = [\mathbf{T}_\ell]$, the pivot row $[\mathbf{p}^r] = [\mathbf{t}_k]$, and the pivot element $[t_{k\ell}]$ be given in addition to $[q_1^{-1}]$ and $[q_2^{-1}]$. Then one securely computes

$$[\mathbf{v}] \leftarrow \mathsf{WriteAtPosition}(\mathbf{1}[t_{k\ell}][q_1^{-1}], [\mathbf{k}], [q_2^{-1}])$$

using $m + 2$ secure multiplications and

$$[\mathbf{w}] \leftarrow \mathsf{WriteAtPosition}([q_1^{-1}][\mathbf{p}^c], [\mathbf{k}], 0)$$

using $2m+2$ secure multiplications. Observe that $[\mathbf{v}]$ and $[\mathbf{w}]$ are computed in 2 interactive rounds. Note that for the RP variants $[t_{k\ell}][q_1^{-1}] = 1$ is public knowledge requiring no secure multiplication at all.

A more efficient choice for $\mathbf{v}$ and $\mathbf{w}$ is given by $\mathbf{v} = \frac{t_{k\ell}}{q_1}\mathbf{1}$ and $\mathbf{w} = \frac{1}{q_1}\mathbf{p}^c - \frac{\mathbf{k}}{q_2}$. Indeed,

$$[\mathbf{v}] = [t_{k\ell}][q_1^{-1}]\mathbf{1}$$

can securely be computed using 1 secure multiplication, and

$$[\mathbf{w}] = [q_1^{-1}][\mathbf{p}^c] - [q_2^{-1}][\mathbf{k}]$$

using $m + 1$ secure multiplications using the fact that $q_2 = 1$ or $q_2 = q_1$.

Lemma 5.2 shows that this choice for $\mathbf{v}$ and $\mathbf{w}$ is also correct.

Lemma 5.2. *Let row k denote the pivot row. For all i, j the entries of $\mathbf{T}$ satisfy*

$$t'_{ij} = t_{ij}v - w_i p^r_j,$$

where $v = \frac{t_{k\ell}}{q_1}$ and $\mathbf{w} = \frac{1}{q_1}\mathbf{p}^c - \frac{\mathbf{k}}{q_2}$, satisfies Eq. (5.7).

Proof. Indeed, let $\delta_{ij} = |i = j|_b$ denote Kronecker's delta. We have

$$\begin{aligned} t'_{ij} &= t_{ij}v - w_i p^r_j \\ &= t_{ij}\frac{t_{k\ell}}{q_1} - t_{kj}\left(\frac{p^c_i}{q_1} - \frac{\delta_{ik}}{q_2}\right) \\ &= \frac{t_{ij}t_{k\ell} - t_{i\ell}t_{kj}}{q_1} + \delta_{ik}\frac{t_{kj}}{q_2} \\ &= (1 - \delta_{ik})\frac{t_{ij}t_{k\ell} - t_{i\ell}t_{kj}}{q_1} + \delta_{ik}\frac{t_{kj}}{q_2}, \end{aligned}$$

where the last equality holds since $t_{ij}t_{k\ell} - t_{i\ell}t_{kj} = 0$ if $i = k$. □

In conclusion, the tableau and basis are securely updated as follows, given $[\mathbf{T}]$, $[\mathbf{s}]$, $[\boldsymbol{\ell}]$, $[\mathbf{k}]$, $[\mathbf{p}^c]$, $[\mathbf{p}^r]$, and $[q]$ if the IP variant is considered,

1. With respect to rational pivoting let $v \leftarrow 1$ and compute $[\mathbf{w}] \leftarrow [t_{k\ell}]^{-1}([\mathbf{p}^c] - [\mathbf{k}])$. With respect to integer pivoting, compute $[v] = [q^{-1}][t_{k\ell}]$ and $[\mathbf{w}] \leftarrow [q^{-1}][\mathbf{p}^c] - [\mathbf{k}]$.
2. For all i, j compute $[t'_{ij}] \leftarrow [t_{ij}][v] - [w_i][p^r_j]$.
3. Update the basis by $[\mathbf{s}]' \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], \sum_{i=1}^{n+m}[\ell_i]i)$.

5.1.2 Small Tableau Simplex

Let $\mathbf{T}$ be a tableau corresponding to basis $\mathbf{s}$ and co-basis $\mathbf{u}$. In this section, we write $\mathbf{T}$ for the corresponding condensed tableau ($\mathbf{T}_{\mathbf{u},n+m+1}$) as introduced in Definition 3.38 in Section 3.2.2.

In Section 3.2 we showed that the small tableau simplex algorithm consists of the following three steps in each iteration:

1. *Column step:* Find an $\ell \in \{1, \ldots, n\}$ such that $t_{(m+1)\ell} < 0$. If no such ℓ exists, then output the current solution being the optimum.

2. *Row step:* Find a $k \in \{1, \ldots, m\}$ such that

$$k = \text{argmin} \left\{ \frac{t_{i(n+m+1)}}{t_{i,\ell}} \middle| t_{i\ell} > 0, \text{ and } i \in \{1, \ldots, m\} \right\}.$$

   If $t_{k\ell} \leq 0$ for all $k \in \{1, \ldots, m\}$, then stop and report that the LP is unbounded.

3. *Update tableau and basis:* In case of rational pivoting, pivot on $t_{k\ell}$ by updating $\mathbf{T}$ according to:

$$t_{ij} = \begin{cases} t_{ij} - t_{i\ell} t_{kj} / t_{k\ell}, & \text{if } i \neq k, \\ t_{ij} / t_{k\ell} & \text{if } i = k. \end{cases}$$

   Replace the column corresponding to the variable entering the basis (ℓ) with the column corresponding the variable leaving the basis (s_k) by computing for all i

$$t_{i\ell} = \begin{cases} -\frac{t_{i\ell}}{t_{k\ell}}, & \text{if } i \neq k, \\ \frac{1}{t_{k\ell}}, & \text{otherwise.} \end{cases}$$

   In case of integer pivoting, pivot on $t_{k\ell}$ by updating $\mathbf{T}$ according to:

$$t_{ij} = \begin{cases} \left(t_{ij} t_{k\ell} - t_{i\ell} t_{kj}\right) / q, & \text{if } i \neq k, \\ t_{ij}, & \text{if } i = k. \end{cases}$$

   Replace the column corresponding to the variable that enters the basis (ℓ) with the column corresponding the variable leaving the basis (s_k) by computing for all i

$$t_{i\ell} = \begin{cases} -t_{i\ell}, & \text{if } i \neq k, \\ q, & \text{otherwise.} \end{cases}$$

   The basis is updated by swapping s_k with u_ℓ.

The detailed protocols can be found in Appendix A.1.2.

5.1.2.1 Column Step

The column step for the small tableau simplex algorithm with Dantzig's original pivoting rule is exactly the same as for the large tableau simplex algorithm. Indeed, Dantzig's original pivoting rule selects an ℓ based solely on the value of $t_{(m+1)\ell}$. Bland's pivoting rule, on the other hand, selects an ℓ for which the corresponding column index (u_ℓ) is the smallest value where $t_{(m+1)\ell} < 0$. Hence for Dantzig's original pivoting rule we can reuse

the protocols for large tableau simplex when selecting the pivot column, but for Bland's pivoting rule we need to be careful.

According to Bland's rule one has to compute ℓ in the condensed tableau by

$$\ell = \text{argmin}\left\{ u_i \in \{1,\ldots,n+m\} \,|\, t_{(m+1)i} < 0 \right\}, \tag{5.8}$$

which cannot be computed securely via FirstNeg (Protocol 5.2), since the entries in $\mathbf{u}$ are not ordered normally.

Observe that Eq. (5.8) is very similar to $k = \text{argmin}(\mathcal{W})$: the equation to select the pivot row, where $\mathcal{W}$ is defined by Eq. (5.4). A secure implementation of the selection of ℓ according to Bland's rule is as follows (see also Protocol 5.5).

1. For all $i = 1,\ldots,n$ do
   (a) compute the bits $[\beta_i] \leftarrow [t_{(m+1)i} < 0]$,
   (b) compute $[v_i] = [u_i] + (1 - [\beta_i])(n+m+1)$.
2. $([min], \boldsymbol{\ell}) \leftarrow \mathsf{FindMin}([\mathbf{v}], \mathsf{LTZ})$.
3. If $\sum_{i=1}^{n} [\beta_i] = 0$ then stop and return the optimal solution.

Since $\mathbf{u} \in \{1,\ldots,n+m\}^n$ is a co-basis the minimum of $\mathbf{u}$ is unique and if some $t_{(m+1)i} < 0$ then argmin($\mathbf{u}$) is equal to ℓ computed by Eq. (5.8). Otherwise, if no $t_{(m+1)i} < 0$ then $\sum_{i=1}^{n} \beta_i = 0$ and the simplex algorithm stops reporting that the current solution is optimal.

Protocol 5.5: $[\boldsymbol{\ell}] \leftarrow \mathsf{FirstNeg}_{\mathsf{ST}}([\mathbf{t}], [\mathbf{u}])$

Input: $[\mathbf{t}] \in \mathbb{Z}^n_{\langle k \rangle}$
Output: $[\boldsymbol{\ell}] \in \{0,1\}^n$

foreach $i \in \{1,\ldots,n\}$ **do parallel**
  $[\beta_i] \leftarrow [t_i] \geq 0$; // 4 rnd, $n(4k+2)$ inv
  $[v_i] \leftarrow [u_i] + [\beta_i](n+m+1)$;
$([min], [\boldsymbol{\ell}]) \leftarrow \mathsf{FindMin}([\mathbf{v}], \mathsf{LTZ})$; // $\lceil \log n \rceil (4+2)$ rnd, $(n-1)((4k+2)+2)$ inv.
$d \leftarrow \sum_{i=1}^{n} [\beta_i]$;
if $d = 1$ **then**
  $\boldsymbol{\ell} = \mathbf{0}$;
return $([\boldsymbol{\ell}])$

In conclusion, the pivot column is securely selected for the small tableau simplex algorithm as follows:

1. Let $[\mathbf{t}] = ([t_{(m+1)1}], \ldots, [t_{(m+1)n+m}])$.

   Dantzig's Pivoting Rule: (a) Compute $[\boldsymbol{\ell}]$ by $\mathsf{FindMin}([\mathbf{t}], \mathsf{LTZ})$.
   (b) Compute and reveal the bit $[d] \leftarrow [t_{(m+1)\ell} < 0]$.

   Bland's Pivoting Rule: (a) Compute $[\boldsymbol{\ell}]$ by $\mathsf{FirstNeg}_{\mathsf{ST}}([\mathbf{t}], [\mathbf{u}])$.
   (b) Compute and reveal the bit $d \leftarrow \sum_{i=1}^{n} [\ell_i]$.

2. If $d = 1$, then return the pivot column $[\mathbf{p^c}] = [\mathbf{T}][\boldsymbol{\ell}]$ and index $[\boldsymbol{\ell}]$. If $d = 0$, then output the current solution being the optimal.

5.1.2.2 Row Step

The row step for small tableau simplex is completely the same as the row step for large tableau simplex since all operations of this step are independent to the co-basis. See also Section 3.2.2.

5.1.2.3 Update Tableau and Basis

This step computes the new (condensed) tableau $\mathbf{T}'$ and basis $\mathbf{s}'$, given (condensed) tableau $\mathbf{T}$, basis $\mathbf{s}$, co-basis $\mathbf{u}$, pivot column index ℓ, and pivot row index k.

In Section 3.2.2 we showed that small tableau simplex performs the following three operations when updating the tableau and basis, when column ℓ is selected as pivot column and k is selected as pivot row: (i) pivot on element $t_{k\ell}$ in $\mathbf{T}$, (ii) replace column ℓ by the result of updating column $\mathbf{e}_k$ after pivoting, and (iii) swap s_k with u_ℓ.

Similarly to the large tableau simplex we show how to simultaneously and obliviously perform both operation (i) and operation (ii). Depending on the fact whether rational pivoting or integer pivoting is applied, the two vectors $\mathbf{v}$ and $\mathbf{w}$ are given. In addition, a new vector $\mathbf{r}$ is introduced where each entry of the (condensed) tableau $\mathbf{T}$ being updated by

$$t'_{ij} = t_{ij}v - r_j w_i.$$

Then $\mathbf{T}'$ is the new condensed tableau corresponding basis $\mathbf{s}'$ and co-basis $\mathbf{u}'$.

Observe that by Lemma 3.37 the updated condensed tableau is computed according to

$$\begin{aligned} t'_{ij} &= \tfrac{t_{ij}t_{k\ell} - t_{i\ell}t_{kj}}{q_1}, && \text{if } i \neq k \text{ and } j \neq \ell, \\ t'_{kj} &= \tfrac{t_{kj}}{q_2}, && \text{if } j \neq \ell, \\ t'_{i\ell} &= -\tfrac{t_{i\ell}}{q_2}, && \text{if } i \neq k, \\ t'_{k\ell} &= \tfrac{q_1}{q_2}, \end{aligned} \tag{5.9}$$

where (q_1, q_2) is equal to either $(t_{k\ell}, t_{k\ell})$ if rational pivoting is applied, or to $(q, 1)$ if integer pivoting is applied.

Lemma 5.3 shows that replacing $\mathbf{p}^r$ with $\mathbf{r} = \mathbf{p}^r + \frac{q_1}{q_2}\boldsymbol{\ell}$ in Lemma 5.2 yields the updated condensed tableau. So the column replacement is for free when rational pivoting is applied and requires $n + 1$ secure multiplications if integer pivoting is applied.

Lemma 5.3. *If row k is the pivot row and column ℓ the pivot column, then*

$$t'_{ij} = t_{ij}v - w_i r_j$$

satisfies Eq. (5.9) *for all i, j, where $v = \frac{t_{k\ell}}{q_1}$, $\mathbf{w} = \frac{1}{q_1}\mathbf{p}^c - \frac{\mathbf{k}}{q_2}$ and $\mathbf{r} = \mathbf{p}^r + \frac{q_1}{q_2}\boldsymbol{\ell}$.*

Proof. Observe that $r_j = p^r_j$ for all $j \neq \ell$ and that, by Lemma 5.2,

$$t'_{ij} = t_{ij}v - w_i r_j$$

satisfies Eq. (5.9) for all $i = 1, \ldots, m + 1$, $j = 1, \ldots, n + 1$, where $j \neq \ell$.

Let $\delta_{ij} = |i = j|_b$ denote Kronecker's delta. If $j = \ell$, then

$$\begin{aligned} t'_{i\ell} &= t_{i\ell}v - w_i r_\ell \\ &= t_{i\ell}v - w_i(t_{k\ell} + q_1/q_2) \\ &= t_{i\ell}\frac{t_{k\ell}}{q_1} - (t_{k\ell} + q_1/q_2)\left(\frac{p_i^c}{q_1} - \frac{\delta_{ik}}{q_2}\right) \\ &= \frac{t_{i\ell}t_{k\ell} - t_{i\ell}t_{k\ell}}{q_1} - \frac{p_i^c}{q_2} + \delta_{ik}\left(\frac{t_{k\ell}}{q_2} + \frac{q_1}{q_2^2}\right) \\ &= -\frac{t_{i\ell}}{q_2} + \delta_{ik}\left(\frac{t_{k\ell}}{q_2} + \frac{q_1}{q_2^2}\right) \\ &= (1 - \delta_{ik})(-\frac{t_{i\ell}}{q_2}) + \delta_{ik}\frac{q_1}{q_2^2}. \end{aligned}$$

□

In conclusion, in small tableau simplex the tableau and basis are securely updated as follows, given $[\mathbf{T}]$, $[\mathbf{s}]$, $[\boldsymbol{\ell}]$, $[\mathbf{k}]$, $[\mathbf{p}^c]$, $[\mathbf{p}^r]$, and $[q]$:

1. With respect to the RP variant set $v \leftarrow 1$ and compute $[\mathbf{w}] \leftarrow [t_{k\ell}]^{-1}([\mathbf{p}^c] - [\mathbf{k}])$ and $\mathbf{r} = \mathbf{p}^r + \boldsymbol{\ell}$. With respect to the IP variant, compute $[v] = [q^{-1}][t_{k\ell}]$, $[\mathbf{w}] \leftarrow [q^{-1}][\mathbf{p}^c] - [\mathbf{k}]$ and $[\mathbf{r}] = [\mathbf{p}^r] + [q][\boldsymbol{\ell}]$.
2. For all i, j compute $[t'_{ij}] \leftarrow [t_{ij}][v] - [w_i][r_j]$.
3. Update the basis by $[\mathbf{s}]' \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\mathbf{u}][\boldsymbol{\ell}])$ and the co-basis by $[\mathbf{u}]' \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}], [\boldsymbol{\ell}], [\mathbf{s}][\mathbf{k}])$.

5.1.3 Revised Simplex

Let

$$\mathbf{T}^0 = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{c} & \mathbf{0} & 0 \end{pmatrix}$$

and

$$\mathbf{D} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix}.$$

Then $\mathbf{T} = \mathbf{D}\mathbf{T}^0$ is a tableau with respect to basis $\mathbf{s}$ and basis matrix $\mathbf{B} = \mathbf{A_s}$.

We showed in Chapter 3 that the revised simplex algorithm consists of the following three steps in each iteration given $\mathbf{D}$ and basis $\mathbf{s}$:

1. *Column step:* Compute the last row of $\mathbf{T}$ by $\mathbf{t}_{m+1} = \mathbf{d}_{m+1}\mathbf{T}$. Find an $\ell \in \{1, \ldots, n+m\}$ such that $t_{(m+1)\ell} < 0$. If no such ℓ exists, then output the current solution being the optimum.
2. *Row step:* Compute the pivot column by $\mathbf{T}_\ell = \mathbf{D}\mathbf{T}^0_\ell$ and the last column of $\mathbf{T}$ by $\mathbf{T}_{m+n+1} = \mathbf{D}\mathbf{T}^0_{m+n+1}$. Find a $k \in \{1, \ldots, m\}$ such that

$$k = \operatorname{argmin}\left\{ \frac{t_{i(n+m+1)}}{t_{i,\ell}} \middle| t_{i\ell} > 0, \text{ and } i \in \{1, \ldots, m\} \right\}.$$

   If $t_{k\ell} \leq 0$ for all $k \in \{1, \ldots, m\}$, then stop and report that the LP is unbounded.

3. *Update tableau and basis:* In case of rational pivoting, pivot on $t_{k\ell}$ by updating $\mathbf{D}$ by Eqs. (3.17) and (3.18):

$$d_{ij} = \begin{cases} d_{ij} - t_{i\ell} d_{kj} / t_{k\ell}, & \text{if } i \neq k, \\ d_{ij} / t_{k\ell}, & \text{if } i = k. \end{cases}$$

Otherwise, in case of integer pivoting, pivot on $t_{k\ell}$ by updating $\mathbf{D}$ by Eqs. (3.30) and (3.31):

$$d_{ij} = \begin{cases} \left(d_{ij} t_{k\ell} - t_{i\ell} d_{kj}\right) / q, & \text{if } i \neq k, \\ d_{ij}, & \text{if } i = k. \end{cases}$$

The basis is updated by replacing s_k with ℓ.

The detailed protocols can be found in Appendix A.1.3.

5.1.3.1 Column Step

Given tableau $\mathbf{T}$ this step selects an $\ell \in \{1, \ldots, n+m\}$ where $t_{(m+1)\ell} < 0$. If no such ℓ exists, then the simplex algorithm stops reporting that the current solution corresponding to $\mathbf{T}$ is optimal.

The only difference between the revised simplex and the large tableau simplex is that the last row of the tableau needs to be computed.

In conclusion in revised simplex the pivot column is securely selected as follows:

1. Compute $[\mathbf{t}] \leftarrow ([\mathbf{d}_{m+1}][\mathbf{T}^0_1], \ldots, [\mathbf{d}_{m+1}][\mathbf{T}^0_{m+n}])$.

   Dantzig' Pivoting Rule: Compute $([min], [\boldsymbol{\ell}]) = \mathsf{FindMin}([\mathbf{t}], \mathsf{LTZ})$ and compute $[d] = [min < 0]$.

   Bland's Pivoting Rule: Compute $[\boldsymbol{\ell}] = \mathsf{FirstNeg}([\mathbf{t}])$ and $[d] = \sum_{i=1}^{n+m} [\ell_i]$.

2. Reveal the bit d.

3. If $d = 1$, then return the pivot column $[\mathbf{p^c}] = [\mathbf{D}][\mathbf{T}^0][\boldsymbol{\ell}]$ and the index $[\boldsymbol{\ell}]$. If $d = 0$, then output the current solution being the optimal.

5.1.3.2 Row Step

Like the column step, the row step is the same as that of the large tableau simplex, except for the fact that the pivot column and the last column of the tableau $\mathbf{T}$ need to be computed.

In revised simplex the pivot row is securely selected as follows given the pivot column $[\mathbf{p}^c]$ and pivot column index ℓ,

1. Compute the bits $[\beta_i] = [p^c_i \leq 0]$ for all $i \in \{1, \ldots, m\}$.

2. Reveal the bit $d = \sum_{i=1}^{m} (1 - [\beta_i])$. If $d = 0$ we stop and report that the LP is unbounded.

3. Compute the last column of $\mathbf{T}$, i.e., $\mathbf{t} = \mathbf{D}\mathbf{T}^0_{n+m+1}$.

Dantzig' Pivoting Rule: (a) Compute the list $[\mathbf{W}]$, where

$$([t_i] + [\beta_i], [p_i^c]) = [\mathbf{w}_i]$$

for all $i = 1, \ldots, m$.

(b) Compute $([min], [\mathbf{k}]) = \mathsf{FindMin}([\mathbf{W}], \mathsf{FracLTZ})$.

Bland's Pivoting Rule: (a) Compute the list $[\mathbf{W}]$ where

$$([t_i] + [\beta_i], [p_i^c], [s_i]) = [\mathbf{w}_i],$$

for all $i = 1, \ldots, m$.

(b) Compute $([min], [\mathbf{k}]) = \mathsf{FindMin}([\mathbf{W}], \mathsf{BlandFracLTZ})$.

4. Compute the pivot row $[\mathbf{p}^r] \leftarrow [\mathbf{k}][\mathbf{D}]$.

5.1.3.3 Update Tableau and Basis

This step computes $\mathbf{D}'$ and basis $\mathbf{s}'$, given $\mathbf{D}$, basis $\mathbf{s}$, pivot column index ℓ, pivot row index k.

Observe that by Lemma 3.39 the updated $\mathbf{D}$ is computed using the same row operations as the pivot operation applied in large tableau simplex, i.e., one computes $\mathbf{D}'$ by

$$\begin{aligned} d'_{ij} &= \tfrac{d_{ij}t_{k\ell} - t_{i\ell}d_{kj}}{q_1} \quad \text{if } i \neq k \\ d'_{kj} &= \tfrac{d_{kj}}{q_2} \end{aligned} \tag{5.10}$$

where (q_1, q_2) is equal to either $(t_{k\ell}, t_{k\ell})$ if rational pivoting is applied, or to $(q, 1)$ if integer pivoting is applied.

We apply $\mathbf{v}$ and $\mathbf{w}$ defined in Section 5.1.1 to compute Eq. (5.10) as follows.

Lemma 5.4. *If row k is the pivot row, then*

$$d'_{ij} = d_{ij}v - w_i p_j^r,$$

where $v = \frac{t_{k\ell}}{q_1}$ and $\mathbf{w} = \frac{1}{q_1}\mathbf{p}^c - \frac{\mathbf{k}}{q_2}$, satisfies Eq. (5.10) for all i, j.

Proof. Indeed, let $\delta_{ij} = |i = j|_b$ denote Kronecker's delta. Then

$$\begin{aligned} d'_{ij} &= d_{ij}v - w_i p_j^r \\ &= d_{ij}\frac{t_{k\ell}}{q_1} - d_{kj}\left(\frac{p_i^c}{q_1} - \frac{\delta_{ik}}{q_2}\right) \\ &= \frac{d_{ij}t_{k\ell} - t_{i\ell}d_{kj}}{q_1} + \delta_{ik}\frac{d_{kj}}{q_2} \\ &= (1 - \delta_{ik})\frac{d_{ij}t_{k\ell} - t_{i\ell}d_{kj}}{q_1} + \delta_{ik}\frac{d_{kj}}{q_2}. \end{aligned}$$

□

In conclusion, the tableau and basis are securely updated as follows, given $[\mathbf{D}]$, $[\mathbf{s}]$, $[\boldsymbol{\ell}]$, $[\mathbf{k}]$, $[\mathbf{p}^c] = [\mathbf{T}_\ell]$, $[\mathbf{p}^r] = [\mathbf{d}_k]$, and $[q]$:

1. With respect to the RP variant let $v \leftarrow 1$ and compute $[\mathbf{w}] \leftarrow [t_{k\ell}]^{-1}([\mathbf{p}^c] - [\mathbf{k}])$. With respect to the IP variant, compute $[v] = [q^{-1}][t_{k\ell}]$ and $[\mathbf{w}] \leftarrow [q^{-1}][\mathbf{p}^c] - [\mathbf{k}]$.
2. For all i, j compute $[d'_{ij}] \leftarrow [d_{ij}][v] - [w_i][p_j^r]$.
3. Update the basis by $[\mathbf{s}]' \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], \sum_{i=1}^{n+m}[\ell_i]i)$.

5.2 Secure Simplex Initialization

This section presents how to securely and efficiently initialize the simplex algorithm via both the two-phase simplex algorithm and the big-M method.

Recall that in this chapter, we consider any LP in standard form:

$$\begin{array}{lrcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \mathbf{Ax} & \leq & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}. \end{array}$$

Observe that if $\mathbf{b} \geq \mathbf{0}$, then $\mathbf{x} = \mathbf{0}$ is feasible. Hence, if $\mathbf{x} \geq \mathbf{0}$ we can initialize simplex directly with tableau

$$\mathbf{T} = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{c} & \mathbf{0} & 0 \end{pmatrix}$$

and basis $\mathbf{s} = (n+1, \ldots, n+m)$ after introducing m slack variables to transform the linear programming in the standard form of Chapter 3.

If $\mathbf{b} \not\geq \mathbf{0}$, then $\mathbf{x} = \mathbf{0}$ is not a feasible solution and we apply the techniques discussed in Section 3.3 to solve the linear program by using artificial variables. For efficiency reasons one could try to avoid applying either the two-phase simplex or the big-M method by computing and revealing the bit $|\mathbf{b} \geq \mathbf{0}|_b$. If it is equal to one, then one initializes the simplex iterations with basis $\mathbf{s} = (n+1, \ldots, n+m)$.

In this section we assume that the bit $\mathbf{b} \geq \mathbf{0}$ is unknown and kept secret.

5.2.1 Standard two-phase Simplex

Recall that the standard two-phase simplex solves the corresponding artificial linear program:

$$\begin{array}{lrcl} \min & \sum_{i=1}^{m} x_{n+m+i}, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}. \end{array} \tag{5.11}$$

In slightly more detail, recall that the two-phase simplex consists of the following steps:

phase I

1. *Initialize phase I:* Given a linear program in standard form and its corresponding artificial linear program, compute a tableau $[\mathbf{T}]$ corresponding to a basis $[\mathbf{s}]$ with respect to a feasible solution $[\mathbf{x}]$ to the artificial linear program.
2. Iterate on $[\mathbf{T}]$ and $[\mathbf{s}]$ using one of the simplex variants until an optimal solution is found. If the optimum has zero costs, then continue, else stop and report that the original linear program is infeasible.

phase II

1. *Initialize phase II:* Given the tableau $[\mathbf{T}]$ and basis $[\mathbf{s}]$ from the result of phase one, compute a tableau $[\mathbf{T}']$ and basis $[\mathbf{s}']$ so that $[\mathbf{T}']$ is a tableau corresponding to basis $[\mathbf{s}']$ with respect to a feasible solution to the original linear program.
2. Iterate on $[\mathbf{T}']$ and $[\mathbf{s}']$ until either a solution is found, or the simplex algorithm reports that the linear program is unbounded.

In the remainder of this section we will show to securely initialize both phase I and phase II. The detailed protocols are presented in Appendix A.2.1.

5.2.1.1 Initialize Phase I

We will show how to efficiently compute a tableau initializing simplex for solving the artificial LP. Secondly, we apply the result of Theorem 3.21 implying that we don't need to add the m columns with respect to the artificial variables to the tableau $\mathbf{T}$.

Consider again the artificial linear program. It has the obvious feasible solution $\mathbf{x_u} = \mathbf{0}$ and $\mathbf{x_s} = \mathbf{b}$ if $\mathbf{b} \geq \mathbf{0}$, where $\mathbf{s} = (n+m+1, \ldots, n+2m)$ and $\mathbf{u} = (1, \ldots, n+m)$.

In general, however, $\mathbf{b}$ might have negative entries. We will show how to securely transform the linear program in standard from into an equivalent linear program that has as corresponding artificial LP the following:

$$\begin{array}{lrcl} \min & \sum_{i=1}^{m} x_{n+m+i}, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A'} & \mathbf{I}_m \end{pmatrix} \mathbf{x} & = & \mathbf{b'}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{5.12}$$

where $\mathbf{x_s} = \mathbf{b'} \geq \mathbf{0}$ is a basic feasible solution corresponding to basis $\mathbf{s} = (n+m+1, \ldots, n+2m)$.

Let $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m)$ and $1 - 2\,|b_i < 0|_b$. It follows that $\mathbf{b'} = \mathrm{diag}(\boldsymbol{\beta})\mathbf{b} \geq \mathbf{0}$, where $\mathrm{diag}(\cdot)$ denotes the diagonal matrix with $\cdot$ on its diagonal.

The linear program

$$\begin{array}{lrcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \mathbf{Ax} & \leq & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array}$$

is equivalent to

$$\begin{array}{lrcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix} \mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array}$$

which is equivalent to

$$\begin{array}{lrcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \mathbf{A'x} & = & \mathbf{b'}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array}$$

where $\mathbf{b'} = \mathrm{diag}(\boldsymbol{\beta})\mathbf{b} \geq 0$ and $\mathbf{A'} = \mathrm{diag}(\boldsymbol{\beta}) \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}$. Hence LP (5.12) is a corresponding artificial LP.

Finally, observe that the last row of $\mathbf{T}$ corresponding to basis

$$\mathbf{s} = (n+m+1, \ldots, n+2m)$$

is equal to

$$\mathbf{c} - \mathbf{c_s} \begin{pmatrix} \mathbf{A'} & \mathbf{b'} \end{pmatrix} = -\mathbf{1} \begin{pmatrix} \mathbf{A'} & \mathbf{b'} \end{pmatrix}$$

after deletion of the columns $n+m+1, \ldots, n+2m$, i.e., the columns with respect to the artificial variables.

In conclusion, given an LP in standard form, the following steps are performed to initialize phase I:

1. Compute $[\beta_i] \leftarrow 1 - 2[b_i < 0]$ for $i = 1, \ldots, m$.
2. Compute $[\mathbf{A}'] = \mathrm{diag}([\boldsymbol{\beta}]) \begin{pmatrix} [\mathbf{A}] & \mathbf{I}_m \end{pmatrix}$ and $[\mathbf{b}'] \leftarrow \mathrm{diag}([\boldsymbol{\beta}])[\mathbf{b}]$.
3. Set $[\mathbf{s}] \leftarrow (n + m + 1, \ldots, n + 2m)$ and

   Large Tableau Simplex: compute the entries of the last row of the tableau by $[\mathbf{c}'] \leftarrow -\sum_{i=1}^{m} [\mathbf{a}'_i]$ and $[z] \leftarrow \sum_{i=1}^{m} [b'_i]$. Set

   $$[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}'] & [\mathbf{b}'] \\ [\mathbf{c}'] & -[z] \end{pmatrix}.$$

   Small Tableau Simplex: set $[\mathbf{u}] \leftarrow (1, \ldots, n + m)$. Compute the entries of the last row of the tableau by $[\mathbf{c}'] \leftarrow -\sum_{i=1}^{m} [\mathbf{a}'_i]$ and $[z] \leftarrow \sum_{i=1}^{m} [b'_i]$. Set

   $$[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}'] & [\mathbf{b}'] \\ [\mathbf{c}'] & -[z] \end{pmatrix}.$$

   Revised Simplex: set

   $$[\mathbf{T^0}] \leftarrow \begin{pmatrix} [\mathbf{A}] & \mathbf{I}_m & [\mathbf{b}] \\ [\mathbf{0}] & \mathbf{0} & 0 \end{pmatrix}$$

   and

   $$[\mathbf{D}] \leftarrow \begin{pmatrix} [\mathrm{diag}(\boldsymbol{\beta})] & [\mathbf{0}] \\ -[\boldsymbol{\beta}] & 1 \end{pmatrix}.$$

Remark 5.5 (Small Tableau Simplex). Observe that the tableaus with respect to the large tableau simplex algorithm and the small tableau simplex algorithm are initially the same. The reason is that initially only artificial variables are basic, therefore, the condensed tableau contains all columns with respect to the non artificial variables.

Observe, furthermore, that a column with respect to an artificial variable is swapped into the tableau after each iteration of the small tableau simplex, where an artificial variable becomes co-basic and a non artificial variable basic. Theorem 3.21 implies that we could remove this column, but this results in shrinking the tableau leaking information.

With respect to small tableau simplex, we will not delete any columns. For efficiency reasons we use the result of Theorem 3.21 by making sure that no artificial variable will ever be selected to become basic. This can be done securely using the following extension to the column selection step.

Initially we set $\mathbf{s} = ((n + m + 1, 0), \ldots, (n + 2m, 0))$ and $\mathbf{u} = ((1, 1), \ldots, (n + m, 1))$. That is, to each (co-) basis entry we add a bit which equals 0 if the value of the entry is larger than $n + m$, i.e., when it corresponds to an artificial variable. Then, instead of choosing a column ℓ, where $t_{(m+1)\ell} < 0$, pick an ℓ, where $t_{(m+1)\ell} u_{\ell 2} < 0$. Indeed, observe that $t_{(m+1)\ell} u_{\ell 2} < 0$ if and only if $u_{\ell 1} \leq n + m$ and $t_{(m+1)\ell} < 0$. ◇

5.2.1.2 Initialize Phase II

The result of phase I is used to check whether the original LP has a feasible solution. If so, then from the tableau and basis returned by phase I, a tableau and basis are computed corresponding a basic feasible solution. Recall that in this step the artificial variables are removed from the basis and all rows with respect to redundant constraints are removed

from the tableau. However, by construction the any standard form LP is canonical having no redundant constraints.

In Section 3.3 we showed that phase II is initialized as follows:

1. If $\mathbf{T}$ is a tableau with respect to basis $\mathbf{s}$ for the artificial LP, where the corresponding basic feasible solution is a basic feasible solution to the original LP, then perform the following steps else stop and report that the original LP is infeasible.

2. Remove all artificial variables from $\mathbf{s}$; this results in a basis $\mathbf{s}'$ for the original LP.

3. Delete all columns with respect to the artificial variables from $\mathbf{T}$ resulting in $\mathbf{T}'$.

4. Compute the last row of $\mathbf{T}'$ so that $\mathbf{T}'$ is a tableau with respect to $\mathbf{s}'$ for the original LP.

Note that by construction, the large tableau simplex algorithm and the revised simplex algorithm did not initialize phase I with columns for the artificial variables. Hence only for small tableau simplex algorithm we need to consider secure column deletion.

Removing Artificial Variables from the Basis

To hide the fact whether the j-th basic variable is an artificial variable all operations on each row should be the same. Naively, to securely remove the artificial variables from the basis one could proceed as follows. For each $j \in \{1, \ldots, m\}$ compute the secret $[\ell]$ index of a nonzero entry in the first $n+m$ entries in the j-th row of the tableau. Then, securely pivot on entry $t_{j\ell}$. This results in $(n+m)m$ secure comparisons and $m^2(n+m)$ secure multiplications.

Lemma 5.6 shows that one can use the structure of LP (5.3), which is in standard form and equivalent to Eq. (5.2), to remove the artificial variables from the basis very efficiently, i.e., using no secure comparison and no expensive tableau updates.

Lemma 5.6. *Consider an LP in standard form, where* $\mathbf{b} \geq \mathbf{0}$*, and the corresponding artificial LP* (5.12)*. Suppose that phase I, that does not allow artificial variables to become basic, returns tableau* $\mathbf{T}$ *with basis* $\mathbf{s}$ *and co-basis* $\mathbf{u}$*. If* x_{n+m+k} *is basic then* $\mathbf{T}_{u_{n+k}} = \mathbf{T}_{n+k} = \pm\mathbf{e}_k$*.*

Proof. Let $\mathbf{T}$ and $\mathbf{s}$ be the tableau and basis returned by phase I. Let $\mathbf{B}$ be the corresponding basis matrix.

Recall that phase I is initialized with basis $(n+m+1, \ldots, n+2m)$. Since, by construction, an artificial variable cannot become basic when it is co-basic at some iteration, it follows that x_{n+m+k} is basic during all iterations in phase I and $s_k = n+m+k$.

Furthermore, $\mathbf{B}_k = \mathbf{e}_k$. Since $\mathbf{A}'_{n+k} = \pm\mathbf{e}_k = \pm\mathbf{B}_k$ the slack variable x_{n+k} has to be co-basic during all iterations in phase I. Hence $u_{n+k} = n+k$.

Lemma 3.41 implies $\mathbf{B}^{-1}\mathbf{e}_k = \mathbf{e}_k$, and, therefore,

$$\mathbf{T}_{n+k} = \mathbf{B}^{-1}\mathbf{A}'_{n+k} = \mathbf{B}^{-1}(\pm\mathbf{e}_k) = \pm\mathbf{B}^{-1}\mathbf{B}_k = \pm\mathbf{e}_k.$$

□

It follows that if x_{n+m+k} is basic then $s_k = n + m + k$ and x_{n+k} is co-basic. For small tableau simplex $u_{n+k} = n+k$. Furthermore, Theorem 3.42 implies that we can update the basis without changing the solution by replacing s_k with $u_{n+k} = n + k$. From Lemma 5.6 we conclude that the updated tableau is obtained by just multiplying row k of $\mathbf{T}$ by $t_{n+k} = \pm 1$.

In conclusion, to remove the artificial variables from the basis securely, we perform the following steps in each variant of simplex using rational pivoting:

1. Compute the locations of artificial variables in the basis securely: compute $[\gamma_i] \leftarrow [s_i = n + m + i]$. This equality comparison is sufficient as y_i is basic if and only if $s_i = n + m + i$. Otherwise, y_i has been selected to enter the basis at some step.
2. Update basis: $[s_i] \leftarrow [\gamma_i](n + i) + (1 - [\gamma_i])[s_i]$.
3. Compute the row multipliers $[\mathbf{w}]$ and update co-basis $[\mathbf{u}]$:

   Large Tableau Simplex: $[w_i] \leftarrow [\gamma_i][t_{i(n+i)}] + (1 - [\gamma_i])$.

   Small Tableau Simplex: $[w_i] \leftarrow [\gamma_i][t_{i(n+i)}] + (1 - [\gamma_i])$ and $[u_{n+i}] \leftarrow [\gamma_i](n + m + i) + (1 - [\gamma_i])[u_{n+i}]$.

   Revised Simplex: $[w_i] \leftarrow [\gamma_i][t^0_{i(n+i)}] + (1 - [\gamma_i])$
4. Update tableau: $[\mathbf{t}'_i] \leftarrow [w_i][\mathbf{t}_i]$.

We do have to be careful when integer pivoting is applied, since the pivot elements $t_{i(n+i)}$ can be negative. In Theorem 3.43 we showed that we need to multiply the tableau with minus one to keep the tableau consistent. Lemma 5.7 shows how to efficiently perform the pivots with respect to integer pivoting, taking care of negative pivots.

Lemma 5.7. *Let tableau* $\mathbf{T}$*, basis* $\mathbf{s}$ *and* q *be returned by phase I corresponding to solution* $\mathbf{x}$ *where integer pivoting is applied. Let* $\gamma_k = |s_k = n + m + k|_b$ *and* $\mathbf{T}'$ *be such that* $t'_{kj} = t_{kj}(\gamma_k t_{k(n+k)}/q + (1 - \gamma_k))$ *and* $t'_{ij} = t_{ij}$ *if* $k \neq i$.

Then $\mathbf{T}'$ *is the tableau corresponding to basic solution* $\mathbf{x}$ *where all artificial variables are co-basic.*

Proof. Let tableau $\mathbf{T}$, basis $\mathbf{s}$ and q be returned by phase I where integer pivoting is applied.

Let x_{n+m+k} be an artificial variable which is basic. Hence $\gamma_k = 1$. To remove artificial x_{n+m+k} from the basis the tableau $\mathbf{T}$ is updated by pivoting on $t_{k(n+k)}$ as a consequence of Lemma 5.6. The corresponding solution remains the same by Theorem 3.42.

Column $n + k$ of $\mathbf{T}$ satisfies $T_{n+k} = \pm q\mathbf{e}_k$ by applying Lemma 5.6 with respect to integer pivoting. Hence pivoting on $t_{k(n+k)}$ results in x_{n+m+k} becoming co-basic and x_{n+k} becoming basic.

By Theorem 3.43 the tableau is updated by

$$t'_{ij} = \alpha \frac{t_{k(n+k)}t_{ij} - t_{i(n+k)}t_{kj}}{q} = \alpha \frac{\alpha q t_{ij} - 0}{q} = t_{ij}, \qquad \text{if } i \neq k,$$

$$t'_{kj} = \alpha t_{kj},$$

where α denotes the sign of $t_{k(n+k)}$. Observe that since $|t_{i(n+i)}| = q$ it follows that $\alpha = \frac{t_{k(n+k)}}{q}$.

Note that if x_{n+m+k} is co-basic, then $\gamma_i = 0$ and $t'_{kj} = t_{kj}$ as required since no pivot operation is applied. □

In conclusion, to remove the artificial variables from the basis securely, we perform the following steps in each variant of simplex using integer pivoting:

1. Compute the locations of artificial variables in the basis securely: compute $[\gamma_i] \leftarrow [s_i = n + m + i]$. This equality comparison is sufficient as y_i is basic if and only if $s_i = n + m + i$. Otherwise, y_i has been selected to enter the basis at some step.
2. Update basis: $[s_i] \leftarrow [\gamma_i](n + i) + (1 - [\gamma_i])[s_i]$.
3. Compute inverse of q: $[q'] \leftarrow [q^{-1}]$.
4. Compute the row multipliers $[\mathbf{w}]$ and update co-basis $[\mathbf{u}]$:

   Large Tableau Simplex: $[w_i] \leftarrow [\gamma_i][q'][t_{i(n+i)}] + (1 - [\gamma_i])$.

   Small Tableau Simplex: $[w_i] \leftarrow [\gamma_i][q'][t_{i(n+i)}] + (1 - [\gamma_i])$ and $[u_{n+i}] \leftarrow [\gamma_i](n + m + i) + (1 - [\gamma_i])[u_{n+i}]$.

   Revised Simplex: $[w_i] \leftarrow [\gamma_i][t^0_{i(n+i)}] + (1 - [\gamma_i])$
5. Update tableau: $[\mathbf{t}'_i] \leftarrow [w_i][\mathbf{t}_i]$.

Secure Column Deletion

When the artificial variables are removed from the basis, their corresponding columns need to be removed. However, we showed in Remark 5.5 that this only applies to small tableau simplex, as the large tableau and revised simplex did not add columns with respect to the artificial variables when initializing phase I.

Suppose that the small tableau simplex algorithm for phase one is implemented using the extension suggested in Remark 5.5. Then the co-basis is given by $\mathbf{U}$, where $u_{2i} = |u_i \geq n + m|_b$. It follows that u_{2i} is equal to one if and only if the corresponding column corresponds to an artificial variable.

We apply $\mathsf{DelCols}(\mathbf{T}, \mathbf{u}_2)$, (Protocol 4.39), to compute securely delete the columns of $\mathbf{T}$ corresponding the artificial variables. In addition we apply $\mathsf{DelCols}(\mathbf{u_1}, \mathbf{u}_2)$ to securely compute a corresponding co-basis.

Note that a $\mathsf{DelCols}$ needs to be executed only once by adding $\mathbf{u}_1$ as a row to $\mathbf{T}$.

Finalizing the Tableau

As a last step the last row of the tableau needs to be changed to be consistent to the cost of the original linear program. Recall that the last row of $\mathbf{T}$ should be equal to

$$\mathbf{t} = (\mathbf{c} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A}', -\mathbf{c_s}\mathbf{B}^{-1}\mathbf{b}').$$

Consider a tableau $\mathbf{T}$ corresponding to basis $\mathbf{s}$ for the artificial LP such that no artificial variable is basic. Then $\mathbf{T}$, after deleting the columns for the artificial LP is equal to

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A}' & \mathbf{b}' \\ \mathbf{0} & 0 \end{pmatrix},$$

where $\mathbf{B} = \mathbf{A}'_{\mathbf{s}}$. Observe that the first m rows of $\mathbf{T}$ are equal to

$$(\mathbf{B}^{-1}\mathbf{A}', \mathbf{B}^{-1}\mathbf{b}').$$

Hence

$$\mathbf{t} = (\mathbf{c}, 0) - \mathbf{c}_{\mathbf{s}} \begin{pmatrix} \mathbf{t}_1 \\ \vdots \\ \mathbf{t}_m \end{pmatrix}.$$

It remains to show how to compute the vector $\mathbf{c}_{\mathbf{s}}$. By converting each s_i into a secret index $\boldsymbol{\sigma}_i$ it follows that $\mathbf{c}_{s_i} = \mathbf{c}\boldsymbol{\sigma}_i$. In other words $\mathbf{c}_{\mathbf{s}}$ is computed as follows

- For each $i = 1, \ldots, m$ compute
  1. Compute $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n + m)$.
  2. Compute $[v_i] = [\boldsymbol{\sigma}_i][\mathbf{c}]$.
- Set $[\mathbf{c}_{\mathbf{s}}] = [\mathbf{v}]$.

5.2.2 Two-Phase Simplex with One Artificial Variable

Recall that the two-phase simplex with one artificial variable considers the following artificial linear program:

$$\begin{array}{lrcl} \min & x_{n+m+1}, & & \\ \text{subject to} & \left(\mathbf{A} \quad \mathbf{I}_m \right) \mathbf{x} - x_{n+m+1} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}. \end{array} \tag{5.13}$$

The two-phase simplex with one artificial variable consists of the same steps as the standard two-phase simplex. We will discuss how to securely initialize phase I and phase II. The detailed protocols are presented in Appendix A.2.2.

5.2.2.1 Initialize Phase I

Recall that by construction the linear program in standard form is equivalent to

$$\begin{array}{lrcl} \min & \mathbf{cx}, & & \\ \text{subject to} & \left(\mathbf{A} \quad \mathbf{I}_m \right) \mathbf{x} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array}$$

by adding m slack variables to $\mathbf{x}$ and $\mathbf{I}_m$ to $\mathbf{A}$.

Observe furthermore that the latter linear program is in canonical form and we could apply the result of Theorem 3.47 to compute an initial feasible solution directly. However, it also follows by construction that if $\mathbf{b} \geq \mathbf{0}$ then phase I is skipped and phase II is initialized immediately. This reveals the fact that $\mathbf{b} \geq \mathbf{0}$.

In order to hide the fact that $\mathbf{b} \geq \mathbf{0}$, we consider yet another artificial LP with one artificial variable:

$$\begin{array}{lrcl} \min & x_{n+m+1}, & & \\ \text{subject to} & \left(\mathbf{A} \quad \mathbf{I}_m \right) \mathbf{x} + \beta x_{n+m+1} & = & \mathbf{b}, \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{5.14}$$

where $\beta = 2\left|\mathbf{b} \geq \mathbf{0}\right|_b - 1$.

The following Lemma shows how to securely find an initial basic feasible solution to Eq. (5.14).

Lemma 5.8. *Consider LP* (5.14)*. Let* $k = \text{argmin}(\mathbf{b})$*. Then the basis* $\mathbf{s} = (n+1, \ldots, n+k-1, n+m+1, n+k+1, \ldots, n+m)$ *corresponds to basic feasible solution* $\mathbf{x}$*, where*

$$x_i = \begin{cases} b_j - b_k, & \text{if } i = n+j, \\ \beta b_k, & \text{if } i = n+m+1, \\ 0, & \text{otherwise.} \end{cases}$$

Proof. If $b_k < 0$ then $\beta = -1$ and the lemma follows from Theorem 3.47.

Suppose that $b_k \geq 0$, then $\mathbf{b} \geq \mathbf{0}$ and $\beta = 1$. Observe that from the proof of Theorem 3.47 it also follows that $\mathbf{s}$ is a basis since the basis matrix

$$\mathbf{B} = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \beta\mathbf{1} \end{pmatrix}_{\mathbf{s}} = (\mathbf{e_1}, \ldots, \mathbf{e}_{k-1}, \mathbf{1}, \mathbf{e}_{k+1}, \ldots, \mathbf{e}_m)$$

has an inverse

$$\mathbf{B}^{-1} = \begin{pmatrix} \mathbf{e}_1 - \mathbf{e}_k \\ \vdots \\ \mathbf{e}_{k-1} - \mathbf{e}_k \\ \mathbf{e}_k \\ \mathbf{e}_{k+1} - \mathbf{e}_k \\ \vdots \\ \mathbf{e}_m - \mathbf{e}_k \end{pmatrix}.$$

The corresponding basic feasible solution $\mathbf{x}$ satisfies

$$\mathbf{x}_{\mathbf{s}} = \mathbf{B}^{-1}\mathbf{b} = \begin{pmatrix} b_1 - b_k \\ \vdots \\ b_{k-1} - b_k \\ b_k \\ b_{k+1} - b_k \\ \vdots \\ b_m - b_k \end{pmatrix}.$$

Hence the lemma follows from the fact that by the choice of k the solution satisfies $\mathbf{x} \geq \mathbf{0}$. □

The basic solution $\mathbf{x}$ of Lemma 5.8 can be computed from basic solution $\mathbf{x}'$ corresponding to basis $(n+1, \ldots, n+m)$ by a pivot operation. The following three lemmas show that this pivot operation can be done efficiently in case of respectively the large tableau simplex algorithm, the small tableau simplex algorithm and the revised simplex algorithm.

Lemma 5.9. *Consider the linear program Eq.* (5.14)*. Let* $\mathbf{T}$ *be the large tableau corresponding to basis* $(n+1, \ldots, n+m)$*. The computation*

$$t'_{ij} = t_{ij} - r_i t_{kj},$$

where $\mathbf{r} = \mathbf{1} - \beta\mathbf{e}_k + (\beta - 1)\mathbf{e}_{m+1}$*, corresponds to pivoting on element* $t_{k(n+m+1)}$*.*

Proof. Observe that $\beta \in \{-1, 1\}$ satisfies $1/\beta = \beta = \text{sgn}(\beta)$ and $\beta^2 = 1$. Furthermore, column $\mathbf{T}_{n+m+1} = (\beta\mathbf{1}, 1)$. Let $\delta_{ij} = |i = j|_b$ denote Kronecker's delta. Then for $i = 1, \ldots, m$

$$\begin{aligned} t'_{ij} &= t_{ij} - r_i t_{kj} \\ &= t_{ij} - (1 - \beta\delta_{ik}) t_{kj} \\ &= (1 - \delta_{ik})\,(t_{ij} - t_{kj}) + \delta_{ik}\,(t_{kj} - (1 - \beta) t_{kj}) \\ &= (1 - \delta_{ik}) \left(t_{ij} - \frac{t_{kj}\beta}{\beta} \right) + \delta_{ik} \left(\frac{t_{kj}}{\beta} \right) \\ &= (1 - \delta_{ik}) \left(t_{ij} - \frac{t_{kj} t_{i(n+m+1)}}{t_{k(n+m+1)}} \right) + \delta_{ik} \left(\frac{t_{kj}}{t_{k(n+m+1)}} \right) \end{aligned}$$

and

$$\begin{aligned} t'_{(m+1)j} &= t_{(m+1)j} - r_{m+1} t_{kj} \\ &= t_{(m+1)j} - (\beta) t_{kj} \\ &= t_{(m+1)j} - \frac{t_{kj}}{\beta} \\ &= t_{(m+1)j} - \frac{t_{(m+1)(n+m+1)} t_{kj}}{t_{k(n+m+1)}}. \end{aligned}$$

These are precisely the equations describing rational pivoting on $t_{k(n+m+1)}$ in tableau $\mathbf{T}$; see Theorem 3.29.

With respect to integer pivoting β might be negative. Observe furthermore that the basis matrix with respect to basis $(n+1, \ldots, n+m)$ is equal to $\mathbf{I}_m$. Hence $|\det(\mathbf{I}_m)|\mathbf{T} = \mathbf{T}$, so the tableaus with respect to basis $(n+1, \ldots, n+m)$ are equal in both simplex methods using either integer or rational pivoting.

Observe that for $i = 1, \ldots, m$

$$\begin{aligned} t'_{ij} &= t_{ij} - r_i t_{kj} \\ &= (1 - \delta_{ik})\,(t_{ij} - t_{kj}) + \delta_{ik}\,(t_{kj} - (1 - \beta) t_{kj}) \\ &= (1 - \delta_{ik})\beta\,(t_{ij}\beta - t_{kj}\beta) + \delta_{ik}\beta\,(t_{kj}) \\ &= (1 - \delta_{ik})\text{sgn}(t_{k(n+m+1)}) \frac{t_{ij} t_{k(n+m+1)} - t_{kj} t_{i(n+m+1)}}{1} + \delta_{ik}\text{sgn}(t_{k(n+m+1)}) t_{kj} \end{aligned}$$

and

$$\begin{aligned} t'_{(m+1)j} &= t_{(m+1)j} - r_{m+1} t_{kj} \\ &= t_{(m+1)j} - \frac{t_{kj}}{\beta} \\ &= \beta \left(t_{(m+1)j}\beta - t_{kj} \right) \\ &= \text{sgn}(t_{k(n+m+1)}) \frac{t_{(m+1)j} t_{k(m+n+1)} - t_{(m+1)(m+n+1)} t_{kj}}{1}. \end{aligned}$$

These are precisely the equations describing integer pivoting on $t_{k(n+m+1)}$ in tableau $\mathbf{T}$ of Theorem 3.43. □

Lemma 5.10. *Consider the linear program Eq.* (5.14)*. Let* $\mathbf{T}$ *be the condensed tableau corresponding to basis* $(n{+}1,\ldots,n{+}m)$ *and co-basis* $(1,\ldots,n,n{+}m{+}1)$*. The computation*

$$t'_{ij} = t_{ij} - r_i w_j,$$

where $\mathbf{r} = \mathbf{1} - \beta\mathbf{e}_k + (\beta-1)\mathbf{e}_{m+1}$ *and* $\mathbf{w} = \mathbf{t}_k + \mathbf{e}_{n+1}$*, corresponds to pivoting on element* $t_{k(n+m+1)}$*.*

Proof. From Lemma 5.9 it follows that if $j \neq n+m$, then $t'_{ij} = t_{ij} - r_i w_j$ will correspond to both rational pivoting and integer pivoting on $t_{k(n+m+1)}$ in tableau $\mathbf{T}$.

If $j = n+1$ and $i \neq m+1$,

$$\begin{aligned} t'_{i(n+1)} &= t_{i(n+1)} - r_i w_{n+1} \\ &= t_{i(n+1)} - (1-\beta\delta_{ik})(t_{k(n+1)}+1) \\ &= \beta - (1-\beta\delta_{ik})(\beta+1) \\ &= -(1-\delta_{ik}) + \delta_{ik}\beta \\ &= (1-\delta_{ik})\frac{-t_{i(n+1)}}{t_{k(n+1)}} + \delta_{ik}\frac{1}{t_{k(n+1)}} \end{aligned}$$

and

$$\begin{aligned} t'_{(m+1)(n+1)} &= t_{(m+1)(n+1)} - r_{m+1} w_{n+1} \\ &= t_{(m+1)(n+1)} - (t_{k(n+1)}+1) \\ &= -\beta \\ &= -\frac{t_{(m+1)(n+1)}}{t_{k(n+1)}}. \end{aligned}$$

Note that these equations correspond to the column replacement with respect to rational pivoting; see Lemma 3.37.

Observe that the basis $\mathbf{B}$ matrix is equal to $\mathbf{I}_m$ and thus that $|\det \mathbf{B}| = 1$. Hence

$$\begin{aligned} t'_{i(n+1)} &= t_{i(n+1)} - r_i w_{n+1} \\ &= -(1-\delta_{ik}) + \delta_{ik}\beta \\ &= (1-\delta_{ik})\left(-\beta^2\right) + \delta_{ik}\beta \\ &= (1-\delta_{ik})\left(-\mathrm{sgn}(t_{k(n+1)})t_{i(n+1)}\right) + \delta_{ik}\mathrm{sgn}(t_{k(n+1)})|\det \mathbf{B}|, \end{aligned}$$

for $i = 1,\ldots,m$, and

$$\begin{aligned} t'_{(m+1)(n+1)} &= t_{(m+1)(n+1)} - r_{m+1} w_{n+1} \\ &= -\beta \\ &= -\mathrm{sgn}(t_{k(n+1)})t_{(m+1)(n+1)}. \end{aligned}$$

Note that these equations correspond to the column replacement with respect to integer pivoting taking negative pivoting into account; see Lemma 3.37 and Theorem 3.43. □

Lemma 5.11. *Consider the linear program Eq.* (5.14)*. Let* $\mathbf{D}$ *be the revised tableau corresponding to basis* $(n+1,\ldots,n+m)$*. The computation*

$$d'_{ij} = d_{ij} - r_i(e_k)_j,$$

where $\mathbf{r} = \mathbf{1} - \beta\mathbf{e}_k + (\beta-1)\mathbf{e}_{m+1}$*, corresponds to pivoting on element* $t_{k(n+m+1)}$*.*

Proof. Observe that the pivot row $\mathbf{d}_k = \mathbf{e}_k$. So the statement can be reformulated as

$$d'_{ij} = d_{ij} - r_i d_{kj}.$$

The proof that this equation is equivalent to pivoting is analogous to the proof of Lemma 5.9. □

To initialize phase I, we apply Theorem 3.21 again to exclude the columns corresponding to the artificial variable in the tableau $\mathbf{T}$. By Remark 5.5 we will not delete this column in small tableau simplex.

In conclusion, given LP (5.2), the following steps are performed to initialize phase I:

1. Let $\mathbf{s} \leftarrow (n+1,\ldots,n+m)$.
2. Compute $([b],[\mathbf{k}]) \leftarrow \mathsf{FindMin}([\mathbf{b}],\mathsf{LTZ})$ for $i = 1,\ldots,m$.
3. Compute $[\beta] \leftarrow 2[b \geq 0] - 1$.
4. Set $[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}(\mathbf{s},[\mathbf{k}],n+m+1)$ and do the following:

   Large Tableau Simplex: (a) Let

   $$[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}] & [\mathbf{I}_m] & [\mathbf{b}] \\ [\mathbf{0}] & [\mathbf{0}] & [0] \end{pmatrix}$$

   be the tableau corresponding to basis $(n+1,\ldots,n+m)$, where column $n+m+1$ is deleted.

   (b) Compute the pivot row by $[\mathbf{p}^r] \leftarrow [\mathbf{T}][\mathbf{k}]$.

   (c) Compute $[\mathbf{r}] \leftarrow \mathbf{1} - [\beta][\mathbf{k}] + ([\beta]-1)\mathbf{e}_{m+1}$.

   (d) Compute $[\mathbf{T}']$ corresponding to $[\mathbf{s}]$ by

   $$[t'_{ij}] \leftarrow [t_{ij}] - [r_i][p^r_j].$$

   Small Tableau Simplex: (a) Let $\mathbf{u} \leftarrow (1,\ldots,n,n+m+1)$.

   (b) Let

   $$[\mathbf{T}] \leftarrow \begin{pmatrix} & [\beta] & \\ [\mathbf{A}] & \vdots & [\mathbf{b}] \\ & [\beta] & \\ [\mathbf{0}] & [1] & [0] \end{pmatrix}$$

   be the condensed tableau corresponding to basis $(n+1,\ldots,n+m)$ and co-basis $(1,\ldots,n,n+1)$.

   (c) Compute $[\mathbf{w}] \leftarrow [\mathbf{T}][\mathbf{k}] + \mathbf{e}_{n+1}$.

   (d) Compute $[\mathbf{r}] \leftarrow \mathbf{1} - [\beta][\mathbf{k}] + ([\beta]-1)\mathbf{e}_{m+1}$.

(e) Compute $[\mathbf{T}']$ corresponding to $[\mathbf{s}]$ by

$$[t'_{ij}] \leftarrow [t_{ij}] - [r_i][w_j].$$

(f) Compute $[\mathbf{u}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}], n+1, \sum_{i=1}^{m}[k_i]i)$.

Revised Simplex: (a) Let

$$[\mathbf{D}] \leftarrow \begin{pmatrix} [\mathbf{I}_m] & \mathbf{0} \\ [\mathbf{0}] & [1] \end{pmatrix}$$

and

$$[\mathbf{T}^0] \leftarrow \begin{pmatrix} [\mathbf{A}] & [\mathbf{I}_m] & [\mathbf{b}] \\ [\mathbf{0}] & [\mathbf{0}] & [0] \end{pmatrix}$$

be such that $\mathbf{T} = \mathbf{D}\mathbf{T}^0$, the tableau corresponding basis $(n+1, \ldots, n+m)$, where column $n+m+1$ is deleted

(b) Compute $[\mathbf{r}] \leftarrow \mathbf{1} - [\beta][\mathbf{k}] + ([\beta] - 1)\mathbf{e}_{m+1}$.

(c) Compute $[\mathbf{D}']$ corresponding to $[\mathbf{s}]$ by

$$[d'_{ij}] \leftarrow [d_{ij}] - [r_i][k_j].$$

5.2.2.2 Initialize Phase II

Similarly to Section 5.2.1 we will show how to remove the artificial variable from the basis while we hide its position in the basis. For the small tableau simplex algorithm, we show next how to securely delete the column with respect to the artificial variable. Lastly, we observe that computing the last row of the tableau for the original LP is exactly the same as for the standard two-phase simplex algorithm.

Removing the Artificial Variable from the Basis

Unfortunately, we cannot assign directly a nonzero pivot entry in the tableau to remove x_{n+m+i} from the basis, similar to the standard two-phase simplex. We show that still it is not necessary to search the whole row for the large tableau simplex algorithm and the revised simplex algorithm to find a nonzero element in Lemma 5.12.

Lemma 5.12. *Consider an LP in standard form and the corresponding artificial LP* (5.14)*. Suppose that phase 1, that does not allow the artificial variable to become basic, returns tableau* $\mathbf{T}$ *with basis* $\mathbf{s}$ *and co-basis* $\mathbf{u}$*. If* $s_j = n+m+1$ *for some* $j \in \{1, \ldots, m\}$ *then* $t_{j(n+i)} \neq 0$*, for some* $i \in \{1, \ldots, m\}$*.*

Proof. Consider an LP in standard form and the corresponding artificial LP with one artificial variable Eq. (5.14). Let $\mathbf{T}$ with basis $\mathbf{s}$ be the tableau and basis returned by phase I corresponding to solution $(\mathbf{x}, y)$. Let $\mathbf{B}$ be the corresponding basis matrix. Let $k \in \{1, \ldots, m\} = \operatorname{argmin}(\mathbf{b})$. Then column $\mathbf{A}'_{n+i} = \mathbf{e}_i$, where $\mathbf{A}' = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}$.

Recall that phase I is initialized with basis $(n+1, \ldots, n+k-1, n+m+1, n+k+1, \ldots, n+m)$. If y is basic then $s_k = n+m+1$, because, by construction, an artificial variable cannot become basic when it is co-basic at some iteration. Since $\mathbf{B}$ is invertible, the k-th row of $\mathbf{B}^{-1}$ is not equal to $\mathbf{0}$. Hence, there should be an $i \in \{1, \ldots, m\}$ where

$$t_{k(n+i)} = \left(\mathbf{B}^{-1}\mathbf{e}_i\right) = \left(\mathbf{B}_i^{-1}\right)_k \neq 0.$$

□

Unfortunately, since the columns in the small tableau simplex algorithm are shuffled, it is unknown which columns correspond to the slack variables. Hence, we cannot use Lemma 5.12 to avoid searching the whole row to find a nonzero element.

Recall that, with respect to integer pivoting, we need to include the sign of the nonzero pivot element in the calculations.

In conclusion, to remove the artificial variables from the basis securely, we perform the following steps in each variant of simplex using rational pivoting. Let $[\mathbf{k}]$ be the secret index of the minimum of $[\mathbf{b}]$, computed when initializing phase I.

1. Compute the location of the artificial variable in the basis securely: compute $[\gamma] \leftarrow [\mathbf{s}][\mathbf{k}] = n + m + 1$. This equality comparison is sufficient as x_{n+m+1} is basic if and only if $s_k = n + m + 1$. Otherwise, x_{n+m+1} has been selected to enter the basis at some step.

2. Update tableau and basis by using Lemma's 5.2, 5.3 and 5.4:

   Large Tableau Simplex: (a) Compute pivot row: $[\mathbf{p}^r] \leftarrow [\mathbf{k}][\mathbf{T}]$
   (b) Compute the secret index $[\boldsymbol{\ell}]$ of the first nonzero element in $[\mathbf{p}^r]$, the pivot column $[\mathbf{p}^c] \leftarrow [\mathbf{T}][\boldsymbol{\ell}]$ and pivot element $[p] \leftarrow [\mathbf{p}^r][\boldsymbol{\ell}]$.
   (c) Update basis: $[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], \sum_{i=1}^{n+m} [\ell_i] i)$.

   Small Tableau Simplex: (a) Compute pivot row: $[\mathbf{p}^r] \leftarrow [\mathbf{k}][\mathbf{T}]$
   (b) Compute the secret index $[\boldsymbol{\ell}]$ of the first nonzero element in $[\mathbf{p}^r]$, the pivot column $[\mathbf{p}^c] \leftarrow [\mathbf{T}][\boldsymbol{\ell}]$ and the pivot element $[p] \leftarrow [\mathbf{p}^r][\boldsymbol{\ell}]$.
   (c) Update basis: $[\mathbf{s}'] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\mathbf{u}][\boldsymbol{\ell}])$.
   (d) Update co-basis $[\mathbf{u}'] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}], [\boldsymbol{\ell}], [\mathbf{s}][\mathbf{k}])$.

   Revised Simplex: (a) Compute pivot row of $\mathbf{T}$ by $[\mathbf{t}] \leftarrow [\mathbf{k}][\mathbf{D}][\mathbf{T}^0]$ and the pivot row of $\mathbf{D}$ by $[\mathbf{p}^r] \leftarrow [\mathbf{k}][\mathbf{D}]$.
   (b) Compute the secret index $[\boldsymbol{\ell}]$ of the first nonzero element in $[\mathbf{t}]$, the pivot column $[\mathbf{p}^c] \leftarrow [\mathbf{D}][\mathbf{T}^0][\boldsymbol{\ell}]$ and the pivot element $[p] \leftarrow [\mathbf{p}^r][\boldsymbol{\ell}]$.
   (c) Update basis: $[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], \sum_{i=1}^{n+m} [\ell_i] i)$.

3. Compute $[v]$, $[\mathbf{w}]$ and $[\mathbf{r}]$ of Lemma 5.2, 5.3 or 5.4.

4. If integer pivoting is applied then $[v] \leftarrow (1 - 2[p < 0])[v]$ and computing $[\mathbf{w}] \leftarrow (1 - 2[p < 0])[\mathbf{w}]$.

5. Prepare the real or dummy pivot by computing $[v] \leftarrow (1 - [\gamma]) + [\gamma][v]$ and $[\mathbf{w}] \leftarrow [\gamma][\mathbf{w}]$.

6. Update tableau by $[t'_{ij}] = [t_{ij}][v] - [w_i][r_j]$.

Secure Column Deletion

When the artificial variable is removed from the basis, its corresponding columns need to be removed. However, as we showed in the previous section this only applies to small tableau simplex, as the large tableau and revised simplex did not add columns with respect to the artificial variables when initializing phase I. We show how to securely delete one column with respect to the artificial variable for small tableau simplex.

Suppose that the small tableau simplex algorithm for phase one is implemented using the extension suggested in Remark 5.5. Then the co-basis is given by $\mathbf{U}$, where $u_{2i} = |u_i \geq n + m|_b$. It follows that u_{2i} is equal to 1 if and only if the corresponding column corresponds to an artificial variable.

In conclusion, we apply $\mathsf{DelCol}(\mathbf{T}, \mathbf{u}_2)$, Protocol 4.38, to securely delete the columns of $\mathbf{T}$ corresponding the artificial variables. In addition we apply $\mathsf{DelCol}(\mathbf{u_1}, \mathbf{u}_2)$ to securely compute a corresponding co-basis.

Note that a DelCol needs to be executed only once by adding $\mathbf{u}_1$ as a row to $\mathbf{T}$.

5.2.3 Big-M Method

This section shows how to implement the big-M method discussed in Section 3.3.3. The detailed protocols are presented in Appendix A.2.3.

The big-M method may be seen as phase I of the two-phase simplex that solves essentially the same artificial linear program, with a different cost function. Initialization of the big-M method is, therefore, equivalent to initializing phase I of the two-phase simplex algorithm.

The remaining issue is to accommodate M in the simplex tableaus. We discussed in Section 3.3.3 two different ways: one where M is represented by some large enough value, and one where M is a hypothetic value (Remark 3.50).

First of all, consider a linear program in standard form. The Big-M method solves a linear program of the form

$$\begin{array}{lrcl} \min & \mathbf{cx} + M\sum_{i=1}^{p} y_i, & & \\ \text{subject to} & \mathbf{Q}\begin{pmatrix} \mathbf{A} & \mathbf{I_m} \end{pmatrix}\mathbf{x} + \mathbf{Cy} & = & \mathbf{Qb}, \\ & \mathbf{x}, \mathbf{y} & \geq & \mathbf{0}, \end{array} \tag{5.15}$$

where we use the general representation of the artificial linear program.

If $p = m$, $\mathbf{Q} = \mathrm{diag}(\boldsymbol{\beta})$ and $\mathbf{C} = \mathbf{I_m}$, then we have the constraints of the artificial linear program that is solved by the standard two-phase simplex. But if $p = 1$, $\mathbf{Q} = \mathbf{I}_m$, and $\mathbf{C} = \beta\mathbf{1}$, then we have the constraints of the artificial linear program that is solved by the two-phase simplex based on one artificial variable.

We limit the choices for p, $\mathbf{Q}$, and $\mathbf{C}$ to the two choices suggested above. If $p = m$, $\mathbf{Q} = \mathrm{diag}(\boldsymbol{\beta})$, and $\mathbf{C} = \mathbf{I_m}$, we use the protocols described in Section 5.2.1 to initialize the big-M method. And if $p = 1$, $\mathbf{Q} = \mathbf{I}_m$, and $\mathbf{C} = \beta\mathbf{1}$, we use the protocols described in Section 5.2.2 to initialize the big-M method.

With respect to computing the last row of an initial tableau for the big-M method, observe the following. Suppose that $\mathbf{T}$ is an initial tableau with respect to phase I of the two-phase algorithm. An initial tableau $\mathbf{T}$ for the big-M method is derived as follows. If M is represented as a large value, then the last row of $\mathbf{T}$ is updated by

$$\mathbf{t}_{m+1} \leftarrow M\mathbf{t}_{m+1} + \mathbf{c}.$$

And if M is hypothetical (see Remark 3.50), then the last row of $\mathbf{T}$ is given by

$$t_{(m+1)i} \leftarrow (c_i, t_{(m+1)i})$$

and thus $\mathbf{T}$ has one extra row.

Suppose $M \in \mathbb{Z}_{\langle k \rangle}$ satisfies the conditions of Lemma 3.49. Then the big-M method performs the following steps:

1. Run the protocols of Section 5.2.1 or 5.2.2 to find a tableau and (co-)basis initializing phase I, without the columns with respect to the artificial variables.
2. Replace the last row of $[\mathbf{T}]$ by

   Large Tableau Simplex: $[\mathbf{t}_{m+1}] \leftarrow M[\mathbf{t}_{m+1}] + [\mathbf{c}]$.

   Small Tableau Simplex: $[\mathbf{t}_{m+1}] \leftarrow M[\mathbf{t}_{m+1}] + [\mathbf{c}]$.

   Revised Simplex: $[\mathbf{t}^0_{m+1}] \leftarrow [\mathbf{c}]$, and $[\mathbf{d}_{m+1}] \leftarrow M[\mathbf{d}_{m+1}]$.
3. Run the corresponding simplex iterations using protocols of Section 5.1
4. Output the result.

If M is hypothetical then the big-M method performs the following steps:

1. Run the protocols of Section 5.2.1 or 5.2.2 to find a tableau $[\mathbf{T}]$ and (co-)basis $[\mathbf{s}]$ and $[\mathbf{u}]$ initializing phase I.
2. Replace the last row of $[\mathbf{T}]$ by two rows, where

   Large Tableau Simplex: $[t_{(m+2)i}] \leftarrow [t_{(m+1)i}]$ and $[t_{(m+1)i}] \leftarrow [c_i]$.

   Small Tableau Simplex: $[t_{(m+2)i}] \leftarrow [t_{(m+1)i}]$ and $[t_{(m+1)i}] \leftarrow [c_i]$.

   Revised Simplex: $[t^0_{(m+2)i}] \leftarrow [t^0_{(m+1)i}]$ and $[t^0_{(m+1)i}] \leftarrow [c_i]$, $[d_{(m+2)i}] \leftarrow [d_{(m+1)i}]$, and $[d_{(m+1)i}] \leftarrow 0$.
3. Run the corresponding simplex iterations using protocols of Section 5.1, with the following changes:
   - Column Selection: Select $[\ell]$ such that

     $$t_{(m+2)\ell} < 0 \vee \left(t_{(m+2)\ell} = 0 \wedge t_{(m+1)\ell} < 0\right),$$

     which can be surely computed by

     $$[t_{(m+2)\ell} < 0] + [t_{(m+2)\ell} = 0][t_{(m+1)\ell} < 0].$$
   - Update tableau: Update row $m + 2$ as well.
4. Output the result.

5.3 Secure Simplex Verification

In Section 3.1.4 we showed how to compute a *certificate* of correctness and how to use it to check the validity of the result returned by simplex. In this section we show how to extract and check such a certificate securely. We will discuss how to securely extract and check a certificate when simplex reports that the optimal has been found, the linear program is infeasible, and the linear program is unbounded. The detailed protocols are presented in Appendix A.3.

5.3.1 Verification of Optimality

Suppose that $\mathbf{T}$, $\mathbf{s}$, and $\mathbf{x}$ are returned being the optimal tableau, basis, and solution of a given LP in standard form. If small tableau simplex is applied, suppose that also the co-basis $\mathbf{u}$ is returned.

We will show how to extract a $\mathbf{p}$ from $\mathbf{T}$ such that $(\mathbf{x}, \mathbf{p})$ is a certificate of optimality if the simplex tableau $\mathbf{T}$ is indeed the tableau corresponding to basis $\mathbf{s}$ and solution $\mathbf{x}$.

We showed in Section 3.1.4.1 that $\mathbf{p}$ should be the solution of the dual linear program. Similarly to the discussion in Section 3.1.4 one observes that the dual of an LP in standard form is given by

$$\begin{array}{rlcl} \max & \mathbf{pb}, & & \\ \text{subject to} & \mathbf{pA} & \leq & \mathbf{c}, \\ & \mathbf{p} & \leq & \mathbf{0}. \end{array} \tag{5.16}$$

By Theorem 3.25 it follows that $\mathbf{x}$ is optimal to the primal LP if and only if (i) $\mathbf{x}$ is feasible, and (ii) $\mathbf{p}$ is feasible to the corresponding dual linear program and $\mathbf{cx} = \mathbf{pb}$.

Hence, to verify the output of simplex reporting the result to be optimal, we need to extract such $\mathbf{p}$ from $\mathbf{T}$, $\mathbf{s}$, and $\mathbf{u}$. Lemma 5.13 shows that if the simplex output is correct, how such $\mathbf{p}$ can be easily extracted from $\mathbf{T}$.

Lemma 5.13. *Consider a tableau* $\mathbf{T}$ *with respect to basis* $\mathbf{s}$ *and solution* $\mathbf{x}$*. Then* $(\mathbf{x}, \mathbf{p})$ *is a certificate of optimality, where*

$$\mathbf{p} = -(t_{(m+1)(n+1)}, \ldots, t_{(m+1)(n+m)}).$$

Proof. Suppose that $\mathbf{x}$ is a solution to with respect to tableau $\mathbf{T}$ corresponding to basis $\mathbf{s}$. Let $\mathbf{B}$ be the corresponding basis matrix.

We have by Theorem 3.7 that $\mathbf{x}$ is optimal if and only if no cost improving basic feasible directions exist at $\mathbf{x}$, i.e., from Lemma 3.9 it follows that $\mathbf{x}$ is optimal if and only if $(t_{(m+1)1}, \ldots, t_{(m+1)(n+m)}) = \overline{\mathbf{c}} \geq \mathbf{0}$.

Recall that the tableau $\mathbf{T}$ satisfies by definition

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c_s}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{c} & \mathbf{0} & 0 \end{pmatrix}.$$

Therefore, $\mathbf{p} = -(t_{(m+1)(n+1)}, \ldots, t_{(m+1)(n+m)}) = \mathbf{c_s}\mathbf{B}^{-1}$.

Since $\mathbf{x}$ is optimal we have by Theorem 3.7 and Lemma 3.9

$$\mathbf{c} - \mathbf{c_s}\mathbf{B}^{-1}\mathbf{A} = \mathbf{c} - \mathbf{pA} \geq \mathbf{0}$$

and

$$-\mathbf{c_s}\mathbf{B}^{-1}\mathbf{I}_m \geq \mathbf{0}.$$

Hence $\mathbf{p}$ is feasible to the dual LP (5.16).

Finally,‘

$$\mathbf{cx} = \mathbf{c_s}\mathbf{x_s} = \mathbf{c_s}\mathbf{B}^{-1}\mathbf{b} = \mathbf{pb}.$$

Hence $\mathbf{p}$ is optimal by Theorem 3.24.

Hence $(\mathbf{x}, \mathbf{p})$ is a certificate of optimality. □

It follows that to verify the output of simplex we need to verify whether $\mathbf{x}$ is feasible to the given LP, $\mathbf{p}$ is feasible to its dual, and $\mathbf{pb} = \mathbf{cx}$. If all tests pass then, since $(\mathbf{x}, \mathbf{p})$ is a certificate of optimality, we know for sure that $\mathbf{x}$ is optimal.

To verify optimality securely, let tableau $[\mathbf{T}]$, basis $[\mathbf{s}]$, solution $[\mathbf{x}]$, and $[q]$ be returned by the simplex protocols. In case of small tableau simplex let $[\mathbf{u}]$ be the co-basis returned by simplex and in case of revised simplex let $[\mathbf{D}]$ be the revised tableau returned by simplex.

To securely extract and verify a certificate of optimality, we perform the following steps:

1. *Extract dual solution:*

   Large Tableau Simplex: Set $[\mathbf{p}] \leftarrow -([t_{(m+1)(n+1)}], \ldots, [t_{(m+1)(n+m)}])$.

   Small Tableau Simplex: (a) Compute the last row $\mathbf{t}$ of the corresponding large tableau from the condensed tableau $[\mathbf{T}]$ and co-basis $[\mathbf{u}]$ by: $[t_{u_i}] \leftarrow [t_{(m+1)i}]$.
   (b) Set $[\mathbf{p}] \leftarrow -([t_{(m+1)(n+1)}], \ldots, [t_{(m+1)(n+m)}])$.

   Revised Simplex: Set $[\mathbf{p}] \leftarrow ([d_{(m+1)1}], \ldots, [d_{(m+1)m}])$.

2. *Verify Certificate:*

   Rational Pivoting: Set $[q] \leftarrow 1$ and $[\chi] \leftarrow 1$.

   Integer Pivoting: Verify positivity of q by $[\chi] \leftarrow [q] > 0$.

   (a) Verify feasibility of primal solution: $[\boldsymbol{\alpha}] \leftarrow [\mathbf{A}][\mathbf{x}] \leq [q][\mathbf{b}]$, and $[\boldsymbol{\alpha}'] \leftarrow [\mathbf{x}] \geq \mathbf{0}$.
   (b) Verify feasibility of dual solution: $[\boldsymbol{\beta}] \leftarrow [\mathbf{p}][\mathbf{A}] \leq [q][\mathbf{c}]$, and $[\boldsymbol{\beta}'] \leftarrow [\mathbf{p}] \leq \mathbf{0}$.
   (c) Verify equal costs: $[\gamma] \leftarrow [\mathbf{x}][\mathbf{c}] = [\mathbf{p}][\mathbf{b}]$.
   (d) Compute result:

$$\delta \leftarrow \mathsf{EQZ}\left([\bar{\chi}] + [\bar{\gamma}] + \sum_{i=1}^{m}\left([\bar{\alpha}_i] + [\bar{\beta}'_i]\right) + \sum_{j=1}^{n}\left([\bar{\alpha}'_j] + [\bar{\beta}_j]\right)\right), \tag{5.17}$$

   where $\bar{b} = 1 - b$ for any $b \in \{0, 1\}$.

In the last step we verify whether all tests have been passed and open the result. Naively, one would multiply all results of the tests with each other and conclude correctness if δ is equal to one. However, observe that if $x \geq 0$ and $y \geq 0$ then $x + y = 0$ if and only if $x = y = 0$. Hence $\delta = 0$ if and only if all bits in the sum are equal to zero, and, therefore, all tests should have been passed.

5.3.2 Verification of Infeasibility

To show that a linear program is infeasible, we need to show that no solution exists satisfying the constraints. In Section 3.1.4.2 we showed Farkas' lemma, which states that the system $\mathbf{Ax} = \mathbf{b}$ has no solution $\mathbf{x} \geq \mathbf{0}$ if and only if there exists some $\mathbf{p}$ such that $\mathbf{pA} \leq \mathbf{0}$ and $\mathbf{pb} > 0$.

We will show how to apply Farkas' lemma to extract a $\mathbf{p}$ from $\mathbf{T}$ and $\mathbf{s}$ so that they are a certificate of infeasibility of the linear program Eq. (5.2).

The tableau $\mathbf{T}$ that simplex returns while reporting that the linear program is infeasible depends heavily on what variant of simplex is applied, being either

- standard two-phase simplex, or
- two-phase simplex with one artificial variable, or
- the big-M method with m artificial variables, or
- the big-M method with 1 artificial variable.

Each variant solves a different linear program, and therefore, has different tableaus. For the sake of simplicity, we can generalize those four linear program as follows

$$\begin{array}{lrcl} \min & \gamma\mathbf{cx} + (1-\gamma+\gamma M)\sum_{i=1}^{p} y_i, & & \\ \text{subject to} & \mathbf{Q}\begin{pmatrix}\mathbf{A} & \mathbf{I_m}\end{pmatrix}\mathbf{x} + \mathbf{Cy} & = & \mathbf{Qb}, \\ & \mathbf{x},\mathbf{y} & \geq & \mathbf{0}, \end{array} \tag{5.18}$$

where in case of

standard two-phase simplex $\gamma = 0$, $p = m$, $\mathbf{C} = \mathbf{I}_m$ and $\mathbf{Q} = \operatorname{diag}(\boldsymbol{\beta})$, where $\beta_i = 1 - 2(b_i < 0)$, or

two-phase simplex with one artificial variable $\gamma = 0$, $p = 1$, $\mathbf{C} = \beta\mathbf{1}$, where $\beta = 1 - 2(\min(\mathbf{b}) < 0)$ and $\mathbf{Q} = \mathbf{I}_m$, or

the big-M method with m artificial variables $\gamma = 1$, $p = m$, $\mathbf{C} = \mathbf{I}_m$ and $\mathbf{Q} = \operatorname{diag}(\boldsymbol{\beta})$, where $\beta_i = 1 - 2(b_i < 0)$, or

the big-M method with 1 artificial variable $\gamma = 1$, $p = 1$, $\mathbf{C} = \beta\mathbf{1}$, where $\beta = 1 - 2(\min(\mathbf{b}) < 0)$ and $\mathbf{Q} = \mathbf{I}_m$.

Lemma 5.14 shows how to extract such $\mathbf{p}$ from the tableau returned by the simplex algorithm while reporting that the linear program is infeasible.

Lemma 5.14. *Suppose that an LP in standard from is given. Consider tableau* $\mathbf{T}$ *for LP* (5.18) *with respect to basis* $\mathbf{s}$ *and some optimal solution* $(\mathbf{x}, \mathbf{y})$*. If* $y_i > 0$ *for some* i*, then* $\mathbf{p}$ *is a certificate of infeasibility with respect to the given LP, where*

$$\mathbf{p} = -(t_{(m+1)(n+1)}, \ldots, t_{(m+1)(n+m)}).$$

Proof. Observe firstly that $\mathbf{Ax} \leq \mathbf{b}$ if and only if there exists some $\mathbf{x}^s \geq \mathbf{0}$ such that $\mathbf{Ax} + \mathbf{x}^s = \mathbf{b}$. Applying Farkas' lemma to the latter yields that $\mathbf{Ax} + \mathbf{x}^s = \mathbf{b}$ has no solution $(\mathbf{x}, \mathbf{x}^s) \geq \mathbf{0}$ if and only if there exists some $\mathbf{p}$ such that $\mathbf{p}\begin{pmatrix}\mathbf{A} & \mathbf{I}_m\end{pmatrix} \leq \mathbf{0}$ and $\mathbf{pb} > 0$.

In conclusion, the linear given LP has no feasible solution if and only if there exists some $\mathbf{p} \leq \mathbf{0}$ such that $\mathbf{pA} \leq \mathbf{0}$ and $\mathbf{pb} > 0$.

Suppose that $\mathbf{T}$ is a tableau with respect to Eq. (5.18) corresponding to basis $\mathbf{s}$ and optimal solution $(\mathbf{x}, \mathbf{y})$. Let $\mathbf{B}$ be the basis matrix and

$$\mathbf{c}' = (\gamma\mathbf{c}, \mathbf{0}, (1-\gamma+\gamma M)\mathbf{1})$$

denote the corresponding cost coefficients. We will show that if $\mathbf{y} \neq \mathbf{0}$, then

$$\mathbf{p} = -(t_{(m+1)(n+1)}, \ldots, t_{(m+1)(n+m)})$$

proves infeasibility of Eq. (5.18).

The tableau $\mathbf{T}$ can by definition be written as

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}'_{\mathbf{s}}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{QA} & \mathbf{Q} & \mathbf{C} & \mathbf{Qb} \\ \gamma\mathbf{c} & \mathbf{0} & (1-\gamma+\gamma M)\mathbf{1} & 0 \end{pmatrix}.$$

First, observe that

$$-\mathbf{p} = (t_{(m+1)(n+1)}, \ldots, t_{(m+1)(n+m)}) = -\mathbf{c}'_{\mathbf{s}}\mathbf{B}^{-1}\mathbf{Q}.$$

Since the tableau corresponds to an optimal solution,

$$(t_{(m+1)(n+1)}, \ldots, t_{(m+1)(n+m)}) \geq \mathbf{0}$$

and

$$t_{(m+1)j} = -\mathbf{c}'_{\mathbf{s}}\mathbf{B}^{-1}\mathbf{QA}_j \geq 0$$

for $j \leq n$. Hence $\mathbf{p} \leq \mathbf{0}$ and $\mathbf{pA} \leq \mathbf{0}$.

Finally, $\mathbf{y} \neq \mathbf{0}$ implies

$$\mathbf{c}' \begin{pmatrix} \mathbf{x} \\ \mathbf{x}^s \\ \mathbf{y} \end{pmatrix} = \gamma\mathbf{cx} + (1-\gamma+\gamma M)\mathbf{y} > 0.$$

Indeed if the two-phase simplex is applied, then $\gamma = 0$ and $\mathbf{c}' \begin{pmatrix} \mathbf{x} \\ \mathbf{x}^s \\ \mathbf{y} \end{pmatrix} = \sum_{i=1}^{p} y_i > 0$. On the other hand if the big-M method is applied then $\gamma = 1$ and by construction $\mathbf{cx} < M\sum_{i=1}^{p} y_i$.

It follows that

$$0 < \mathbf{c}' \begin{pmatrix} \mathbf{x} \\ \mathbf{x}^s \\ \mathbf{y} \end{pmatrix} = -t_{(m+1)(n+m+p+1)} = \mathbf{c}'_{\mathbf{s}}\mathbf{B}^{-1}\mathbf{Qb} = \mathbf{pb}.$$

□

Let tableau [$\mathbf{T}$], basis [$\mathbf{s}$], and solution [$\mathbf{x}$] be returned by the simplex protocols. In case of small tableau simplex let [$\mathbf{u}$] be the co-basis returned by simplex and in case of revised simplex let [$\mathbf{D}$] be the revised tableau returned by simplex.

In conclusion, to securely extract and verify a certificate of infeasibility we perform the following steps:

1. *Extract* $\mathbf{p}$*:*

   Large Tableau Simplex: Set $[\mathbf{p}] \leftarrow -([t_{(m+1)(n+1)}], \ldots, [t_{(m+1)(n+m)}])$.

   Small Tableau Simplex: (a) Compute the last row $\mathbf{t}$ of the corresponding large tableau from the condensed tableau [$\mathbf{T}$] and co-basis [$\mathbf{u}$] by: $[t_{u_i}] \leftarrow [t_{(m+1)i}]$.
   (b) Set $[\mathbf{p}] \leftarrow -([t_{(m+1)(n+1)}], \ldots, [t_{(m+1)(n+m)}])$.

   Revised Simplex: Set $[\mathbf{p}] \leftarrow ([d_{(m+1)1}], \ldots, [d_{(m+1)m}])$.

2. *Verify Certificate:*

   (a) Verify nonpositivity of $[\mathbf{p}]$: $[\boldsymbol{\alpha}] \leftarrow [\mathbf{p}] \leq [\mathbf{0}]$.

   (b) Verify nonpositivity of $[\mathbf{pA}]$: $[\boldsymbol{\beta}] \leftarrow [\mathbf{p}][\mathbf{A}] \leq \mathbf{0}$.

   (c) Verify positivity of $\mathbf{pb}$: $[\gamma] \leftarrow [\mathbf{p}][\mathbf{b}] > 0$.

   (d) Compute result:

$$\delta \leftarrow \mathsf{EQZ}\left([\bar{\gamma}] + \sum_{i=1}^{m}[\bar{\alpha}_i] + \sum_{j=1}^{n}[\bar{\beta}_j]\right),$$

   where $\bar{b} = 1 - b$ for any $b \in \{0, 1\}$.

5.3.3 Verification of Unboundedness

To prove that a linear program is unbounded, we need to find a solution $\mathbf{x}$ that has a cost-improving feasible direction $\mathbf{d}$ that is nonnegative (Theorem 3.10).

Lemma 5.15 shows how to extract such direction from $\mathbf{T}$ if simplex returns tableau $\mathbf{T}$ and basis $\mathbf{s}$ reporting that the linear program Eq. (5.2) is unbounded.

Lemma 5.15. *Consider tableau* $\mathbf{T}$ *for a given LP in standard form with respect to basis* $\mathbf{s}$ *and feasible solution* $\mathbf{x}$*. If the* i*-th cost-improving basic feasible direction* $\mathbf{d}^i$ *is nonnegative, then* $(\mathbf{x}, \mathbf{d}^i)$ *is a certificate of unboundedness with respect to the given LP, where* $\mathbf{d}^i = -(t_{1i}, \ldots, t_{mi})$*.*

Proof. From Theorem 3.10 it follows that the given LP is unbounded if there exists some cost-improving feasible direction having nonnegative entries at some feasible solution. We show that if $\mathbf{T}$ is a tableau with respect to basis $\mathbf{s}$ and feasible solution $\mathbf{x}$, where the i-th cost-improving basic feasible direction $\mathbf{d}^i$ is nonnegative, then the i-th column of $\mathbf{T}$ proves unboundedness.

Suppose that $\mathbf{T}$ is a tableau with respect to basis $\mathbf{s}$ and feasible solution $\mathbf{x}$. Suppose furthermore that the i-th cost-improving basic feasible direction $\mathbf{d}^i$ is nonnegative. Let $\mathbf{B}$ be the basis matrix and $\mathbf{A}' = \begin{pmatrix} \mathbf{A} & \mathbf{I}_m \end{pmatrix}$.

Recall that from $\mathbf{A}'(\mathbf{x} + \mathbf{d}^i) = \mathbf{b}$ it follows that $\mathbf{d}^i_{\mathbf{s}} = -\mathbf{c}_{\mathbf{s}}\mathbf{B}^{-1}\mathbf{A}_i$. By definition $d^i_i = 1$ and the remaining co-basic entries are equal to zero.

Since

$$\mathbf{T} = \begin{pmatrix} \mathbf{B}^{-1} & \mathbf{0} \\ -\mathbf{c}_{\mathbf{s}}\mathbf{B}^{-1} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{I}_m & \mathbf{b} \\ \mathbf{c} & \mathbf{0} & 0 \end{pmatrix}$$

we conclude that $\mathbf{d}^i_{\mathbf{s}} = -(t_{1i}, \ldots, t_{mi}) \geq \mathbf{0}$. □

Let tableau $[\mathbf{T}]$, basis $[\mathbf{s}]$, pivot column index $[\boldsymbol{\ell}]$, and solution $[\mathbf{x}]$ be returned by the simplex protocols. In case of small tableau simplex let $[\mathbf{u}]$ be the co-basis returned by simplex and in case of revised simplex let $[\mathbf{D}]$ be the revised tableau returned by simplex.

In conclusion, to securely extract and verify a certificate of unboundedness we perform the following steps:

1. *Extract* $\mathbf{d}$*:*

   Set $[\mathbf{d}] \leftarrow \mathbf{0}$.

   Large Tableau Simplex: Set $[\mathbf{d_s}] \leftarrow -([t_{1\ell}], \ldots, [t_{m\ell}])$ and $[d_\ell] \leftarrow 1$.

Small Tableau Simplex: Set $[\mathbf{d_s}] \leftarrow -([t_{1\ell}], \ldots, [t_{m\ell}])$ and $[d_{u_\ell}] \leftarrow 1$.

Revised Simplex: Set $[\mathbf{d_s}] \leftarrow -([\mathbf{d}_1], \ldots, [\mathbf{d}_m])[\mathbf{T}_\ell]$ and $[d_\ell] \leftarrow 1$.

2. *Verify Certificate:*

Rational Pivoting: Set $[q] \leftarrow 1$ and $[\chi] \leftarrow 1$.

Integer Pivoting: Verify positivity of $[\chi] \leftarrow [q] > 0$.

(a) Verify feasibility of primal solution: $[\boldsymbol{\alpha}] \leftarrow [\mathbf{A}][\mathbf{x}] \leq [q][\mathbf{b}]$, and $[\boldsymbol{\alpha}'] \leftarrow [\mathbf{x}] \geq \mathbf{0}$.

(b) Verify $\mathbf{d}$ is cost-improving: $[\gamma] \leftarrow [\mathbf{c}(\mathbf{x} + \mathbf{d})] < [\mathbf{cx}]$.

(c) Verify feasibility of $[\mathbf{d}]$: $[\boldsymbol{\beta}] \leftarrow [\mathbf{Ad}] = \mathbf{0}$. (Note that if $\mathbf{d} \geq \mathbf{0}$ and $\mathbf{d}$ is a valid direction then $\mathbf{d}$ is feasible direction.)

(d) Verify nonnegativity of $[\mathbf{d}]$: $[\boldsymbol{\beta}'] \leftarrow [\mathbf{d}] \geq \mathbf{0}$.

(e) Compute result:

$$\delta \leftarrow \mathsf{EQZ}\left([\bar{\chi}] + [\bar{\gamma}] + \sum_{j=1}^{n}\left([\bar{\beta}_j] + [\bar{\beta'}_j] + [\bar{\alpha'}_j]\right) + \sum_{i=1}^{m}[\bar{\alpha}_i]\right),$$

where $\bar{b} = 1 - b$ for any $b \in \{0, 1\}$.

5.4 Performance Comparison

This section summarizes the security properties and performance of the secure simplex variants. We evaluate on a high level the differences by the tableau representation, pivoting rule, number representation, and the simplex initialization algorithm. The analysis in this section will be theoretical. In Chapter 8 we will present a brief analysis based on practical experience.

There is no best choice of the simplex variant in general; it depends on the problem instance what variant of the simplex algorithm will be most efficient. The desired security properties may restrict applicability of some variants as well as the desired precision of the results.

In this section we consider solving a linear program in standard form Eq. (5.2) using n to denote the number of variables and m the number of constraints. We let the numbers for the simplex variants with integer pivoting (IP) be represented by $\mathbb{Z}_{\langle k_I \rangle}$ and the numbers for the simplex variants with rational pivoting (RP) by $\mathbb{Q}_{\langle k_R, f \rangle}$.

Secure Simplex Iterations

Table 5.1 shows the efficiency of each variant counting the number of secure comparisons, secure multiplications, and interactive rounds. We assume that the constant round protocol for the secure comparisons is applied, i.e., Protocol 4.28 using Protocol 4.22.

Overall, large tableau simplex has worst performance. The small tableau simplex algorithm has best round complexity and the least number of comparisons. The revised simplex algorithm has the least number of multiplications if $n \gg m$. However, we showed in Chapter 4 that each comparison gate, cf. Protocol 4.28, is equivalent to evaluating $O(k)$ secure multiplications where k denotes the bit size of the numbers to be compared.

IP/RP	pivot rule	tableau	comparison	multiplication	round
IP	Dantzig	LT	$n+3m-1$	$O(m(n+m))$	$O(\log(m(n+m)))$
		ST	$n+2m-1$	$O(nm)$	$O(\log(nm))$
		RS	$n+3m-1$	$O(m^2)$	$O(\log(m(n+m)))$
IP	Bland	LT	$n+5m-3$	$O(m(n+m))$	$O(\log m)$
		ST	$n+4m-3$	$O(nm)$	$O(\log m)$
		RS	$n+5m-3$	$O(m^2)$	$O(\log m)$
RP	Dantzig	LT	$n+3m-1$	$O(f(n+m))$	$O(\log(k_R m(n+m)))$
		ST	$n+2m-1$	$O(fnm)$	$O(\log(k_R nm))$
		RS	$n+3m-1$	$O(fm^2)$	$O(\log(k_R m(n+m)))$
RP	Bland	LT	$n+5m-3$	$O(fm(n+m))$	$O(\log(k_R m))$
		ST	$n+4m-3$	$O(fnm)$	$O(\log(k_R m))$
		RS	$n+5m-3$	$O(fm^2)$	$O(\log(k_R m))$

Table 5.1: Performance overview per iteration

It follows that the revised simplex has better communication complexity than the small tableau simplex if $n \gg k+m$.

In all cases, the simplex variants using Blands pivoting rule require $2m-2$ additional secure comparisons, but less rounds compared to the simplex variants using Dantzig's pivoting rule. On the other hand, Bland's pivoting rule ensures termination as opposed to Dantzig's pivoting rule.

Lastly, the simplex variants with integer pivoting require less rounds and less multiplications than the simplex variants with rational pivoting due to the expensive division protocol. However, if $k_I \gg k_R$, then the difference in the complexities of the secure comparisons will outweigh the additional costs for the divisions. Hence the simplex variants with rational pivoting will be more efficient.

A major drawback of simplex with rational pivoting is that due to truncation, the solution will not be exact. Another consequence of rounding errors may cause the simplex algorithm to become unstable. This typically happens when a value that is very close to zero is assigned to be the next pivot element. It follows that division leads to a large number that cannot be represented by $\mathbb{Q}_{k_R,f}$ anymore.

Simplex Initialization

Initializing the simplex algorithm is done by executing the simplex iterations on an artificial LP that has, compared to the given LP, the same amount of constraints but additional variables. These artificial variables would imply that the tableaus for the artificial LP have more columns than the tableaus for the given LP, but we showed that this is not necessary for LT and RS.

The two-phase simplex and the big-M method are essentially the same. The difference is that the big-M method solves an artificial LP with the property that the result of phase I is an optimum to the given LP if it is feasible. Since only numbers in the last row of the tableaus will depend on M, it will be more efficient to split the last row into two parts, instead of doubling the size of all numbers in the tableau, cf. Remark 3.50.

It follows that the performance differences of the iterations between the simplex initial-

Algorithm	artificial var.	tableau	comparison
Two-phase	m	LT	$n+m$
		ST	$n+m$
		RS	$n+m$
two-phase	1	LT	$n+m$
		ST	$n+1$
		RS	$n+m$
Big-M	m	LT	$2n+2m$
		ST	$2n+2m$
		RS	$2n+2m$
Big-M	1	LT	$2n+2m$
		ST	$2n+2$
		RS	$2n+2m$

Table 5.2: Comparisons required for column selection in phase I

ization algorithms are dominated by the comparisons required for the column selection. Table 5.2 shows the number of comparisons required for the column selection.

Overall, the two-phase simplex with one artificial variable based on ST has the best performance and the big-M method based on m variables has the worst performance. However, the big-M method reveals only the total number of iterations, while the two-phase simplex reveals the number of iterations it requires for both phases.

By construction, the given LP will be in canonical form. Therefore, the two-phase simplex and the big-M method with one artificial variable are easily initialized. When the given LP is not canonical, then the initialization may be equivalent to securely computing the rank of a matrix, which may outweigh the efficiency gain.

In conclusion, selecting the most suitable variant to securely solve a given linear program depends on many aspects. In Chapter 8, we discuss heuristic choices made in tests by SecureSCM [Sec08].

Chapter 6

Universal Verifiability

In this chapter we address the issue in secure computation that the protocols break down if none of the parties is honest. The security properties of the protocols are formulated in terms of an adversary limited to corrupting proper subsets of parties. Nothing is guaranteed in case an adversary is capable of corrupting parties beyond the stated limit, and, in particular, if the adversary is capable of corrupting all parties.

While this seems reasonable, the lack of any security guarantee for secure computations performed by corrupt parties only is actually unacceptable in many cases. Consider, for instance, cryptographic schemes for electronic voting. In such voting schemes the election result is determined by a secure computation executed by so-called talliers. While it cannot be prevented that the talliers learn all votes if they all collude, it would be unacceptable if the talliers are also able to announce a fabricated election result without anyone noticing. Indeed, many cryptographic voting schemes guarantee that false election results will never be accepted. Similarly, in secure cloud computing, where computations are outsourced, one generally requires a means to check the validity of the computed results.

In this chapter we construct protocols for which the correctness of the output is guaranteed even if all parties in the protocol collude. More precisely, we will require that correctness of the output can be verified, so that a false result will never be accepted even if all parties collude. Protocols satisfying this property will be called *universally verifiable*, using the same terminology as commonly used in voting schemes. We will introduce universal verifiability for secure computation based on threshold homomorphic cryptosystems [CDN01].

Intuitively, a protocol is universally verifiable if all messages sent by each party are accompanied by a noninteractive zero-knowledge proof of correctness. This implies that the protocol of [CDN01] will be universally verifiable if all the zero-knowledge proofs are made noninteractive by, for example, the Fiat-Shamir transform. However, applying the Fiat-Shamir transform to the zero-knowledge proofs leads to complications in the security proof of the overall protocol of [CDN01]. We will show how to transform the zero-knowledge proofs to achieve universal verifiability so that the original security proof still applies.

6.1 Universally Verifiable Secure Computation

In [GMW87] the basic observation was made that any protocol for secure multiparty computation that is secure against passive adversaries can be made secure against active adversaries by requiring all parties to prove correctness for each message they sent. Note, however, that even if all these proofs pass verification, correctness cannot be guaranteed in

general. For instance, if interactive zero-knowledge proofs are used, an adversary controlling all parties is able to generate protocol runs with incorrect output, using zero-knowledge simulators for the respective proofs.

The protocol of [CDN01] follows the same structure as [GMW87]. We will show how to transform the interactive zero-knowledge proofs into noninteractive proofs, so that correctness can be guaranteed and security is maintained.

We will restrict our treatment of universal verifiability to admissible protocols defined as follows.

Definition 6.1. *A protocol π is called* admissible *if every publicly transmitted bit is either part of a zero-knowledge proof or not.*

The protocol of [CDN01] can easily be seen to be admissible. We will now define what we mean by universal verifiability in Definition 6.3. But first we define the tools required in the following definition.

Definition 6.2. *Let π be an admissible n-party protocol that securely evaluates f. Let $\mathsf{tr}_\pi(\mathbf{x}, \mathbf{r})$ denote the collection of all publicly transmitted messages during an execution of π on input $\mathbf{x}$ using randomness $\mathbf{r}$. By $\mathsf{tr}_{\pi,\sigma}(\mathbf{x}, \mathbf{r})$ we denote the collection of all transmitted bits that belong to a zero-knowledge proof and by $\mathsf{tr}_{\pi,f}(\mathbf{x}, \mathbf{r}) = \mathsf{tr}_\pi(\mathbf{x}, \mathbf{r}) \backslash \mathsf{tr}_{\pi,\sigma}(\mathbf{x}, \mathbf{r})$ we denote the collection of remaining transmitted bits. Finally, let $\mathsf{tr}^h_\pi(\mathbf{x}, \mathbf{r})$ denote the collection of all publicly transmitted bits during an execution of π on input $\mathbf{x}$ using randomness $\mathbf{r}$ in the absence of any adversary.*

Definition 6.3. *Let π be an admissible n-party protocol that securely evaluates f and let $\mathbf{y} = \pi(\mathbf{x}, \mathbf{r})$ denote the public output of π on input $\mathbf{x}$ and randomness $\mathbf{r}$. Then, we say that π is* universally verifiable *if $\mathsf{tr}_{\pi,\sigma}(\mathbf{x}, \mathbf{r})$ is a noninteractive zero-knowledge proof of the fact that $\mathbf{y} = f(\mathbf{x})$.*

Evidently, following the observation of [GMW87], it holds that any protocol for secure multiparty computation that is secure against passive adversaries can be made universally verifiable by requiring that all parties send a noninteractive zero-knowledge proof of correctness for each message they sent.

Lemma 6.4. *Let π be an admissible n-party protocol that securely evaluates f and let π' be the n-party protocol derived from π by removing all zero-knowledge proofs. If $\mathsf{tr}_{\pi,\sigma}(\mathbf{x}, \mathbf{r})$ is a noninteractive zero-knowledge proof of the fact that $\mathsf{tr}_{\pi,f}(\mathbf{x}, \mathbf{r}) \in L_{R_f}$, where*

$$R_f = \{(\mathsf{tr}; \mathbf{x}, \mathbf{r}) | \mathsf{tr} = \mathsf{tr}^h_{\pi'}(\mathbf{x}, \mathbf{r})\},$$

then π is universally verifiable.

In [CDN01] Σ-protocols are used to enforce honest behavior of the parties. It follows by Lemma 6.4 that the protocol of [CDN01] will be universally verifiable if all multiparty Σ-protocols are noninteractive zero-knowledge proofs. The following sections show how to transform the interactive multiparty Σ-protocols into noninteractive zero-knowledge proofs in the random oracle model.

6.1.1 Multiparty Σ-protocols

First, we introduce a multiparty proof of knowledge of [CDN01], called multiparty Σ-protocols, and we prove that these are indeed Σ-protocols, if the collection of all parties acting as a verifier is considered as one entity. With this result we can turn these proofs into noninteractive zero-knowledge proofs using the techniques described in Section 2.2.2.

Second, we present the simulator S^{mpc} for the multiparty Σ-protocols that is used by [CDN01] to prove security of the overall protocol. Based on this simulator we are able to easily give a simulator of our noninteractive multiparty Σ-protocols that can be used to prove security of the overall protocol of [CDN01].

The Multiparty Σ-protocol

In a multiparty Σ-protocol there are n parties, where one party plays the role of a prover and the remaining parties play the role of the verifier.

Consider a two-party Σ-protocol Σ_1. Let A, B and C be the p.p.t. algorithms used in Σ_1, see Section 2.2.1. The transformation from [CDN01] in which a Σ-protocol Σ_1 for an NP-relation R is transformed into a multiparty Σ-protocol for relation R is given by Protocol 6.1. The number τ is chosen such that if a majority of the parties is honest, then at least κ bits are uniformly random for security parameter κ. We refer to [CDN01] for the details on the choice of τ.

Protocol 6.1: $\mathbf{v} \leftarrow \Sigma^{\mathsf{mpc}}(\Sigma_1, \mathbf{x}, \mathbf{w}, k)$

foreach party $i = 1, \ldots, n$ **do**
  pick $u_i \in_R \{0,1\}^k$;
  pick $z_i \in_R \{0,1\}^k$;
  $a_i \leftarrow A(x_i, w_i, u_i)$;
  $e_i \leftarrow b_i(a_i, z_i)$;
  broadcast e_i;
foreach party $i = 1, \ldots, n$ **do**
  pick $c_i \in_R \{0,1\}^\tau$;
  broadcast c_i;
$c \leftarrow c_1 || \ldots || c_n$;
foreach party $i = 1, \ldots, n$ **do**
  $r_i \leftarrow B(x_i, w_i, u_i, c)$;
  broadcast (a_i, z_i, r_i);
foreach $i = 1, \ldots, n$ **do**
  $v_i \leftarrow C(x_i, a_i, c, r_i) = 1 \wedge e_i = b_i(a_i, z_i)$;
return $\mathbf{v}$;

Lemma 6.5. *If Σ_1 is a Σ-protocol and b_i are perfectly hiding commitment functions for $i = 1, \ldots, n$, then the n-party Σ-protocol 6.1 is complete, special sound, and special honest-verifier zero-knowledge.*

Proof. For completeness observe that if the parties follow the protocol, then $e_i = b_i(a_i, z_i)$. Hence completeness follows by completeness of Σ_1.

For special soundness, suppose that for all $i = 1, \ldots, n$ a common input x_i and two accepting conversations (e_i, a_i, z_i, c, r_i) and $(e_i, a_i', z_i', c', r_i')$, where $c \neq c'$, are given. Then $a_i = a_i'$ by the binding property of b_i.

It follows that (a_i, c, r_i) and (a_i, c', r_i') are accepting conversations for Σ_1. Hence, we can run the extractor E for Σ_1 to compute a witness w_i such that $(x_i; w_i) \in R$.

Lastly, given c and x_i, let S_1 be the simulator for Σ_1 that provides accepting triples (a, c, r) that are indistinguishable from real conversations if $x_i \in L_R$. The simulator S is then defined as follows: run S_1 several times to get (a_i, c, r_i) for all i and compute commitment $e_i = b_i(a_i, z_i)$ by generating a random z_i.

Since Σ_1 is special honest-verifier zero-knowledge and b_i is perfectly hiding it follows that the simulated transcripts (e_i, a_i, c, r_i) are perfectly indistinguishable from real conversations. □

Observe that Σ_1^{mpc} is a proof of knowledge with knowledge error $2^{-\kappa}$ if a majority of the parties is honest. If all parties are corrupt, then it is certainly not a proof of knowledge. Indeed, the collusion of all parties could first generate the random c and then simulate accepting transcripts using the simulator for Σ_1.

The Simulator

The security proof of [CDN01] requires an ideal world simulator S_1^{mpc} for Σ_1^{mpc} with the following properties.

- S_1^{mpc} runs in expected polynomial time and provides views that are perfectly indistinguishable from the outputs of a real protocol execution using trapdoors of the commitment scheme for the honest parties.
- S_1^{mpc} outputs in addition valid witnesses for each corrupted parties providing accepting conversations.

Let t_i be the trapdoor with respect to b_i for party P_i. Let $\mathcal{A}$ denote the adversary. Then, it is proven in [CDN01, full version, pp.15–17] that the following algorithm for S_1^{mpc} satisfies both properties:

1. For each honest P_i, use trapdoor t_i to compute a commitment e_i that can be opened arbitrarily and give e_i to $\mathcal{A}$. For each corrupted P_i get e_i from $\mathcal{A}$.
2. For each honest P_i generate τ random bits c_i and send those to $\mathcal{A}$ and receive c_i from $\mathcal{A}$ on behalf of all corrupted P_i. Compute $c = c_1 || \ldots || c_n$.
3. For each party P_i, in which $\mathcal{A}$ may choose the order, do:
   - If P_i is honest, run the simulator for Σ_1 on input c to get an accepting conversation a_i, c, r_i. Use the trapdoor t_i to compute z_i so that $e_i = b_i(a_i, z_i)$. Send (z_i, a_i, r_i) to $\mathcal{A}$.
   - If P_i is corrupted, then receive (z_i, a_i, r_i) from $\mathcal{A}$.
4. For each corrupted party P_i for which (z_i, a_i, r_i) is accepting then do:
   (a) Rewind $\mathcal{A}$ to just before the state where the challenge is computed.

(b) Generate fresh random values on behalf of the honest parties. This results in a new challenge c'.

(c) Generate accepting proofs on behalf of the honest parties using the challenge c' and receive (z_i', a_i', r_i') from the adversary. If the proof is not accepting return to 4(a).

(d) If $a_i' \neq a_i$ compute a valid witness w_i using the extractor from Σ_1, else stop and return $(e_i, a_i, a_i', z_i, z_i')$ as a break of the commitment scheme.

6.1.2 Non-interactive Multiparty Σ-proofs

This section shows how to transform the multiparty Σ-protocols from [CDN01] into a noninteractive zero-knowledge proof. We discuss two alternatives:

(i) using the Fiat-Shamir transform,

(ii) using the Generalized Fiat-Shamir transform.

Observe that both heuristics act on a Σ-protocol between two parties. In Figure 6.1 we present a Σ-protocol that is derived from Protocol 6.1, where all provers are considered as one party P communicating with some verifier V. Clearly by symmetry it follows from Lemma 6.5 that the protocol of Figure 6.1 satisfies all properties of a Σ-protocol.

Prover	**Common input:**	**Verifier**
for $i = 1, \ldots, n$ **do:**		
$u_i \in_R \{0,1\}^k$		
$z_i \in_R \{0,1\}^k$		
$a_i \leftarrow A(x_i, w_i, u_i)$		
$e_i \leftarrow b_i(a_i, z_i)$		
	$\xrightarrow{e_1, \ldots, e_n}$	
		$c \in_R \{0,1\}^\kappa$
	$\xleftarrow{c}$	
for $i = 1, \ldots, n$ **do:**		
$r_i \leftarrow B(x_1, w_i, u_i, c)$		
	$\xrightarrow{r_1, \ldots, r_n}$	
		for $i = 1, \ldots, n$ **do:**
		$C(x_i, a_i, c, r_i) \stackrel{?}{=} 1$

Figure 6.1: Σ-protocol for relation R using perfectly hiding commitment functions b_i

We can apply the Fiat-Shamir transform and the generalized Fiat-Shamir transform to the protocol of Figure 6.1, turning it into a noninteractive zero-knowledge proof, cf. Theorem 2.8 and Theorem 2.9 respectively. Note that noninteractive means here that P generates a proof on its own without interacting with V. However, in this context $P = \{P_1, \ldots, P_n\}$ and interaction between the parties $P_1, \ldots, P_n$ may be necessary to generate a proof.

Consider the protocol of [CDN01]. The protocol consists of multiple interactive rounds, where in each round one multiparty Σ-proof is executed to prove knowledge and correctness of all transmitted messages in that particular round. Note that the security proofs of the Fiat-Shamir transform and the generalized Fiat-Shamir transform are in the stand-alone situation. To avoid complications due to composition of the protocols, in each round we apply one noninteractive proof and we use a different random oracle.

Let $\mathcal{H}_i : \{0,1\}^* \to \{0,1\}^\kappa$ be a random oracle for round i. Note that given a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\kappa$ one can define $\mathcal{H}_i(m) := \mathcal{H}(i,m)$. Protocol 6.2 shows how to apply both heuristics to the protocol of Figure 6.1 and, therefore, shows how to transform the multiparty Σ-protocols of [CDN01] into noninteractive zero-knowledge proofs.

Protocol 6.2: $\boldsymbol{\sigma} \leftarrow \Sigma^{\mathsf{mpc}}_{\mathsf{VAR}}(\Sigma_1, \mathbf{x}, \mathbf{w}, k, rnd)$

Input: $\Sigma_1, \mathbf{x}, \mathbf{w}, k, rnd$
Output: $\boldsymbol{\sigma}$

1 **foreach party** $i = 1, \ldots, n$ **do**
2 **pick** $u_i \in_R \{0,1\}^k$;
3 **pick** $z_i \in_R \{0,1\}^k$;
4 $a_i \leftarrow A(x_i, w_i, u_i)$;
5 $e_i \leftarrow b_i(a_i, z_i)$;
6 **broadcast** (e_i);

$\mathsf{VAR} = \mathsf{Fiat-Shamir}$
7a $c \leftarrow \mathcal{H}(rnd||e_1||\ldots||e_n)$;

$\mathsf{VAR} = \mathsf{GeneralizedFiat-Shamir}$
7b **foreach party** $i = 1, \ldots, n$ **do**
8b **pick** $c_i \in_R \{0,1\}^\tau$;
9b **broadcast** (c_i);
10b $c \leftarrow \mathcal{H}(rnd||c_1||\ldots||c_n||e_1||\ldots||e_n)$;

11 **foreach party** $i = 1, \ldots, n$ **do**
12 $r_i \leftarrow B(x_i, w_i, u_i, c)$;
13 **broadcast** (a_i, z_i, r_i);
14 $\sigma_i \leftarrow (e_i, a_i, c, z_i, r_i)$;
15 **return** $\boldsymbol{\sigma}$;

It remains to show that if the multiparty Σ-protocols of [CDN01] are replaced by Protocol 6.2 the overall protocol of [CDN01] remains secure. Let t_i denote the trapdoor of b_i. We define a simulator S_j^{mpc} for round j as follows:

1. For each honest P_i, use trapdoor t_i to compute a commitment e_i that can be opened arbitrarily and give e_i to $\mathcal{A}$. For each corrupted P_i get e_i from $\mathcal{A}$.

2′. **Fiat-Shamir:** Compute $c = \mathcal{H}(j||e_1||\ldots||e_n)$.

Generalized Fiat-Shamir: For each honest P_i generate τ random bits c_i and send those to $\mathcal{A}$ and receive c_i from $\mathcal{A}$ on behalf of all corrupted P_i. Compute $c = \mathcal{H}(j||c_1||\ldots||c_n||e_1||\ldots||e_n)$.

3. For each party P_i, in which $\mathcal{A}$ may choose the order, do:

- If P_i is honest, run the simulator for Σ_1 on input c to get an accepting conversation a_i, c, r_i. Use the trapdoor t_i to compute z_i such that $e_i = b_i(a_i, z_i)$. Send (z_i, a_i, r_i) to $\mathcal{A}$.
- If P_i is corrupted, then receive (z_i, a_i, r_i) from $\mathcal{A}$.

4′. For each corrupted party P_i for which (z_i, a_i, r_i) is accepting then do:

Fiat-Shamir: Run the extractor of the forking lemma to get an accepting conversation $(e_i, z_i', c_i', a_i', r_i')$, where $c_i' \neq c_i$.

Generalized Fiat-Shamir:

(a) Rewind the adversary to just before the state the challenge is computed.

(b) Generate fresh random values on behalf of the honest parties. This results in a new challenge c'.

(c) Generate accepting proofs on behalf of the honest parties using the challenge c' and receive (z_i', a_i', r_i') from the adversary. If the proof is not accepting return to 4(a).

If $a_i' \neq a_i$ compute a valid witness w_i using the extractor from Σ_1, else stop and return $(e_i, a_i, a_i', z_i, z_i')$ as a break of the commitment scheme.

Note that the only difference between S^{mpc} and S_j^{mpc} is the generation of the random challenge in step 2 and the extraction of the witnesses for each corrupt party providing an accepting proof in this round.

The random challenges are in both the simulated and protocol uniformly random by the random oracle. The runtime of the witness extraction is $O(1/\epsilon_i)$, where ϵ_i is the probability that the adversary provides an accepting proof on behalf of the corrupt party P_i. Since with probability ϵ_i the simulator S_j^{mpc} is going to extract a witness for party P_j the expected runtime for step 4 is $O(1)$.

It follows that the security of the overall protocol of [CDN01] is maintained.

6.2 Efficient Universally Verifiable Computation from Certificate Validation

In Section 3.1.4 we showed how to validate an optimal solution for a given linear program, or the fact that the given linear program is unbounded, or infeasible. Compared to computing a result, the validation of a result turned out to be some relatively simple computations on a *certificate of correctness*. In Chapter 5 we provided protocols for solving linear programs, extraction of a certificate, and validation of the certificate.

Linear programming is just one example of a problem where any solution can be efficiently validated. Table 6.1 provides other examples. Actually observe that any function solving an NP problem can be efficiently validated.

Observe that if the protocols for verification of the certificate are universally verifiable, then the overall protocol is universally verifiable. Indeed, if anyone can check that the validation is correct, then correctness of the outputs can be verified by anyone using the result of the validation.

More precisely, the following lemma shows that if there exists a validating function g for some function f, then a protocol that evaluates f and g successively, where the encryptions

Problem	Input	Output	Verification
n-th root	x	$y = \sqrt[n]{x}$	$y^n \stackrel{?}{=} x$
field inverse	x	$y = \frac{1}{x}$	$yx \stackrel{?}{=} 1$
division with remainder ($y = \alpha x + \beta$)	(x, y)	(α, β)	$y \stackrel{?}{=} \alpha x + \beta$ $0 \leq \beta \stackrel{?}{<} x$
roots of f	f	y	$f(y) \stackrel{?}{=} 0$
extended gcd of x and y	(x, y)	(α, β, d)	$\alpha x + \beta y \stackrel{?}{=} d$ $d \mid x$ $d \mid y$ $d > 0$
bit decomposition	x	$x_0, \ldots, x_\ell$	$\sum_{i=0}^{\ell} 2^i x_i \stackrel{?}{=} x$
matrix inverse	A	$B = A^{-1}$	$AB \stackrel{?}{=} I$
eigenvalue	A	(λ, v)	$Av \stackrel{?}{=} \lambda v$

Table 6.1: Examples of problems where the solution can be efficiently validated

of the inputs g are public before g is evaluated and the evaluation of g is universal verifiable, is universal verifiable. We denote by $[\![x]\!]$ a probabilistic homomorphic encryption of x.

Lemma 6.6. *Suppose that π_f is a protocol that securely evaluates function f. Suppose that π_f returns (y, c) on input x. Suppose further that c is a certificate of the fact that $y = f(x)$, with validating function g (see Definition 3.22). If π_g universally verifiably evaluates g, then the following protocol universally verifiably evaluates f:*

1. *The parties execute protocol π_f.*
2. *All parties broadcast an encryption of their inputs $[\![x]\!]$ and outputs $[\![y]\!]$ and $[\![c]\!]$.*
3. *The parties execute π_g on input $[\![x]\!]$, $[\![y]\!]$, and $[\![c]\!]$.*
4. *The parties accept y as the correct solution if all proofs in π_g are accepting and if the output of π_g is equal to 1.*

Proof. Let b be the output of π_g. Since π_g is universally verifiable and is executed on inputs $[\![x]\!]$, $[\![y]\!]$, and $[\![c]\!]$, Definition 6.3 implies that $\mathsf{tr}_{\pi_g,\sigma}$ is a noninteractive zero-knowledge proof of the fact that $b = g(x, y, c)$.

Since g is a validating function, see Definition 3.22, $b = 1$ if and only if $y = f(x)$. Hence $\mathsf{tr}_{\pi_g,\sigma}$ is a noninteractive proof of the fact that $y = f(x)$. □

Next, we will show how to apply this lemma to build protocols that solve linear programs universally verifiably.

Universally Verifiable Linear Programming

We will show universally verifiable protocols solving linear programs without enforcing honest behavior in each step of the computation using Lemma 6.6. To give a precise

example, we apply [CDN01] with Paillier's homomorphic cryptosystem. We note that for other cryptosystems similar protocols can be applied.

From Lemma 6.6 it follows that the following protocol is universally verifiable, where only the protocol of [CDN01] with one of the transformations of previous section is applied to the certificate validation part, cf. Section 5.3. Suppose that parties wish to solve a linear program with coefficients $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{c}$.

1. All parties execute one of the protocols described in Chapter 3.
2. Upon reception of the solution $\mathsf{pred} \in \{\mathsf{Optimal}, \mathsf{UnboundedLP}, \mathsf{InfeasibleLP}\}$, the parties run the protocols of Section 5.3 to extract a certificate $[\mathbf{v}]$.
3. The parties convert $[\mathbf{v}]$ into homomorphic encryptions $[\![\mathbf{v}]\!]$ for the protocol of [CDN01].
4. All parties P_i broadcast encryptions of their inputs $[\![\mathbf{A}]\!]$, $[\![\mathbf{b}]\!]$ and $[\![\mathbf{c}]\!]$, certificate $[\![\mathbf{v}]\!]$ and solution pred.
5. The parties use [CDN01] and one of the transformations of the previous section to universally verifiably validate the certificate of pred on input $[\![\mathbf{A}]\!]$, $[\![\mathbf{b}]\!]$, $[\![\mathbf{c}]\!]$, and $[\![\mathbf{v}]\!]$.

It remains to provide protocols for transforming Shamir shares into homomorphic encryptions. The next section provides protocols that securely convert Shamir shares into Paillier encryptions.

Conversion of Shamir Shares into Paillier Encryptions

To convert Shamir shares into Paillier encryptions, we use the protocol from Algesheimer et al. [ACS02] that converts additive shares over any prime field into additive shares over the integers. They prove that their protocol is statistically secure against any static t-limited adversary in the model of [Can00].

Let $[x]^A$ be an additive sharing over $\mathbb{Z}_q$ of $x \in \mathbb{Z}_{\langle k \rangle}$, Protocol 6.3 shows how to convert $[x]^A$ into $(z_1, \ldots, z_n)$ the additive sharing of x over $\mathbb{Z}$. For the simulator we refer to [ACS02].

To convert Shamir shares $[x]$ into Paillier encryptions we proceed as follows:

1. The parties compute the reconstruction vector $\lambda_1, \ldots, \lambda_n$.
2. Each party P_i computes his additive share of x by $[x]_i^A = \lambda_i [x]_i$.
3. The parties run Protocol 6.3 resulting in each party having z_i an additive share of x over $\mathbb{Z}_{\langle k+\kappa+\log n \rangle}$.
4. The parties broadcast $[\![z_i]\!]$ and the result is computes as $[\![z]\!] = \prod_{i=1}^{n} [\![z_i]\!]$.

Protocol 6.3: $(z_1, \ldots, z_n) \leftarrow \mathsf{ConvertZQ2Z}([x]^A, q)$

$t \leftarrow \kappa + k + 2$;
foreach party $i = 1, \ldots, n$ **do**
  $a_i \leftarrow \left\lfloor \frac{[x]_i^A}{2^t} \right\rfloor$;
  broadcast a_i;
$\ell \leftarrow \left\lceil \frac{2^t \sum_{i=1}^n a_i}{q} \right\rfloor$;
$\alpha \leftarrow -\mathrm{sgn}(\ell)$;
foreach party $i = 1, \ldots, n$ **do**
  $[b_i]^A \leftarrow \mathsf{AShare}(0, \mathbb{Z}_{\langle \log(q)+\kappa \rangle})$;
$[0]^A = \sum_{i=1}^n [b_i]^A$;
foreach party $i = 1, \ldots, n$ **do**
  if $i \leq |\ell|$ **then**
    $z_i \leftarrow [x]_i^A + [0]_i^A + \alpha q$;
  else
    $z_i \leftarrow [x]_i^A + [0]_i^A$;

Protocol 6.4: $[\![z]\!] \leftarrow \mathsf{ShamirToPaillier}([x], q, N)$

foreach party $i = 1 \ldots, n$ **do**
  $[x]_i^A \leftarrow [x]_i \prod_{j=1, j\neq i}^n \frac{-j}{i-j}$;
$(z_1, \ldots, z_n) \leftarrow \mathsf{ConvertZQ2Z}([x]^A, q)$;
foreach party $i = 1, \ldots, n$ **do**
  pick $r_i \in_R \mathbb{Z}_N$;
  $[\![z_i]\!] \leftarrow (1+N)^{z_i} r_i^N$;
  broadcast $[\![z_i]\!]$;
$[\![z]\!] \leftarrow \prod_{i=1}^n [\![z_i]\!]$;
return $[\![z]\!]$

Chapter 7

Restricted Shuffling

A basic primitive in many secure multiparty computation protocols is shuffling. A *shuffle* is an operation on a list of encrypted messages that produces a new list of encrypted messages with the property that after decryption both lists are identical except for the order of the messages.

Secure shuffling is applied in, for example, the protocol of [LA06] for linear programming. To hide the pivot element the rows and columns of the tableau are shuffled. The Mix and Match protocol from [JJ00] involves shuffling of truth tables of boolean gates to hide information that can be extracted from the position of the match. Furthermore, in many protocols for secure integer comparison [BK04, ABFL06, GSV07, RT09a], shuffling is applied to hide the position of a certain specific value in the list.

This chapter discusses proofs of *restricted shuffles*. Given two lists of (homomorphic) encrypted messages $[\![\mathbf{x}]\!]$ and $[\![\mathbf{y}]\!]$, one proves in zero-knowledge that $y_i = x_{\pi(i)}$ for each entry, where π is a permutation that satisfies some properties.

In [HSSV09] the design of zero-knowledge protocols for rotation is discussed. This result can be applied to design zero-knowledge protocols with respect to any of the following restricted shuffles: rotation, affine transformation and Möbius transform [VB10]. The idea is to decompose any such restricted shuffle into multiple successive rotations to which the approach of [HSSV09] is applied.

More general constructions are considered in [TW10, Kel11], providing zero-knowledge protocols for broad classes of restricted shuffles. We show how to instantiate the protocols of [TW10] so that the resulting protocol is a zero-knowledge protocol with respect to any of the following restricted shuffles: rotation, affine transformation and Möbius transformation.

This chapter is organized as follows: first we will discuss the protocol from [TW10] and the successive sections will show how to instantiate the protocol to prove correctness of a rotation, affine transformation, and Möbius Transformation.

7.1 Proofs of Restricted Shuffles

This section introduces the main ideas behind the protocol of [TW10].

Suppose that given two lists of homomorphic encryptions $[\![x_1]\!], \ldots, [\![x_k]\!]$ and $[\![y_1]\!], \ldots, [\![y_k]\!]$ one wishes to prove in zero-knowledge that $y_i = x_{\pi(i)}$, where π is a permutation from some set of permutations.

The protocol of Terelius and Wikström is a 5 move zero-knowledge proof of knowledge in which the prover proves knowledge of a matrix M, such that $\mathbf{y} = M\mathbf{x}$. The matrix M is a *permutation matrix* of some permutation $\pi \in \mathcal{P} \subseteq S_k$, where $\mathcal{P}$ is a *permutation group*

and S_k the group of all permutations acting on k elements.

Let $d \geq 1$ be some positive integer. The idea is to apply a polynomial $p_{\mathcal{P}} : \mathbb{Z}_q^{kd} \to \mathbb{Z}_q$, that is invariant under the permutation group $\mathcal{P}$, i.e., $\pi \in \mathcal{P}$ if and only if $p(\mathbf{z}_1, \dots \mathbf{z}_d) = p(\mathbf{z}'_1, \dots, \mathbf{z}'_d)$, where $z'_{ji} = z_{j\pi(i)}$ for all $i = 1, \dots k$ and all $j = 1, \dots d$.

The protocol of [TW10] is as follows. Suppose that a permutation group $\mathcal{P}$, a polynomial $p_{\mathcal{P}}$, and the inputs $[\![\mathbf{x}]\!]$ and $[\![\mathbf{y}]\!]$ are given. The prover P proves that $\mathbf{x}$ is permuted into $\mathbf{y}$ according to a permutation $\pi \in \mathcal{P}$ by

(i) proving knowledge of a matrix $\mathbf{M} \in \mathbf{Z}_q^{k \times k}$ such that $\mathbf{y} = \mathbf{M}\mathbf{x}$, and

(ii) proving that $\mathbf{M}$ is such that $p_{\mathcal{P}}(\mathbf{M}\mathbf{e}_1, \dots, \mathbf{M}\mathbf{e}_d) = p_{\mathcal{P}}(\mathbf{e}_1, \dots, \mathbf{e}_d)$, where $\mathbf{e}_i \in_R \mathbb{Z}_q^k$ are uniformly random challenges from the verifier.

This chapter shows given $\mathcal{P}$ how to find a polynomial p that is invariant under $\mathcal{P}$. The results are based on (hyper)graphs.

Let $G = (V, A)$ denote a directed hypergraph on vertices V and arcs $A \subseteq 2^V$, where 2^V denotes the power set of V. The hypergraph G is called u-uniform, with $u \geq 1$, if every arc in A contains exactly u vertices. If $u = 2$, then G is simply called a directed graph.

Definition 7.1. *Let $G = (V, A)$ be a u-uniform hypergraph on n vertices. Permutation $\pi \in S_n$ is an* automorphism *of G if and only if*

$$a = (v_1, \dots, v_u) \in A \iff (\pi(v_1), \dots, \pi(v_u)) \in A$$

for all $a \in A$.

It is well-known that if $\pi \in S_n$ is an automorphism of G then π^{-1} is also an automorphism of G and, moreover, if $\sigma \in S_n$ is an automorphism of G then $\pi \circ \sigma$ is also an automorphism of G, where $\pi \circ \sigma(x) = \pi(\sigma(x))$. Hence the collection of all automorphisms of G (say $\mathcal{P}$) and the operation $\circ$ form a group. This group is called the *automorphism group* of G.

In [TW10] it is observed that the automorphism group of any graph G is the permutation group $\mathcal{P}$ if and only if the polynomial

$$p(\mathbf{v}_1, \dots, \mathbf{v}_u) = \sum_{(i_1, \dots, i_u) \in A} \prod_{i=1}^{u} v_{i_u}. \tag{7.1}$$

is invariant under $\mathcal{P}$.

To instantiate the protocol of [TW10] given permutation group $\mathcal{P}$, we wish to find a polynomial given by Eq. (7.1) that is invariant under $\mathcal{P}$. Recall that the invariance test by [TW10] is done by computing

$$p(\mathbf{v}_1, \dots, \mathbf{v}_u) \stackrel{?}{=} p(\mathbf{M}\mathbf{v}_1, \dots, \mathbf{M}\mathbf{v}_u),$$

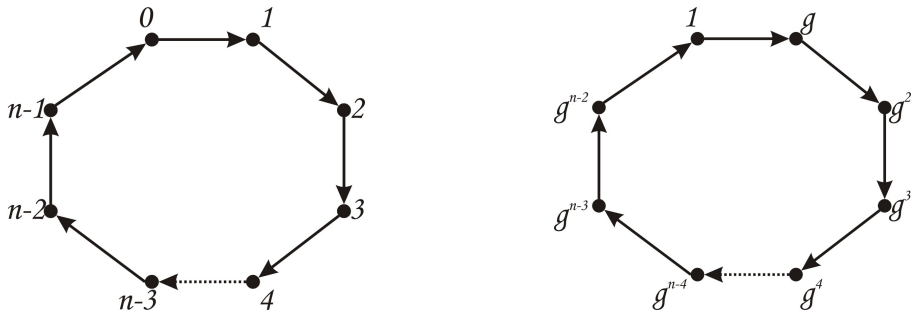
where M is a $k \times k$ permutation matrix. Furthermore, M is used to prove the fact that the secret inputs $\mathbf{x}$ and $\mathbf{y}$ of the protocol satisfy $\mathbf{y} = \mathbf{M}\mathbf{x}$.

Given permutation group $\mathcal{P}$, we can find such polynomial if we can find a hypergraph $G_{\mathcal{P}} = (V_{\mathcal{P}}, A_{\mathcal{P}})$ that satisfies $|V_{\mathcal{P}}| = k$, and

$$\pi \in \mathcal{P} \iff [(v_1, \dots, v_u) \in A_{\mathcal{P}} \iff (\pi(v_1), \dots, \pi(v_u)) \in A_{\mathcal{P}}] \tag{7.2}$$

for all $v_1, \dots, v_u \in V_{\mathcal{P}}$.

The following sections discusses (hyper)graphs, where the automorphism group is either the group of rotations, affine transformations, or Möbius transformations.



(a) Rotation Invariant Graph (b) Rescaling Invariant Graph

Figure 7.1: Cyclic graphs

Remark 7.2 (Cayley Graphs). In 1939 Frucht proved that for any finite permutation group $\mathcal{P}$ there exists a finite undirected graph that has $\mathcal{P}$ as the group of automorphisms [Fru39]. The proof is based on the *Cayley graph* corresponding to $\mathcal{P}$. The nice property of the Cayley graph of $\mathcal{P}$ is that its vertices are identified by the elements of $\mathcal{P}$ and the group of automorphisms is (isomorphic) to $\mathcal{P}$.

Consider two lists $\mathbf{x}$ and $\mathbf{y}$ that are indexed by elements from $\mathcal{P}$. To prove in zero-knowledge that for each $\pi \in \mathcal{P}$ the entry y_π satisfies $y_\pi = x_{\pi \circ \pi'}$ for some $\pi' \in \mathcal{P}$ one could apply [TW10] by constructing the Cayley Graph for $\mathcal{P}$.

Note that for the Cayley graph the order of the automorphism group is equal to the number of vertices. Frucht's theorem follows by adding vertices and edges to the Cayley graph in such a way that the automorphism group is preserved after removing all colors and directions. Hence, if we wish to find a graph on a number of vertices that is smaller than the order of the group of automorphisms we cannot simply build a Cayley graph.

This chapter discusses two cases where we wish to find a graph, where the number of vertices is smaller than the order of the group of automorphisms. For example, let p be prime, we show how to find a 3-uniform hypergraph, where the elements of $\mathbb{Z}_p$ are its vertices and where the group of automorphisms is exactly the group of affine transformations. Note that the group of affine transformations on $\mathbb{Z}_p$ has order $p^2 - p$. $\diamond$

7.2 Rotation and Rescaling

Rotation

Let $\mathcal{R}$ be the set of all rotations of a list of n elements. Hence

$$\pi \in \mathcal{R} \iff \exists_{0 \leq r < n} \forall_{x \in \mathbb{Z}_n} : \pi(x) = x + r \bmod n. \tag{7.3}$$

Let $V_\mathcal{R} = \mathbb{Z}_n$ and $A_\mathcal{R} = \{(i, i+1 \bmod n) | i \in \mathbb{Z}_n\}$. The graph $G_\mathcal{R} = (V_\mathcal{R}, A_\mathcal{R})$ is depicted in Figure 7.1(a).

Theorem 7.3. *$\pi \in S_n$ is an automorphism of $G_\mathcal{R}$ if and only if $\pi \in \mathcal{R}$.*

Proof. Observe first that by rotating the vertices of the graph, the neighbors of each vertex remain the same in the same order. Hence rotation is indeed an automorphism of $G_\mathcal{R}$.

For the other direction, suppose $\pi \in S_n$ is an automorphism of $G_{\mathcal{R}}$. Then $(\pi(i), \pi(i+1)) \in A_{\mathcal{R}})$. Hence

$$\pi(i+1 \bmod n) = \pi(i) + 1 \bmod n$$

for all $i = 0, \ldots, n-1$.

Define $\pi(0) =: r$, then $0 \leq r < n$ and $\pi(1) = \pi(0) + 1 \bmod n = r + 1 \bmod n$. Next assume $\pi(i) = i + r \bmod n$ then by induction

$$\pi(i+1 \bmod n) = \pi(i) + 1 \bmod n = (i+1) + r \bmod n.$$

Hence $\pi \in \mathcal{R}$ by Eq. (7.3). □

Rescaling

Let p be a prime and let $\mathcal{S}$ be the set of the permutations on $\mathbb{Z}_p^*$ that is defined by:

$$\pi \in \mathcal{S} \Longleftrightarrow \exists_{1 \leq a < p} \forall_{x \in \mathbb{Z}_p^*} : \pi(x) = ax \bmod p, \tag{7.4}$$

Let $V_{\mathcal{S}} = \mathbb{Z}_p^*$ be generated by g. Let $A_{\mathcal{S}} = \{(i, gi \bmod p) | i \in \mathbb{Z}_p^*\}$. Then $G_{\mathcal{S}} = (V_{\mathcal{R}}, A_{\mathcal{R}})$ is given in Figure 7.1(b).

Theorem 7.4. *$\pi \in S_p$ is an automorphism of $G_{\mathcal{S}}$ if and only if $\pi \in \mathcal{S}$.*

Proof. Observe that $(\mathbb{Z}_n^*, \cdot)$ is isomorphic to $(\mathbb{Z}_{n-1}, +)$. Indeed, the map $\phi : \mathbb{Z}_n^* \to \mathbb{Z}_{n-1}$ defined by

$$\phi(x) := \log_g x$$

is an isomorphism.

Hence, the theorem follows from Theorem 7.3. □

7.3 Affine Transformations

This section discusses how to find a graph with p vertices such that its automorphism group is the set of affine transformations on $\mathbb{Z}_p$. We will first show that this is impossible on normal graphs and that we have to switch to hypergraphs. Then, we will give an example of a hypergraph where the group of automorphisms are the affine transformations.

Let p be prime. Suppose $\mathcal{A}$ is the set of the permutations on $\mathbb{Z}_p$ defined by:

$$\pi \in \mathcal{A} \Longleftrightarrow \exists_{a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p} \forall_{x \in \mathbb{Z}_p} : \pi(x) = ax + b \bmod p. \tag{7.5}$$

Theorem 7.5 (Impossibility Result). *There exists no directed graph $G = (\mathbb{Z}_p, A)$, where $A \subset \mathbb{Z}_p^2$, whose automorphism group are precisely the affine transformations on $\mathbb{Z}_p$.*

Proof. We will show that if all affine transformations on $\mathbb{Z}_p$ are an automorphism of G, then G has to be the complete graph K_p. This implies that *all* permutations on $\mathbb{Z}_p$ are an automorphism of G contradicting the fact that the group of automorphisms consists of only the affine transformations.

Suppose $G = (V, A)$ has as automorphism group $\mathcal{A}$ consisting only of affine transformations.

If A is the empty set or consists of self loops only then by Definition 7.1 all permutations belong to the automorphism group of G.

Let $(x, y) \in A$, where $x \neq y$. Now consider (x', y') for $x' \neq y'$. Observe that $a = \frac{x'-y'}{x-y} \mod p$ exists and is not equal to zero. Moreover from

$$x' - \frac{x'-y'}{x-y}x = y' - \frac{x'-y'}{x-y}y \bmod p \quad (=: b)$$

it follows that

$$\begin{cases} x' = ax + b, \\ y' = ay + b. \end{cases}$$

Hence $\pi(x) := ax+b$, where $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$. Hence $\pi \in \mathcal{A}$. Since π is an automorphism of G it follows that $(x', y') \in A$. □

Extending to Hypergraphs

From the impossibility result it follows that since one relation $x' = \pi(x)$ does not fix $\pi \in \mathcal{A}$ at all, we get the complete graph. It also follows from the two relations $x' = \pi(x)$ and $y' = \pi(y)$ that π is fixed to a specific permutation in $\mathcal{A}$. So intuitively one could guess that in order to restrict π to being an element of $\mathcal{A}$ giving a third independent relation $z' = \pi(z)$ suffices. So, we should at least consider A being a subset of all possible (directed) triples (x, y, z), for $x, y, z \in V$.

Suppose $V_{\mathcal{A}} = \mathbb{Z}_p$, and g generating $\mathbb{Z}_p^*$. We will show that the group of automorphisms of $G_{\mathcal{A}} = (V_{\mathcal{A}}, A_{\mathcal{A}})$, where

$$A_{\mathcal{A}} = \{(b, a + b, ga + b) | 1 \leq a < p, 0 \leq b < p\},$$

is given by $\mathcal{A}$.

Theorem 7.6. *$\pi \in S_p$ is an automorphism of $G_{\mathcal{A}}$ if and only if $\pi \in \mathcal{A}$.*

Proof. Let $(X_1, X_2, X_3) \in A_{\mathcal{A}}$ and let $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$ be such that

$$(X_1, X_2, X_3) = (b, a + b, ga + b).$$

Hence

$$(\pi(X_1), \pi(X_2), \pi(X_3)) = (a_\pi b + b_\pi, a_\pi(a + b) + b_\pi, a_\pi(ga + b) + b_\pi) = (b', a' + b', ga' + b'),$$

where $a' = a_\pi a \in \mathbb{Z}_p^*$ and $b' = a_\pi b + b_\pi \in \mathbb{Z}_p$. It follows that $(\pi(X_1), \pi(X_2), \pi(X_3)) \in A_{\mathcal{A}}$.

Second, suppose that $\pi \in S_p$ is an automorphism of $G_{\mathcal{A}}$. Hence

$$(\pi(b), \pi(a + b), \pi(ga + b)) \in A_{\mathcal{A}}$$

for all $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p$.

If $b = 0$ and $a = 1$ then there exists $a' \in \mathbb{Z}_p^*$ and $b' \in \mathbb{Z}_p$ such that

$$(\pi(0), \pi(1), \pi(g)) = \left(b', a' + b', ga' + b'\right).$$

Assume that $\pi(g^i) = g^i a' + b'$. Then, if $b = 0$ and $a = g^i$

$$\left(\pi(0), \pi(g^i), \pi(g^{i+1})\right) = \left(b', g^i a' + b', g(g^i a') + b'\right) = \left(b', g^i a' + b', g^{i+1} a' + b'\right).$$

By induction, it follows that $\pi(x) = a'x + b'$ for all $x \in \mathbb{Z}_p$. Hence $\pi \in \mathcal{A}$. □

7.4 Möbius Transforms

Let p be prime. Given $a, b, c, d \in \mathbb{Z}_p$, the Möbius transform $\pi : \overline{\mathbb{Z}_p} \to \overline{\mathbb{Z}_p}$ is defined by

$$\pi(x) := \frac{ax + b}{cx + d}, \tag{7.6}$$

where $ad \neq bc$ and $\overline{Z_p} = \mathbb{Z}_p \cup \{\infty\}$.

Let $a \in \mathbb{Z}_p$. The following rules apply in $\overline{Z_p}$:

- $a + b$ and ab are computed as in the field $\mathbb{Z}_p$ for any $b \in \mathbb{Z}_p$,
- $a + \infty = \infty$,
- $a\infty = \infty$,
- $a/0 = \infty$ if $a \neq 0$,
- $a/\infty = 0$ if $a \neq 0$, and
- $\infty/\infty = 1$.

It is well-known that any three different evaluations of π fixes a permutation represented by a certain Möbius transform. It follows that the hypergraphs used in previous sections are not sufficient anymore since we would again get the complete 3-uniform hypergraph under the assumption that all Möbius transforms are representing an automorphism of the hypergraph.

Möbius transformations are an important primitive in projective geometry, i.e., they form the group $\mathrm{PG}(2, \mathbb{F})$ of projective transformations of the projective line $\mathrm{P}^1(\mathbb{F})$ for any field $\mathbb{F}$. Note that $\mathrm{P}^1(\mathbb{F}) = \overline{\mathbb{Z}_p}$ for $\mathbb{F} = \mathbb{Z}_p$.

We give some well known results for the Möbius transformations in projective geometry. Proofs of the following theorems can be found for example in [FL12].

Definition 7.7. *Let z_1, z_2, z_3, z_4 be any four distinct points from $\overline{\mathbb{Z}_p}$. Then, their* cross-ratio *is defined by*

$$[z_1, z_2, z_3, z_4] := \frac{z_1 - z_3}{z_2 - z_3} \cdot \frac{z_2 - z_4}{z_1 - z_4}.$$

Let $\mathcal{M}$ be the collection of the following permutations:

$$\pi \in \mathcal{M} \iff \exists_{a,b,c,d \in \mathbb{Z}_p} \forall x \in \overline{\mathbb{Z}_p} : \pi(x) = \frac{ax + b}{cx + d}, ad - bc \neq 0. \tag{7.7}$$

Then $\mathcal{M}$ is a subgroup of S_{p+1}, the group of all permutations on $p + 1$ elements.

Theorem 7.8 ([FL12])**.** *Let z_1, z_2, z_3, z_4 be any four distinct points from $\overline{\mathbb{Z}_p}$ and let $\pi : \overline{\mathbb{Z}_p} \to \overline{\mathbb{Z}_p}$ be an injective map. Then π is a Möbius transformation if and only if it the cross-ratio is invariant under π. In other words, $\pi \in \mathcal{M}$ if and only if*

$$[z_1, z_2, z_3, z_4] = [\pi(z_1), \pi(z_2), \pi(z_3), \pi(z_4)] .$$

◇

Let $G_{\mathcal{M}} = (V_{\mathcal{M}}, A_{\mathcal{M}})$ be defined by

$$V_{\mathcal{M}} = \overline{\mathbb{Z}_p} = \{0, g^0, g^1, \ldots, g^{n-1}, \infty\}$$

and

$$A_{\mathcal{M}} = \{ (z_1, z_2, z_3, z_4) | \, z_1, z_2, z_3, z_4 \in \overline{\mathbb{Z}_p} \text{ are distinct and } [z_1, z_2, z_3, z_4] = g \}$$

Theorem 7.9. *$\pi \in S_{p+1}$ is an automorphism of $G_{\mathcal{M}}$ if and only if $\pi \in \mathcal{M}$.*

Proof. This is a direct consequence of Theorem 7.8. □

The following lemma provides an alternative definition of $A_{\mathcal{M}}$.

Lemma 7.10. *Let $A'_{\mathcal{M}}$ be given by*

$$A'_{\mathcal{M}} = \left\{ \left(\frac{a}{c}, \frac{b}{d}, \frac{a+b}{c+d}, \frac{ga+b}{gc+d} \right) \middle| \, a, b, c, d \in \mathbb{Z}_p, ad \neq bc \right\}.$$

Then $A'_{\mathcal{M}} = A_{\mathcal{M}}$.

Proof. Let $(z_1, z_2, z_3, z_4) \in A_{\mathcal{M}}$. Consider the function

$$f(z) := \frac{z - z_2}{z_3 - z_2} \cdot \frac{z_3 - z_1}{z - z_1}.$$

Then $f(z_1) = \infty$, $f(z_2) = 0$ and $f(z_3) = 1$. Moreover $f(z_4) = [z_1, z_2, z_3, z_4] = g$ and

$$\begin{aligned}
[f(z_1), f(z_2), f(z_3), f(z_4)] &= \frac{f(z_1) - f(z_3)}{f(z_2) - f(z_3)} \cdot \frac{f(z_2) - f(z_4)}{f(z_1) - f(z_4)} \\
&= \frac{f(z_1) - f(z_3)}{f(z_1) - f(z_4)} \cdot \frac{f(z_2) - f(z_4)}{f(z_2) - f(z_3)} \\
&= 1 \cdot \frac{0 - g}{0 - 1} \\
&= g.
\end{aligned}$$

Hence $f(z)$ is a Möbius transformation and thus $\pi(z) := f^{-1}(z)$ exists and is also a Möbius transformation. Therefore, there exists $a, b, c, d \in \mathbb{Z}_p$, where $ad - bc \neq 0$, such that $\pi(\infty) = a/c = z_1$, $\pi(0) = b/d = z_2$, $\pi(1) = (a+b)/(c+d) = z_3$, and $\pi(g) = (ga+b)/(gc+d) = z_4$.

Next, let $(z_1, z_2, z_3, z_4) \in A'_{\mathcal{M}}$. By definition there exist $a, b, c, d \in \mathbb{Z}_p$, where $ad - bc \neq 0$, and where

$$(z_1, z_2, z_3, z_4) = \left(\frac{a}{c}, \frac{b}{d}, \frac{a+b}{c+d}, \frac{ga+b}{gc+d} \right).$$

The cross-ratio is computed as follows

$$\begin{aligned}
[z_1, z_2, z_3, z_4] &= \frac{z_1 - z_3}{z_2 - z_3} \cdot \frac{z_2 - z_4}{z_1 - z_4} \\
&= \frac{\frac{a}{c} - \frac{a+b}{c+d}}{\frac{b}{d} - \frac{a+b}{c+d}} \cdot \frac{\frac{b}{d} - \frac{ga+b}{gc+d}}{\frac{a}{c} - \frac{ga+b}{gc+d}} \\
&= \frac{ad - bc}{c(c+d)} \cdot \frac{d(c+b)}{bc - ad} \cdot \frac{g(cb - ab)}{d(gc+d)} \cdot \frac{c(gc+d)}{ad - bc} \\
&= g.
\end{aligned}$$

□

7.5 Other Permutation groups

Let $\mathcal{P} \subset S_n$ be a permutation group acting on n elements, where $\mathcal{P}$ is represented by a rotation, rescaling, affine transformation, or Möbius transformation. The previous sections discussed u-uniform hypergraphs on n vertices satisfying the property that its group of automorphisms is precisely $\mathcal{P}$. We used the following property of $\mathcal{P}$: given only u evaluations of some permutation π, one can decide wether $\pi \in \mathcal{P}$ and, if so, one can uniquely determine π.

It is well-known that there are finitely many permutation groups acting on n elements satisfying the property that, given some fixed $u < n$ evaluations of some permutation π, one can decide whether some permutation π belongs to the group and, if so, uniquely determine π. Furthermore, these groups have been classified and it turned out that there are just a few groups with this property (see for example [Pas91]). Let $\mathcal{P}$ be such permutation group. We conjecture that there exists a u-uniform hypergraph on n vertices such that the group of automorphisms is precisely $\mathcal{P}$.

Note that for the trivial permutation groups, the symmetric group S_n and the alternating group A_n, such hypergraphs are also trivial. Indeed for S_n one could take the n-uniform hypergraph $G = (V, A)$ with $|V| = n$, where the arcs are the $n!$ permutations of S_n applied to V. Similarly for A_n one could take the $(n-1)$-uniform hypergraph $G' = (V, A')$ with $|V| = n$, where the arcs are the $n!/2$ permutations of A_n applied to some fixed subset of V of length $n-1$.

If we drop the restriction that the graph should have n vertices if $\mathcal{P} \subset S_n$, then for any permutation group $\mathcal{P}$ we can find an arc-colored directed 2-uniform graph, such that its group of automorphisms is precisely $\mathcal{P}$. Indeed the Cayley graph of $\mathcal{P}$ has the desired properties.

Definition 7.11 (Cayley graph)**.** *Let $\mathcal{P} = \langle S \rangle$ be a permutation group with generating set $S \subseteq \mathcal{P}$. The* Cayley graph *of $\mathcal{P}$ is an arc-colored directed graph $G(V, A)$ with color set C, where $V = \mathcal{P}$ and $C = \{c_s | s \in S\}$. The arc with color c_s are given by*

$$A_{c_s} = \{(\pi, \pi \circ s) | \pi \in \mathcal{P}\}$$

for each $s \in S$. And $A = \bigcup_{s \in S} A_{c_s}$.

Observe that the graphs for rotation and rescaling in Section 7.2 are the Cayley graphs of the cyclic groups $(\mathbb{Z}_n, +)$, generated by 1, and $(\mathbb{Z}_p^*, \times)$, generated by g.

Example 7.12. A more advanced example is the permutation π represented by a translation over $\mathbb{F}_q$, where $q = p^m$ for some prime p and $m > 1$, i.e., $\pi(x) = x + r$ for some fixed $r \in \mathbb{F}_q$.

Note that with respect to addition $\mathbb{F}_q$ is isomorphic to $\mathbb{Z}_p^m$, which is generated by the m unity vectors $\mathbf{e}_1, \ldots, \mathbf{e}_m$.

Let the colors be given by $C = \{c_1, \ldots, c_m\}$. Then the Cayley graph of $\mathbb{Z}_p^m$ is given by $G = (V, A)$, where $V = \mathbb{Z}_p^m$ and $A = \bigcup_{i=1}^m A_{c_i}$, where

$$A_{c_i} = \{(\mathbf{v}, \mathbf{v} + \mathbf{e}_i) | \mathbf{v} \in V\}.$$

◇

Chapter 8

Conclusions

In this thesis, we have considered secure linear programming as a case study in secure multiparty computation. To this end, we have discussed in detail how to implement the simplex algorithm securely.

Our protocols have been tested during the EU-FP7 project SecureSCM [Sec08]. The performance due to choices between the tableau representations, pivoting rules, and number representations is compared. It turned out that small tableau simplex with Dantzig's pivoting rule performed best in the test cases. Furthermore, rational pivoting turned out to be advantageous over integer pivoting: the running time for solving linear programs with about 150 constraints and 150 variables were a few minutes for small tableau simplex using rational pivoting and about 1 hour for small tableau simplex using integer pivoting.

In addition, we have implemented our protocols using VIFF (Virtually Ideal Framework Functionality, see [Gei10, VIF08]). Our protocols required about 1 day of computation on an ordinary Windows 7 PC to solve linear programs with about 300 variables and 200 constraints.

In conclusion, our protocols are able to solve problem instances way beyond toy examples. However, it will still be practically infeasible to solve linear programming problems having over millions of variables and constraints, which are quite reasonable in practice.

To be able to solve practical problems securely via multiparty computation the cryptographic tools need to be optimized further. In our experience, it turned out that the protocols spend most of the time in the protocols for comparison and, more specifically, generation of random bits. Improving these primitives would increase the efficiency of our protocols significantly.

With respect to secure optimization as a whole, it would be interesting to have efficient secure protocols for interior point methods. When solving more general optimization problems securely such as quadratic programming and semi-definite programming one is limited to the interior point methods. But also with respect to linear programming current research is improving the interior point methods so that at some stage they will perform better in general than the simplex algorithm.

In addition to solving linear programs securely, we addressed the problem of secure validation of a result. We showed how to extract a certificate from the outputs of the protocols for linear programming that can be used to validate the outputs very efficiently.

We showed that this idea can be used to design in an efficient way universally verifiable protocols, i.e., protocols with the property that the outputs can be validated by anyone. Indeed, if a protocol outputs a solution and a corresponding certificate proving correctness, then one just needs an universally verifiable protocol to check the certificate. It follows that any protocol solving an NP-problem is universally verifiable if validation of the certificate can be verified universally.

We showed how to transform the protocol of [CDN01] into an universally verifiable protocol by turning the interactive Σ-protocols into noninteractive zero-knowledge proofs. While the protocol of [CDN01] provides security under modular composition, see [Can00], we note that our transformations can also be applied to the protocol of [DN03] to achieve security under any composition, see for example [Can01].

Since universally verifiability is an important issue in practical applications of secure multiparty computation such as electronic voting and cloud computing, it would be interesting to have more efficient protocols than [CDN01] that enable universally verifiable computation.

Bibliography

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the fiat-shamir transform: Minimizing assumptions for security and forward-security. In *EUROCRYPT*, pages 418–433, 2002. (Cited on pages 18, 19, and 20).

[ABFL06] M. Atallah, M. Blanton, K. Frikken, and J. Li. Efficient Correlated Action Selection. In *Proc. 10th Financial Cryptography and Data Security Conference*. Anguilla, British West Indies, 2006. (Cited on page 155).

[ACS02] J. Algesheimer, J. Camenish, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 417–432. Springer-Verlag, 2002. (Cited on page 153).

[AP01] D. Azulay and J. Pique. A revised simplex method with integer Q-matrices. *ACM Trans. Math. Softw.*, 27:350–360, September 2001. (Cited on page 47).

[AR93] S. Axsater and K. Rosling. Notes: installation vs. echelon stock policies for multilevel inventory control. In *Manegement Science 39*, volume 10, 1993. (Cited on page 4).

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *Proc. 8th annual ACM Symposium on Principles of distributed computing*, pages 201–209. ACM Press, 1989. (Cited on page 88).

[BBR09] Alice Bednarz, Nigel Bean, and Matthew Roughan. Hiccups on the road to privacy-preserving linear programming. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, WPES '09, pages 117–120, 2009. ISBN 978-1-60558-783-7. (Cited on page 4).

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009. (Cited on page 2).

[Bea55] E. M. L. Beale. Cycling in the dual simplex algorithm. *Naval Research Logistics Quarterly*, 2(4):269–275, 1955. (Cited on page 34).

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 1991. (Cited on page 1).

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 103–112. ACM, New York, NY, USA, 1988. ISBN 0-89791-264-0. (Cited on page 18).

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO*, pages 390–420, 1992. (Cited on page 14).

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault tolerant distributed computation. In *Proc. of 20th ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988. (Cited on pages 1, 2, 6, 23, 75, and 84).

[BK04] I. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in cryptology - ASIACRYPT'04*, volume 3329 of *LNCS*, pages 515–529. Springer-Verlag, 2004. (Cited on page 155).

[Bla77] Robert G. Bland. New Finite Pivoting Rules for the Simplex Method. *Mathematics of Operations Research*, 2:103–107, 1977. (Cited on pages 27, 33, and 34).

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, CCS '93, pages 62–73. ACM, New York, NY, USA, 1993. ISBN 0-89791-629-8. (Cited on page 18).

[BT97] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, Massachusetts, 1997. ISBN 1-886529-19-1. (Cited on pages 3, 27, 28, 29, and 31).

[Can00] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1), 2000. (Cited on pages 1, 20, 22, 23, 76, 85, 153, and 164).

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001. (Cited on pages 1 and 164).

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19. ACM, 1988. (Cited on page 2).

[CDI05] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Proc. of 2nd Theory of Cryptography Conference (TCC'05)*, pages 342–362, 2005. (Cited on pages 6, 9, 75, 79, 82, and 85).

[CDN01] R. Cramer, I. Damgård, and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–300. Springer-Verlag, 2001.
Full version (Oct 2000): http://eprint.iacr.org/2000/055. (Cited on pages 2, 7, 13, 23, 145, 146, 147, 148, 149, 150, 151, 153, and 164).

[CH10a] Octavian Catrina and Sebastiaan de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *SCN*, pages 182–199, 2010. (Cited on pages 6 and 94).

[CH10b] Octavian Catrina and Sebastiaan de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In *Proceedings of the 15th European conference on Research in computer security*, ESORICS'10, pages 134–150, 2010. ISBN 3-642-15496-4, 978-3-642-15496-6. (Cited on pages 4, 6, 75, and 107).

[CS60] A. Clark and H. Scarf. Optimal policies for a multi-echelon inventory problem. In *Manegement Science 6*, 1960. (Cited on page 4).

[CS10] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security*, pages 35–50, 2010. (Cited on pages 6 and 75).

[Dam10] Ivan Damgård. On Sigma Protocols, 2010. (Cited on pages 5, 9, 14, and 15).

[Dik74] I.I. Dikin. On the Convergence of an Iterative Process. *Upravlyaemye Sistemi*, 12:54–60, 1974. (Cited on pages 35 and 36).

[DK09] Rafael Deitos and Florian Kerschbaum. Parallelizing secure linear programming. *Concurr. Comput. : Pract. Exper.*, 21(10):1321–1350, 2009. ISSN 1532-0626. (Cited on page 4).

[DK11] Jannik Dreier and Florian Kerschbaum. Practical secure and efficient multiparty linear programming based on problem transformation. Cryptology ePrint Archive, Report 2011/108, 2011. (Cited on page 4).

[DN03] I. Damgård and J.B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer-Verlag, 2003. (Cited on pages 2 and 164).

[DT97] George B. Dantzig and Mukund N. Thapa. *Linear programming 1: introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. ISBN 0-387-94833-3. (Cited on pages 3, 27, 31, 36, and 37).

[DT03] George B. Dantzig and Mukund N. Thapa. *Linear Programming 2: Theory and Extensions*. Springer, 2003. ISBN 0-387-98613-8. (Cited on page 37).

[DT08] Ivan Damgård and Rune Thorbek. Efficient conversion of secret-shared values between different fields. *IACR Cryptology ePrint Archive*, 2008. (Cited on pages 6 and 75).

[Du01] Wenliang Du. *A study of several specific secure two-party computation problems*. Ph.D. thesis, West Lafayette, IN, USA, 2001. AAI3043719. (Cited on page 4).

[EL03] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann, 2003. (Cited on page 102).

[FH96] M. Franklin and S. Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, 1996. (Cited on pages 2 and 98).

[FL12] H. Finkelnberg and M. Lübke. Projectieve Meetkunde, 2012. (Cited on page 160).

[Fru39] R. Frucht. Herstellung von Graphen mit vorgegebener abstrakter Gruppe. *Compositio Math.*, 6:239–250, 1939. (Cited on page 157).

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology–CRYPTO '86*, pages 186–194. Springer-Verlag, London, UK, 1986. (Cited on page 18).

[Gei10] Martin Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD dissertation, University of Aarhus, Denmark, Department of Computer Science, 2010. (Cited on page 163).

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. of 19th ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987. (Cited on pages 1, 145, and 146).

[Goe94] M. Goemans. Linear programming. Course Notes, Octobre 1994. URL `http://www-math.mit.edu/~goemans/notes-lp.ps`. (Cited on page 50).

[Gol02] O. Goldreich. Secure Multi-party Computation. http://www.wisdom.weizmann.ac.il/∼oded/pp.html, 2002. (Cited on page 1).

[GSV07] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *PKC # 07*, volume 4450 of *LNCS*, pages 330–342. Springer-Verlag, Berlin, 2007. (Cited on pages 75, 91, 93, and 155).

[Had93] J. Hadamard. Résolution d'une question relative aux déterminantes. *Bull. des sciences Math*, 2(17):240–248, 1893. (Cited on page 50).

[Hro01] Juraj Hromkovič. *Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics*. Springer-Verlag New York, Inc., New York, NY, USA, 2001. ISBN 3-540-66860-8. (Cited on page 40).

[HSSV09] Sebastiaan de Hoogh, Berry Schoenmakers, Boris Skoric, and José Villegas. Verifiable Rotation of Homomorphic Encryptions. In *Public Key Cryptography*, pages 393–410, 2009. (Cited on pages 7 and 155).

[ISN87] M. Itoh, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *IEEE Globecom*, pages 99–102, 1987. (Cited on page 9).

[JJ00] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 162–177. Springer-Verlag, 2000. (Cited on pages 2 and 155).

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Combinatorica, Vol. 4*, pages 373–395, 1984. (Cited on page 35).

[Kel11] Marcel Keller. *Theory and Practice of Cryptographic Protocols -or- Cryptography: Will It Blend?* PhD dissertation, University of Aarhus, Denmark, Department of Computer Science, 2011. (Cited on page 155).

[KM72] Victor Klee and George J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities*, volume III, pages 159–175. Academic Press, New York, 1972. (Cited on pages 27, 29, and 34).

[LA06] J. Li and M. Atallah. Secure and Private Collaborative Linear Programming. In *Proc. 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (ColaborateCom 2006)*, pages 19–26. Atlanta, USA, 2006. (Cited on pages 4, 50, and 155).

[LB93] H. Lee and C. Billington. Material management in decentralized supply chains. In *Operations research 41*, volume 5, 1993. (Cited on page 4).

[LPG02] Ch. Haehling von Lanzenhauer and K. Pilz-Glombik. Coordinating supply chain decisions: an optimization model. In *OR Spectrum 24*, volume 1, 2002. (Cited on page 4).

[Lue73] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, 1973. ISBN 0-201-04347-5. (Cited on pages 27, 28, 29, and 31).

[Mil00] Ronald E. Miller. *Optimization: foundations and applications*. John Wiley and Sons, Inc., 2000. ISBN 0-471-35169-5. (Cited on pages 3 and 37).

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991. (Cited on page 1).

[Mur05] Katta G. Murty. A gravitational interior point method for LP. *Opsearch*, 42 (1):28–36, 2005. ISSN 0030-3887. (Cited on pages 3 and 37).

[Nab09] H. Nabli. An overview on the simplex algorithm. *Applied Mathematics and Computation 210*, pages 479–489, 2009. (Cited on page 73).

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 1999. ISBN 0387987932. (Cited on pages 3, 36, and 37).

[Pas91] Antonio Pasini. Diagram Geometries for Sharply n-Transitive Sets of Permutations or of Mappings. *Des. Codes Cryptography*, 1(4):275–297, 1991. (Cited on page 162).

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000. (Cited on pages 18 and 19).

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, 2001. (Cited on page 1).

[Rei09] Tord Ingolf Reistad. Multiparty comparison - an improved multiparty protocol for comparison of secret-shared values. In *SECRYPT*, pages 325–330. INSTICC Press, 2009. (Cited on pages 6, 75, 92, and 93).

[Ros05] G. Rosenberg. Enumeration of All Extreme Equilibria of Bimatrix Games with Integer Pivoting and Improved Degeneracy Check. Research Report LSE-CDAM-2005-18, London School of Economics and Political Science, Department of Mathematics, 2005. www.cdam.lse.ac.uk/Reports/Files/cdam-2005-18.pdf. (Cited on pages 27 and 47).

[RT09a] T.I. Reistad and T. Toft. Information theoretic security. chapter Secret Sharing Comparison by Transformation and Rotation, pages 169–180. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-10229-5. (Cited on page 155).

[RT09b] Tord Reistad and Tomas Toft. Linear, constant-rounds bit-decomposition. In *ICISC*, pages 245–257, 2009. (Cited on pages 95 and 98).

[Sch12] Berry Schoenmakers. Lecture Notes Part 1 Cryptographic Protocols, 2012. (Cited on page 14).

[Sec08] SecureSCM. SecureSCM Project: Secure Supply Chain Management, 2008. (Cited on pages 143 and 163).

[Sec09] SecureSCM. Security Analysis. Deliverable D9.2, EU FP7 Project Secure Supply Chain Management (SecureSCM), 2009. (Cited on page 6).

[Sec10] SecureSCM. Security Analysis. Deliverable D3.2, EU FP7 Project Secure Supply Chain Management (SecureSCM), 2010. (Cited on page 6).

[Sha79] A. Shamir. How to share a secret. In *Communications of the ACM, 22(11)*, pages 612–613, 1979. (Cited on page 9).

[Sho05] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005. ISBN 0-521516-44-7. (Cited on page 76).

[ST06] B. Schoenmakers and P. Tuyls. Efficient Binary Conversion for Paillier Encryptions. In *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 522–537. Springer-Verlag, 2006. (Cited on pages 6, 75, 76, 90, 95, 96, and 98).

[Tof07] Tomas Toft. *Primitives and Applications for Multi-party Computation*. PhD dissertation, University of Aarhus, Denmark, BRICS, Department of Computer Science, 2007. (Cited on pages 50, 107, and 109).

[Tof09] Tomas Toft. Solving linear programs using multiparty computation. In *Financial Cryptography*, pages 90–107, 2009. (Cited on pages 4, 6, 75, 104, and 111).

[TW10] Björn Terelius and Douglas Wikström. Proofs of Restricted Shuffles. In *AFRICACRYPT*, pages 100–113, 2010. (Cited on pages 7, 155, 156, and 157).

[Vai09a] Jaideep Vaidya. Privacy-preserving linear programming. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 2002–2007, 2009. ISBN 978-1-60558-166-8. (Cited on page 4).

[Vai09b] Jaideep Vaidya. A secure revised simplex algorithm for privacy-preserving linear programming. In *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications*, pages 347–354, 2009. ISBN 978-0-7695-3638-5. (Cited on page 4).

[VB10] José Antonio Villegas Bautista. *Design of Advanced Primitives for Secure Multiplarty Computation: Special Shuffles and Integer Comparison.* Ph.D. thesis, Eindhoven, The Netherlands, 2010. NUR919. (Cited on pages 6 and 155).

[VIF08] VIFF Development Team. Virtually Ideal Functionality Framework, 2008. (Cited on page 163).

[WRW11] C. Wang, K. Ren, and J. Wang. Secure and practical outsourcing of linear programming in cloud computing. In *Proceedings of IEEE INFOCOM*, 2011. (Cited on page 4).

[Yao82] A. C. Yao. Protocols for secure computations. In *Proc. of the 23th IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982. (Cited on page 1).

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Proc. of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986. (Cited on page 1).

Appendix A

Secure Simplex Protocols

Here we list the protocols corresponding to Chapter 5.

A.1 Simplex Iteration

Protocol A.1:
$([\mathbf{T}], [\mathbf{s}], \text{pred}, [\mathbf{u}], [q]) \leftarrow \mathsf{Iterate}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [\mathbf{T}^0], [\mathbf{u}], [q])$

Input: $\mathbf{T} \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n'+1)}, \mathbf{s} \in \{1, \ldots, n+m\}^m$,
$\mathbf{T^0} \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}, \mathbf{u} \in \{1, \ldots, n\}^n$, $q \in \mathbb{Z}_{\langle k \rangle}$.

Output: $\mathbf{T} \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}, \mathbf{s} \in \{1, \ldots, n\}^m$,pred $\in \{\mathsf{UnboundedLP}, \mathsf{Optimal}\}$,
$\mathbf{u} \in \{1, \ldots, n\}^n$, $q \in \mathbb{Z}_{\langle k \rangle}$.

1 $(d, [\boldsymbol{\ell}], [\mathbf{p}^c]) \leftarrow \mathsf{GetPivotColumn}_{\mathsf{VAR}_1}([\mathbf{T}], [\mathbf{T}^0])$;
// LT: Prot. A.2, ST: Prot. A.6 or RS: Prot. A.10

2 **if** $d = 0$ **then**

3 **return** $([\mathbf{T}], [\mathbf{s}], \mathsf{Optimal}, [\mathbf{u}], [q])$;

4 $(d, [\boldsymbol{k}], [\mathbf{p}^r]) \leftarrow \mathsf{GetPivotRow}_{\mathsf{VAR}_1}(\mathbf{T}, \mathbf{p}^c, \mathbf{T}^0)$;
// LT Prot. A.3, ST: Prot. A.7 or RS: Prot. A.11

5 **if** $d = 0$ **then**

6 **return** $(\mathbf{T}, \mathbf{s}, \mathsf{UnboundedLP}, [\mathbf{u}], [q])$;

7 $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{Update}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [\boldsymbol{k}], [\mathbf{p}^c], [\mathbf{p}^r], [\mathbf{T}^0], [\mathbf{u}], [q])$;
// LT-RP: Prot. A.5, LT-IP: Prot. A.4, ST-RP: Prot. A.9, ST-IP: Prot. A.8, or , RS-IP: Prot. A.12

8 **return** $\mathsf{Iterate}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q])$;

A.1.1 Large Tableau Simplex

Protocol A.2: $(d, [\ell], [\mathbf{p^c}]) \leftarrow \mathsf{GetPivotColumn_{LT}}([\mathbf{T}])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$

Output: $d \in \{0,1\}$, $[\boldsymbol{\ell}] \in \{0,1\}^{n+m}$, $[\mathbf{p^c}] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$

$[\mathbf{t}] = ([t_{(m+1)1}], \ldots, [t_{(m+1)(n+m)}])$;

PIVOTRULE = DANTZIG :

2a $([\boldsymbol{\ell}], [min]) \leftarrow \mathsf{FindMin}([\mathbf{t}], \mathsf{LTZ})$; // $8\lceil \log(n+m) \rceil$ rnd, $(n+m-1)(4k+4)$ inv

3a $[d] \leftarrow [min] < 0$; // 6 rnd, $(4k+2)$ inv

PIVOTRULE = BLAND :

2b $[\boldsymbol{\ell}] \leftarrow \mathsf{FirstNeg}([\mathbf{t}])$; // 9 rnd, $(n+m)(4k+7)-1$ inv

3b $[d] \leftarrow \sum_{i=1}^{n} [\ell_i]$;

$d \leftarrow \mathsf{Open}([d])$; // 1 rnd, 1 inv.

if $d = 0$ **then return** $(0, [\ell], \mathbf{0})$;

$[\mathbf{p^c}] = [\mathbf{T}][\ell]$; // 1 rnd, $m+1$ inv.

return $(1, [\boldsymbol{\ell}], [\mathbf{p^c}])$

Protocol A.3: $(d, [\mathbf{k}], [\mathbf{p^r}]) \leftarrow \mathsf{GetPivotRow_{LT}}([\mathbf{T}], [\mathbf{p^c}])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{p^c}] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$.

Output: $d \in \{0,1\}$, $[\mathbf{k}] \in \{0,1\}^{m}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{n+m+1}$

$[\mathbf{t}] = ([t_{1(n+m+1)}], \ldots, [t_{m(n+m+1)}])$;

foreach $i \in \{1, \ldots, m\}$ **do parallel**

$[\beta_i] \leftarrow [p_i^c] \leq 0$; // 6 rnd, $m(4k+2)$ inv.

$d \leftarrow \sum_{i=1}^{m} [\beta_i] = m$; // 1 rnd, 1 inv.

if $d = 1$ **then return** $(0, [\mathbf{0}], \mathbf{0})$;

$[\mathbf{t}] \leftarrow [\mathbf{t}] + [\boldsymbol{\beta}]$;

PIVOTRULE = DANTZIG :

7a $([\mathbf{k}], [min]) \leftarrow \mathsf{FindMin}(([t_1], [p_1^c]), \ldots, ([t_m], [p_m^c]), \mathsf{FracLTZ})$; // $\lceil \log m \rceil (6+3)$ rnd, $(m-1)((4k+2)+5)$ inv.

PIVOTRULE = BLAND :

7b $([\mathbf{k}], [min]) \leftarrow \mathsf{FindMin}(([t_1], [p_1^c], [s_1]), \ldots, ([t_m], [p_m^c], [s_m]), \mathsf{BlandFracLTZ})$; // $\lceil \log m \rceil (\max\{6, \log^*(k)\} + 4)$ rnd, $(m-1)(8k + \log^*(k) \log k + 11)$ inv.

$[\mathbf{p^r}] = [\mathbf{k}][\mathbf{T}]$; // 1 rnd, $n+m+1$ inv.

return $(1, [\mathbf{k}], [\mathbf{p^r}])$

Protocol A.4:
$([\mathbf{T}'], [\mathbf{s}], [q]) \leftarrow \mathsf{Update}_{\mathsf{LT},\mathsf{IP}}([\mathbf{T}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\boldsymbol{\ell}] \in \{0,1\}^{n+m}$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{n+m+1}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $[\mathbf{T}'] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

$[p] \leftarrow [\mathbf{p}^c][\mathbf{k}]$; // 1 rnd, 1 inv.
$[t] \leftarrow \mathsf{Inv}([q])$; // 1 rnd, 1 inv.
$[\mathbf{w}] \leftarrow [t][\mathbf{p}]^c - [\mathbf{k}]$ // 1 rnd, $m+1$ inv.
$[v] \leftarrow [t][p]$; // 1 inv.
foreach $i \in \{1, \ldots, n'+1\}$ **do**
  foreach $j \in \{1, \ldots, m+1\}$ **do**
    $[t'_{ij}] \leftarrow ([t_{ij}], [r_j]) \cdot ([v], -[w_i])$; // 1 rnd, $(n'+1)(m+1)$ inv.
$[\ell] \leftarrow \sum_{j=1}^{n+m} [\ell_j]$;
$[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\ell])$; // 1 rnd, m inv.
return $([\mathbf{T}'], [\mathbf{s}], [\mathbf{u}], [p])$;

Protocol A.5:
$([\mathbf{T}'], [\mathbf{s}]) \leftarrow \mathsf{Update}_{\mathsf{LT},\mathsf{RP}}([\mathbf{T}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\boldsymbol{\ell}] \in \{0,1\}^{n+m}$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{n+m+1}$.

Output: $[\mathbf{T}'] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n'+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$.

$[p] \leftarrow [\mathbf{p}^c][\mathbf{k}]$; // 1 rnd, 1 inv.
$[t] \leftarrow \mathsf{Rec}([p], k)$;
$[\mathbf{w}] \leftarrow [t]([\mathbf{p}^c] - 2^f[\mathbf{k}])$; // 1 rnd, $m+1$ inv.
$[\mathbf{r}] \leftarrow [\mathbf{p}^r]$;
foreach $i \in \{1, \ldots, n+m+1\}$ **do**
  foreach $j \in \{1, \ldots, m+1\}$ **do**
    $[t'_{ij}] \leftarrow [t_{ij}] - \mathsf{TruncPr}([r_j][w_i], 3k, 2k)$; // 2 rnd, $2(n+m+1)(m+1)$ inv.
$[\ell] \leftarrow \sum_{j=1}^{n''} [\ell_j]$;
$[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\ell])$; // 1 rnd, m inv.
return $([\mathbf{T}'], [\mathbf{s}], [\mathbf{u}], [p])$;

A.1.2 Small Tableau Simplex

Protocol A.6: $(d, [\ell], [\mathbf{p^c}]) \leftarrow \mathsf{GetPivotColumn}_{\mathsf{ST}}([\mathbf{T}])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$

Output: $d \in \{0,1\}$, $[\boldsymbol{\ell}] \in \{0,1\}^n$, $[\mathbf{p^c}] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$

$[\mathbf{t}] = ([t_{(m+1)1}], \ldots, [t_{(m+1)n}])$;

PIVOTRULE = DANTZIG :

2a $([\boldsymbol{\ell}], [min]) \leftarrow \mathsf{FindMin}([\mathbf{t}], \mathsf{LTZ})$; // $\lceil \log n \rceil (6+2)$ rnd, $(n-1)((4k+2)+2)$ inv

3a $[d] \leftarrow [min] < 0$; // 6 rnd, $(4k+2)$ inv

PIVOTRULE = BLAND :

2b $[\boldsymbol{\ell}] \leftarrow \mathsf{FirstNeg}([\mathbf{t}])$; // $(6+3)$ rnd, $n((4k+2)+5)-1$ inv

3b $[d] \leftarrow \sum_{i=1}^{n} [\ell_i]$;

$d \leftarrow \mathsf{Open}([d])$; // 1 rnd, 1 inv.

if $d = 0$ **then return** $(0, [\boldsymbol{\ell}], \mathbf{0})$;

$[\mathbf{p^c}] = [\mathbf{T}][\ell]$; // 1 rnd, $m+1$ inv.

return $(1, [\boldsymbol{\ell}], [\mathbf{p^c}])$

Protocol A.7: $(d, [\mathbf{k}], [\mathbf{p^r}]) \leftarrow \mathsf{GetPivotRow}_{\mathsf{ST}}([\mathbf{T}], [\mathbf{p^c}])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$.

Output: $d \in \{0,1\}$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{p^r}] \in \mathbb{Z}_{\langle k \rangle}^{n+1}$

return $\mathsf{GetPivotRow}_{\mathsf{LT}}([\mathbf{T}], [\mathbf{p^c}])$

Protocol A.8:
$([\mathbf{T}'], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{Update}_{\mathsf{ST},\mathsf{IP}}([\mathbf{T}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [\mathbf{u}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\boldsymbol{\ell}] \in \{0,1\}^n$, $[\mathbf{k}] \in \{0,1\}^m$,
$[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{n+1}$, $[\mathbf{u}] \in \{1, \ldots, n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $[\mathbf{T}'] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\mathbf{u}] \in \{1, \ldots, n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

$[p] \leftarrow [\mathbf{p}^c][\mathbf{k}]$; // 1 rnd, 1 inv.

$[t] \leftarrow \mathsf{Inv}([q])$; // 2 rnd, 2 inv.

$[\mathbf{w}] \leftarrow [[t][\mathbf{p}]^c] - [\mathbf{k}]$; // 1 rnd, $m+1$ inv.

$[v] \leftarrow [t][p]$ // 1 inv.

$\mathbf{r} \leftarrow \mathbf{r} + [q][\boldsymbol{\ell}]$; // 1 rnd, n inv.

foreach $i \in \{1, \ldots, n'+1\}$ **do**

  foreach $j \in \{1, \ldots, m+1\}$ **do**

    $[t'_{ij}] \leftarrow ([t_{ij}], [r_j]) \cdot ([v], -[w_i])$; // 1 rnd, $(n'+1)(m+1)$ inv.

$[\ell'] \leftarrow [\mathbf{u}][\boldsymbol{\ell}]$; // 1 rnd, 1 inv.

$[k'] \leftarrow [\mathbf{s}][\mathbf{k}]$; // 1 inv

$[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\ell'])$; // 1 rnd, m inv.

$[\mathbf{u}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}], [\boldsymbol{\ell}], [k'])$; // n inv.

return $([\mathbf{T}'], [\mathbf{s}], [\mathbf{u}], [p])$;

Protocol A.9:
$([\mathbf{T}'], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{Update}_{\mathsf{ST},\mathsf{RP}}([\mathbf{T}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [\mathbf{u}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\boldsymbol{\ell}] \in \{0,1\}^n$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{n+1}$, $[\mathbf{u}] \in \{1, \ldots, n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $[\mathbf{T}'] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\mathbf{u}] \in \{1, \ldots, n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $[p] \leftarrow [\mathbf{p}^c][\mathbf{k}]$; // 1 rnd, 1 inv.
2 $[t] \leftarrow \mathsf{Rec}([p], k)$;
3 $[\mathbf{w}] \leftarrow [t]([\mathbf{p}]^c - 2^f[\mathbf{k}])$; // 1 rnd, $m+1$ inv.
4 $[\mathbf{r}] \leftarrow [\mathbf{p}^r]$;
5 $\mathbf{r} \leftarrow \mathbf{r} + 2^f[\boldsymbol{\ell}]$;
6 **foreach** $i \in \{1, \ldots, n'+1\}$ **do**
7 **foreach** $j \in \{1, \ldots, m+1\}$ **do**
8 $[t'_{ij}] \leftarrow [t_{ij}] - \mathsf{TruncPr}([r_j][w_i], 3k, 2k)$; // 2 rnd, $2(n'+1)(m+1)$ inv.
9 $[\ell'] \leftarrow [\mathbf{u}][\boldsymbol{\ell}]$; // 1 rnd, 1 inv.
10 $[k'] \leftarrow [\mathbf{s}][\mathbf{k}]$; // 1 inv
11 $[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\ell'])$; // 1 rnd, m inv.
12 $[\mathbf{u}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}], [\boldsymbol{\ell}], [k'])$; // n inv.
13 **return** $([\mathbf{T}'], [\mathbf{s}], [\mathbf{u}], [p])$;

A.1.3 Revised Simplex

Protocol A.11: $(d, [\mathbf{k}], [\mathbf{p^r}]) \leftarrow \mathsf{GetPivotRow}_{\mathsf{RS}}([\mathbf{D}], [\mathbf{T}^0], [\mathbf{p^c}])$

Input: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$.

Output: $d \in \{0,1\}$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{p^r}] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$

1 $[\mathbf{t}] = \left([\mathbf{d}_1][\mathbf{T}^0_{n+m+1}], \ldots, [\mathbf{d}_m][\mathbf{T}^0_{n+m+1}]\right)$; // 1 rnd, m inv.
2 **foreach** $i \in \{1, \ldots, m\}$ **do parallel**
3 $[\beta_i] \leftarrow [p_i^c] \leq 0$; // 6 rnd, $m(4k+2)$ inv.
4 $d \leftarrow \sum_{i=1}^m [\beta_i] = m$; // 1 rnd, 1 inv.
5 **if** $d = 1$ **then return** $(0, [\ell], \mathbf{0})$;
6 $[\mathbf{t}] \leftarrow [\mathbf{t}] + [\boldsymbol{\beta}]$;

PIVOTRULE = DANTZIG :

7a $([\mathbf{k}], [min]) \leftarrow \mathsf{FindMin}(([t_1], [p_1^c]), \ldots, ([t_m], [p_m^c]), \mathsf{FracLTZ})$; // $\lceil \log m \rceil (6+3)$ rnd, $(m-1)((4k+2)+5)$ inv.

PIVOTRULE = BLAND :

7b $([\mathbf{k}], [min]) \leftarrow \mathsf{FindMin}(([t_1], [p_1^c], [s_1]), \ldots, ([t_m], [p_m^c], [s_m]), \mathsf{BlandFracLTZ})$; // $\lceil \log m \rceil (\max\{6, \log^*(k)\} + 4)$ rnd, $(m-1)(8k + \log^*(k) \log k + 11)$ inv.

8 $[\mathbf{p^r}] = [\mathbf{k}][\mathbf{D}]$; // 1 rnd, $n+m+1$ inv.
9 **return** $(1, [\mathbf{k}], [\mathbf{p^r}])$

Protocol A.10: $(d, [\ell], [\mathbf{p^c}]) \leftarrow \mathsf{GetPivotColumn}_{\mathsf{RS}}([\mathbf{D}], [\mathbf{T}^0])$

Input: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$
Output: $d \in \{0,1\}$, $[\boldsymbol{\ell}] \in \{0,1\}^{n+m}$, $[\mathbf{p^c}] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$

$[\mathbf{t}] = \mathbf{d}_{m+1}[\mathbf{T}^0_{(1,\ldots,n+m)}]$; // 1 rnd, $n+m$ inv.
PIVOTRULE = DANTZIG :
  $([\boldsymbol{\ell}], [min]) \leftarrow \mathsf{FindMin}([\mathbf{t}], \mathsf{LTZ})$; // $\lceil \log n \rceil (6+2)$ rnd, $(n-1)((4k+2)+2)$ inv
  $[d] \leftarrow [min] < 0$; // 6 rnd, $(4k+2)$ inv
PIVOTRULE = BLAND :
1a $[\boldsymbol{\ell}] \leftarrow \mathsf{FirstNeg}([\mathbf{t}])$; // $(6+3)$ rnd, $n((4k+2)+5)-1$ inv
2a $[d] \leftarrow \sum_{i=1}^{n} [\ell_i]$;
$d \leftarrow \mathsf{Open}([d])$; // 1 rnd, 1 inv.
if $d = 0$ **then return** $(0, [\boldsymbol{\ell}], \mathbf{0})$;
$[\mathbf{p^c}] = [D]([\mathbf{T}][\ell])$; // 2 rnd, $2m+2$ inv.
return $(1, [\boldsymbol{\ell}], [\mathbf{p^c}])$

Protocol A.12:
$([\mathbf{D'}], [\mathbf{s}], [q]) \leftarrow \mathsf{Update}_{\mathsf{RS,IP}}([\mathbf{D}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [q])$

Input: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1,\ldots,n\}^m$, $[\boldsymbol{\ell}] \in \{0,1\}^n$, $[\mathbf{k}] \in \{0,1\}^m$,
$[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{u}] \in \{1,\ldots,n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
Output: $[\mathbf{T}'] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n'+1)}$, $[\mathbf{s}] \in \{1,\ldots,n\}^m$, $[\mathbf{u}] \in \{1,\ldots,n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
return $\mathsf{Update}_{\mathsf{LT,IP}}([\mathbf{D}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [q])$

Protocol A.13:
$([\mathbf{D'}], [\mathbf{s}], [q]) \leftarrow \mathsf{Update}_{\mathsf{RS,IP}}([\mathbf{D}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [q])$

Input: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1,\ldots,n\}^m$, $[\boldsymbol{\ell}] \in \{0,1\}^n$, $[\mathbf{k}] \in \{0,1\}^m$,
$[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{u}] \in \{1,\ldots,n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
Output: $[\mathbf{T}'] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n'+1)}$, $[\mathbf{s}] \in \{1,\ldots,n\}^m$, $[\mathbf{u}] \in \{1,\ldots,n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
return $\mathsf{Update}_{\mathsf{LT,RP}}([\mathbf{D}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [q])$

A.2 Simplex Initialization

A.2.0.1 Zero Feasible Simplex

Protocol A.14: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{ZeroFeasSimplex}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$
Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{UnboundedLP}, \mathsf{Optimal}\}$

1 $[\mathbf{s}] \leftarrow ([n+1], \ldots, [n+m])$;
2 $[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}] & [\mathbf{I}_m] & [\mathbf{b}] \\ [\mathbf{c}] & [\mathbf{0}] & [0] \end{pmatrix}$;
3 $([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.1
4 **if** pred = Optimal **then**
5 $([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.49
6 **return** $([\mathbf{x}], \text{pred})$

Protocol A.15: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{ZeroFeasSimplex}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$
Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{UnboundedLP}, \mathsf{Optimal}\}$

1 $[\mathbf{s}] \leftarrow ([n+1], \ldots, [n+m])$;
2 $[\mathbf{u}] \leftarrow ([1], \ldots, [n])$;
3 $[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}] & [\mathbf{b}] \\ [\mathbf{c}] & [0] \end{pmatrix}$;
4 $([\mathbf{T}], [\mathbf{s}], \text{pred}, [\mathbf{u}], [q]) \leftarrow \mathsf{Iterate}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q])$; // Prot. A.1
5 **if** pred = Optimal **then**
6 $([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.50
7 **return** $([\mathbf{x}], \text{pred})$

Protocol A.16: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{ZeroFeasSimplex}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$
Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{UnboundedLP}, \mathsf{Optimal}\}$

1 $[\mathbf{s}] \leftarrow ([n+1], \ldots, [n+m])$;
2 $[\mathbf{T}^0] \leftarrow \begin{pmatrix} [\mathbf{A}] & [\mathbf{I}_m] & [\mathbf{b}] \\ [\mathbf{c}] & [\mathbf{0}] & [0] \end{pmatrix}$;
3 $[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{I}_m] & [\mathbf{0}] \\ [\mathbf{0}] & \\ [1] & \end{pmatrix}$;
4 $([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.1
5 **if** pred = Optimal **then**
6 $([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.51
7 **return** $([\mathbf{x}], \text{pred})$

A.2.1 Standard two-phase Simplex

A.2.1.1 Large Tableau Simplex

Protocol A.17: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{TwoPhaseSimplex}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$

Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

1 $([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; // Prot. A.18

2 $([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.1

3 $t \leftarrow \mathsf{Open}([t_{m+1,n+m+1}])$; // 1 rnd, 1 inv.

4 **if** $t! = 0$ **then**

5 **return** $(\mathbf{0}, \text{pred})$;

6 $([\mathbf{T}], [\mathbf{s}]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{c}], [q])$; // Prot. A.19

7 $([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.1

$[\mathbf{x}] \leftarrow \mathsf{GetSolution}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.49

8 **return** $([\mathbf{x}], \text{pred})$

Protocol A.18: $([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$.

Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $[\mathbf{s}] \leftarrow ([n+m+1], \ldots, [n+2m])$;

$\mathsf{VAR} = \mathsf{IP}$:

2 $[q] \leftarrow [1]$;

3 **foreach** $i \in \{1, \ldots, m\}$ **do parallel**

4 $[\beta_i] \leftarrow 1 - 2([b_i] < 0)$; // 6 rnd, $m(4k+2)$ inv

5 $[b'_i] \leftarrow [\beta_i][b_i]$; // 1 rnd, m inv.

6 $[\mathbf{a}'_i] \leftarrow [\mathbf{a}_i][\beta_i]$; // $m(n+m)$ inv.

7 $[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}'] & [\text{Diag}(\boldsymbol{\beta})] & [\mathbf{b}'] \\ -\sum_{i=1}^{m} [\mathbf{a}'_i] & [\boldsymbol{\beta}] & [0] \end{pmatrix}$;

8 **return** $([\mathbf{T}], [\mathbf{s}])$

Protocol A.19: $([\mathbf{T}], [\mathbf{s}]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{c}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$

Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$.

$\mathsf{VAR} = \mathsf{IP}$:
  $[q'] \leftarrow \mathsf{Invert}(q)$;
1 **foreach** $i \in \{1, \ldots, m\}$ **do**
2 $[\gamma_i] \leftarrow [s_i] = n + m + i$; // $\log^*(n+2m)$ rnd, $m(4k+2)_=$ inv.

  $\mathsf{VAR} = \mathsf{RP}$:
3a $[w_i] \leftarrow [\gamma_i][t_{i(n+i)}] + (1 - [\gamma_i])$; // 1 rnd, m inv.

  $\mathsf{VAR} = \mathsf{IP}$:
3b $[w_i] \leftarrow [\gamma_i][q'][t_{i(n+i)}] + (1 - [\gamma_i])$; // 2 rnd, $2m$ inv.

4 $[s_i] \leftarrow [\gamma_i](n+i) + (1 - [\gamma_i])[s_i]$; // m inv.
5 $[\mathbf{t}_i] \leftarrow [w_i][\mathbf{t}_i]$; // 1 rnd, $m(n+m+1)$ inv
6 $[\mathbf{T}] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{c}], [\mathbf{s}])$; // Prot. A.20
7 **return** $([\mathbf{T}], [\mathbf{s}])$;

Protocol A.20: $[\mathbf{T}] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{c}], [\mathbf{s}])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $\mathbf{s} \in \{1, \ldots, n+m\}^m$

Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$

1 **foreach** $i = 1, \ldots, m$ **do**
2 $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n+m)$; // Prot. 4.40
3 $[v_i] \leftarrow [\mathbf{c}][\boldsymbol{\sigma}_i]$; // 1 rnd, m inv.
4 **foreach** $j = 1, \ldots, n+m+1$ **do**

  $\mathsf{VAR} = \mathsf{RP}$:
5a $[t_{(m+1)j}] \leftarrow \mathsf{TruncPR}([c_j] - [\mathbf{v}][\mathbf{T}_i], k+f, f)$;

  $\mathsf{VAR} = \mathsf{IP}$:
5b $[t_{(m+1)j}] \leftarrow [q][c_j] - [\mathbf{v}][\mathbf{T}_i]$;

6 **return** $[\mathbf{T}]$

A.2.1.2 Small Tableau Simplex

Protocol A.21: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{TwoPhaseSimplex}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$

Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

$([\mathbf{T}], [\mathbf{S}], [\mathbf{U}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}])$; // Prot. A.22

$([\mathbf{T}], [\mathbf{S}], \text{pred}, [\mathbf{U}], [q]) \leftarrow \mathsf{IteratePhaseIST}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{S}], [\mathbf{U}], [q])$; // Prot. A.52

$t \leftarrow \mathsf{Open}([t_{m+1,n+1}])$; // 1 rnd, 1 inv.

if $t < 0$ **then**

  return $(\mathbf{0}, \text{pred})$;

$([\mathbf{T}], [\mathbf{s}], [\mathbf{u}]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{S}], [\mathbf{c}], [\mathbf{U}], [q])$; // Prot. A.23

$([\mathbf{T}], [\mathbf{s}], \text{pred}, [\mathbf{u}], [q]) \leftarrow \mathsf{Iterate}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q])$; // Prot. A.1

$([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.50

return $([\mathbf{x}], \text{pred})$

Protocol A.22: $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$.

Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^{m}$, $[\mathbf{u}] \in \{1, \ldots, n+2m\}^{n+m}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$

$[\mathbf{s}_1] \leftarrow ([n+m+1], \ldots, [n+2m])$;

$[\mathbf{s}_2] \leftarrow \mathbf{1}$;

$[\mathbf{u}_1] \leftarrow ([1], \ldots, [n+m])$;

$[\mathbf{u}_2] \leftarrow \mathbf{0}$;

$\mathsf{VAR} = \mathsf{IP}$:

  $[q] = [1]$;

foreach $i \in \{1, \ldots, m\}$ **do parallel**

  $[\beta_i] \leftarrow 1 - 2([b_i] < 0)$; // 6 rnd, $m(4k+2)$ inv

  $[b'_i] \leftarrow [\beta_i][b_i]$; // 1 rnd, m inv.

  $[\mathbf{a}'_i] \leftarrow [\mathbf{a}_i][\beta_i]$; // $m(n+m)$ inv.

$$[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}'] & [\text{Diag}(\boldsymbol{\beta})] & [\mathbf{b}'] \\ -\sum_{i=1}^{m} [\mathbf{a}'_i] & [\boldsymbol{\beta}] & [0] \end{pmatrix};$$

$[\mathbf{u}] \leftarrow ([1], \ldots, [n+m])$;

return $([\mathbf{T}], [\mathbf{S}], [\mathbf{U}], [q])$

Protocol A.23: $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{c}], [\mathbf{u}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$,
$[\mathbf{u}] \in \{1, \ldots, n+2m\}^{n+m}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$
Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+1}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[\mathbf{u}] \in \{1, \ldots, n+m\}^n$.

VAR = IP :
1 $[q]' \leftarrow \mathsf{Invert}(q)$; // 2 rnd, 2 inv.
2 **foreach** $i \in \{1, \ldots, m\}$ **do**
3 $[\gamma_i] \leftarrow [s_i] = n + m + i$; // $\log^*(n+2m)$ rnd, $m \log^*(k) \log k$ inv.
VAR = RP :
4 $[w_i] \leftarrow [\gamma_i][t_{i(n+i)}] + (1 - [\gamma_i])$; // 1 rnd, m inv.
VAR = IP :
1a $[w_i] \leftarrow [\gamma_i][q'][t_{i(n+i)}] + (1 - [\gamma_i])$; // 2 rnd, $2m$ inv.
2 $[s_{1i}] \leftarrow [\gamma_i](n+i) + (1 - [\gamma_i])[s_{1i}]$; // m inv.
3 $[u_{1(n+i)}] \leftarrow [\gamma_i](n+m+i) + (1 - \gamma_i)[u_{1(n+i)}]$; // m inv.
4 $[u_{2(n+i)}] \leftarrow [\gamma_i] + (1 - \gamma_i)[u_{2(n+i)}]$; // m inv.
5 $[\mathbf{t}_i] \leftarrow [w_i][\mathbf{t}_i]$; // 1 rnd, $m(n+1)$ inv
6 $([\mathbf{T}], [\mathbf{u}_1]) \leftarrow \mathsf{DelCols}(([\mathbf{T}], [\mathbf{u}_1]), [\mathbf{u}_2])$; // Prot. 4.39
7 $[\mathbf{T}] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{ST},\mathsf{IP}}([\mathbf{T}], [\mathbf{c}], [\mathbf{s}_1], [\mathbf{u}_1])$; // Prot. A.24
8 **return** $([\mathbf{T}], [\mathbf{s}_1], [\mathbf{u}_2])$;

Protocol A.24: $[\mathbf{T}] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{c}], [\mathbf{s}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $\mathbf{s} \in \{1, \ldots, n+m\}^m$
Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$

1 **foreach** $i = 1, \ldots, m$ **do**
2 $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n+m)$; // Prot. 4.40
3 $[v_i] \leftarrow [\mathbf{c}][\boldsymbol{\sigma}_i]$; // 1 rnd, m inv.
4 **foreach** $j = 1, \ldots, n+1$ **do**
5 $[\boldsymbol{v}_j] \leftarrow \mathsf{ConvertUnary}([u_j], n+m)$; // Prot. 4.40
VAR = RP :
6a $[t_{(m+1)j}] \leftarrow \mathsf{TruncPr}([\mathbf{c}][\boldsymbol{v}_j] - [\mathbf{v}][\mathbf{T}][\boldsymbol{v}_j], k+f, f)$; // 2 rnd, $m+4$ inv.
VAR = IP :
6b $[t_{(m+1)j}] \leftarrow [q][\mathbf{c}][\boldsymbol{v}_j] - [\mathbf{v}][\mathbf{T}][\boldsymbol{v}_j]$; // 2 rnd, $m+4$ inv.
7 **return** $[\mathbf{T}]$

A.2.1.3 Revised Simplex

Protocol A.25: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{TwoPhaseSimplex}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$

Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

$([\mathbf{D}], [\mathbf{T}^0], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; // Prot. A.26

$([\mathbf{D}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.1

$[T] \leftarrow [\mathbf{d}_{m+1}][\mathbf{T}^0_{n+m+1}]$; // 1 rnd, 1 inv

$t \leftarrow \mathsf{Open}([t])$; // 1 rnd, 1 inv.

if $t < 0$ **then**

  return $(\mathbf{0}, \text{pred})$;

$([\mathbf{D}], [\mathbf{T}^0], [\mathbf{s}]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{RS}}([\mathbf{D}], [\mathbf{s}], [\mathbf{c}], [\mathbf{T}^0], [q])$; // Prot. A.27

$([\mathbf{D}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.1

$([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.51

return $([\mathbf{x}], \text{pred})$

Protocol A.26: $([\mathbf{T}], [\mathbf{T}^0], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$.

Output: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$

$[\mathbf{s}] \leftarrow ([n+m+1], \ldots, [n+2m])$;

$\mathsf{VAR} = \mathsf{IP}$:

  $[q] = [1]$;

foreach $i \in \{1, \ldots, m\}$ **do parallel**

  $[\beta_i] \leftarrow 1 - 2([b_i] < 0)$; // 6 rnd, $m(4k+2)$ inv

  $[b'_i] \leftarrow [\beta_i][b_i]$; // 1 rnd, m inv.

  $[\mathbf{a}'_i] \leftarrow [\mathbf{a}_i][\beta_i]$; // $m(n+m)$ inv.

$\mathbf{T}^0 \leftarrow \begin{pmatrix} [\mathbf{A}'] & [\text{Diag}(\boldsymbol{\beta})] & [\mathbf{b}'] \\ [\mathbf{0}] & [\mathbf{1}] & [0] \end{pmatrix}$;

$\mathbf{D} \leftarrow \begin{pmatrix} [\mathbf{I}_m] & [\mathbf{0}] \\ [-\mathbf{1}] & [1] \end{pmatrix}$;

return $([\mathbf{D}], [\mathbf{T}^0], [\mathbf{s}], [q])$

Protocol A.27: $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{RS}}([\mathbf{D}], [\mathbf{s}], [\mathbf{c}], [\mathbf{T}^0], [q])$

Input: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times m+1}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$

Output: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times m+1}$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$.

1 **foreach** $i \in \{1, \ldots, m\}$ **do**
2 $\quad [\gamma_i] \leftarrow [s_i] = n + m + i$; // $\log^*(n+2m)$ rnd, $m \log^*(n+2m) \log(n+2m)$ inv.
3 $\quad [v_i] \leftarrow [\gamma_i][t^0_{i(n+i)}] + (1 - [\gamma_i])$; // 1 rnd, m inv.
4 $\quad [s_i] \leftarrow [\gamma_i](n+i) + (1 - [\gamma_i])[s_i]$; // m inv.
5 **foreach** $i \in \{1, \ldots, m\}$ **do**
6 $\quad [\mathbf{d}_i] \leftarrow [v_i][\mathbf{d}_i]$; // 1 rnd, $m(m+1)$ inv
7 $([\mathbf{D}], [\mathbf{T}^0]) \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{RS}}([\mathbf{D}], [\mathbf{c}], [\mathbf{T}^0], [\mathbf{s}])$; // Prot. A.28
8 **return** $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0])$;

Protocol A.28: $([\mathbf{D}], [\mathbf{T}^0]) \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{RS}}([\mathbf{D}], [\mathbf{c}], [\mathbf{s}])$

Input: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times m+1}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $\mathbf{s} \in \{1, \ldots, n+m\}^m$

Output: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times m+1}$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{m+1 \times n+m+1}$

1 **foreach** $i = 1, \ldots, m$ **do**
2 $\quad [\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n+m)$; // Prot. 4.40
3 $\quad [v_i] \leftarrow [\mathbf{c}][\boldsymbol{\sigma}_i]$; // 1 rnd, m inv.
4 $[\mathbf{t}^0_{(m+1)}] \leftarrow [\mathbf{c}]$;
5 $[\mathbf{d}_{m+1}] \leftarrow (-[\mathbf{v}], 1)[\mathbf{D}]$; // 1 rnd, $m+1$ inv.
6 **return** $([\mathbf{D}], [\mathbf{T}^0])$

A.2.2 Two-Phase Simplex with One Artificial Variable

A.2.2.1 Large Tableau Simplex

Protocol A.29: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{TwoPhaseSimplex}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}^{m\times n}_{\langle k\rangle}$, $[\mathbf{b}] \in (\mathbb{Z})^{m}_{\langle k\rangle}$, $[\mathbf{c}] \in \mathbb{Z}^{n}_{\langle k\rangle}$
Output: $[\mathbf{x}] \in \mathbb{Z}^{n}_{\langle k\rangle}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

$([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; // Prot. A.30
$([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.1
$t \leftarrow \mathsf{Open}([t_{m+1,n+m+1}])$; // 1 rnd, 1 inv.
if $t! = 0$ **then**
    return $(\mathbf{0}, \text{pred})$;
$([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{c}], [q])$;
// Prot. A.31 or Prot. A.32
$([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.1
$([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.49
return $([\mathbf{x}], \text{pred})$

Protocol A.30: $([\mathbf{T}], [\mathbf{s}], [\mathbf{k}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$

Input: $[\mathbf{A}] \in \mathbb{Z}^{m\times n}_{\langle k\rangle}$, $[\mathbf{b}] \in \mathbb{Z}^{m}_{\langle k\rangle}$.
Output: $[\mathbf{T}] \in \mathbb{Z}^{(m+1)\times(n+m+1)}_{\langle k\rangle}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[q] \in \mathbb{Z}_{\langle k\rangle}$
$\mathsf{VAR} = \mathsf{IP}$:
    $[q] = [1]$;
$([b], [\mathbf{k}]) \leftarrow \mathsf{FindMin}([\mathbf{b}], \mathsf{LTZ})$; // $\lceil \log m \rceil (6+2)$ rnd, $(m-1)((4k+2)+2)$ inv.
$[\beta] \leftarrow 2([b] \geq 0) - 1$; // 6 rnd, $(4k+2)$ inv.
$[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}\,((n+1, \ldots, n+m), [\mathbf{k}], n+1)$;
$[\mathbf{T}] \leftarrow \begin{pmatrix} [\mathbf{A}] & [\mathbf{I}_m] & [\mathbf{b}] \\ [\mathbf{0}] & [\mathbf{0}] & [0] \end{pmatrix}$;
$[\mathbf{r}] \leftarrow \mathbf{1} - [\beta]\mathbf{k} + (1 - [\beta])\mathbf{e}_{m+1}$; // 1 rnd, m inv.
$[\mathbf{w}] \leftarrow [\mathbf{k}][\mathbf{T}]$; // $n+m+1$ inv.
foreach $i \in \{1, \ldots, m+1\}$ **do**
    foreach $j \in \{1, \ldots, n+m+1\}$ **do**
        $[t'_{ij}] \leftarrow [t_{ij}] - [r_i][w_j]$; // 1 rnd, $(m+1)(n+m+1)$ inv.
return $([\mathbf{T}'], [\mathbf{s}], [\mathbf{k}], [q])$

Protocol A.31: $([\mathbf{T}], [\mathbf{s}]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{LT},\mathsf{RP}}([\mathbf{T}], [\mathbf{s}], [\mathbf{k}], [\mathbf{c}])$

Input: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^n$.

Output: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$.

$[s] \leftarrow [\mathbf{s}][\mathbf{k}]$; // 1 rnd, 1 inv.

$[\gamma] \leftarrow [s] = n+m+1$; // $\log^*(n+m+1)$ rnd, $\log^*(n+m+1)\log(n+m+1)$ inv.

$[\mathbf{r}] \leftarrow [\mathbf{k}][\mathbf{T}]$; // $n+m+1$ inv.

$[\boldsymbol{\ell}'] \leftarrow \mathsf{FindFirst}([\mathbf{r}_{(n+1,\ldots,n+m)}], 1 - \mathsf{EQZ}(\cdot))$;

$[\boldsymbol{\ell}] \leftarrow (\mathbf{0}, [\boldsymbol{\ell}'])$; // Add n zero's to ℓ.

$[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\gamma](\sum_{i=1}^{n+m}[\ell_i]) + (1 - [\gamma])[s])$;
// 2 rnd, $m+2$ inv.

$[p'] \leftarrow \mathsf{Rec}([\mathbf{r}][\boldsymbol{\ell}], k)$;

$[v] \leftarrow 1$;

$[\mathbf{w}] \leftarrow [p']([\mathbf{T}][\boldsymbol{\ell}] - 2^f[\mathbf{k}])$;

$[\mathbf{w}] \leftarrow [\gamma][\mathbf{w}]$; // 1 rnd, $m+1$ inv.

foreach $i \in \{1, \ldots, m+1\}$ **do**

  foreach $j \in \{1, \ldots, n+m+1\}$ **do**

    $[t'_{ij}] \leftarrow \mathsf{TruncPr}([t_{ij}][v] - [w_i][r_j], 3k, 2k)$; // 1 rnd, $(m+1)(n+m+1)$ inv.

$[\mathbf{T}] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{LT},\mathsf{RP}}([\mathbf{T}], [\mathbf{c}], [\mathbf{s}])$; // Prot. A.20

return $([\mathbf{T}'], [\mathbf{s}])$

Protocol A.32: $([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{LT},\mathsf{IP}}([\mathbf{T}], [\mathbf{s}], [\mathbf{k}], [\mathbf{c}], [q])$

Input: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $[s] \leftarrow [\mathbf{s}][\mathbf{k}]$; // 1 rnd, 1 inv.

2 $[\gamma] \leftarrow [s] = n+m+1$; // $\log^*(n+m+1)$ rnd, $\log^*(n+m+1)\log(n+m+1)$ inv.

3 $[\mathbf{r}] \leftarrow [\mathbf{k}][\mathbf{T}]$; // $n+m+1$ inv.

4 $[\boldsymbol{\ell}'] \leftarrow \mathsf{FindFirst}([\mathbf{r}_{(n+1,\ldots,n+m)}], 1 - \mathsf{EQZ}(\cdot))$;

5 $[\boldsymbol{\ell}] \leftarrow (\mathbf{0}, [\boldsymbol{\ell}'])$; // Add n zero's to ℓ.

6 $[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\gamma](\sum_{i=1}^{n+m}[\ell_i]) + (1-[\gamma])[s])$; // 1 rnd, m inv.

7 $[q'] \leftarrow \mathsf{Invert}([q])$;

8 $[p] \leftarrow [\mathbf{r}][\boldsymbol{\ell}]$;

9 $[\alpha] \leftarrow 1 - 2([p] \leq 0)$;

10 $[v] \leftarrow [\alpha][q'][p]$;

11 $[v] \leftarrow [\gamma][v] + (1-[\gamma])$;

12 $[\mathbf{w}] \leftarrow [\alpha]\,([q'][\mathbf{T}][\boldsymbol{\ell}] - [\mathbf{k}])$;

13 $[\mathbf{w}] \leftarrow [\gamma][\mathbf{w}]$; // 1 rnd, $m+1$ inv.

14 **foreach** $i \in \{1, \ldots, m+1\}$ **do**

15 **foreach** $j \in \{1, \ldots, n+m+1\}$ **do**

16 $[t'_{ij}] \leftarrow [t_{ij}][v] - [w_i][r_j]$; // 1 rnd, $(m+1)(n+m+1)$ inv.

17 $[q] \leftarrow [\gamma][\alpha][p] + (1-[\gamma])[q]$; // 2 rnd, 3 inv.

18 $[\mathbf{T}] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{LT},\mathsf{IP}}([\mathbf{T}], [\mathbf{c}], [\mathbf{s}], [q])$; // Prot. A.20

return $([\mathbf{T}'], [\mathbf{s}])$

A.2.2.2 Small Tableau Simplex

Protocol A.33: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{TwoPhaseSimplex}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$

Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

1 $([\mathbf{T}], [\mathbf{S}], [\mathbf{U}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}])$; // Prot. A.34

2 $([\mathbf{T}], [\mathbf{S}], \text{pred}, [\mathbf{U}], [q]) \leftarrow \mathsf{IteratePhaseIST}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{S}], [\mathbf{U}], [q])$; // Prot. A.52

3 $t \leftarrow \mathsf{Open}([t_{m+1,n+1}])$; // 1 rnd, 1 inv.

4 **if** $t < 0$ **then**

5 **return** $(\mathbf{0}, \text{pred})$;

6 $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{S}], [\mathbf{c}], [\mathbf{U}], [q])$; // Prot. A.35 or Prot. A.36

7 $([\mathbf{T}], [\mathbf{s}], \text{pred}, [\mathbf{u}], [q]) \leftarrow \mathsf{Iterate}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q])$; // Prot. A.1

$([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$; // Prot. A.50

8 **return** $([\mathbf{x}], \text{pred})$

Protocol A.34: $([\mathbf{T}], [\mathbf{S}], [\mathbf{k}], [\mathbf{U}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$.

Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1) \times (n+2)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{u}] \in \{1, \ldots, n+m+1\}^{n+1}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$

VAR = IP :

1 $[q] = [1]$;

2 $([b], [\mathbf{k}]) \leftarrow \mathsf{FindMin}([\mathbf{b}], \mathsf{LTZ})$; // $\lceil \log m \rceil (6+2)$ rnd, $(m-1)((4k+2)+2)$ inv.

3 $[\beta] \leftarrow 2([m] \geq 0) - 1$; // 6 rnd, $(4k+2)$ inv.

4 $[\mathbf{s}_1] \leftarrow \mathsf{WriteAtPosition}\,((n+1, \ldots, n+m), [\mathbf{k}], n+m+1)$;

5 $[\mathbf{s}_2] \leftarrow \mathsf{WriteAtPosition}\,((0, \ldots, 0), [\mathbf{k}], 1)$;

6 $[\mathbf{u}_1] \leftarrow (1, \ldots, n, \sum_{i=1}^{m} [k_i] i)$;

7 $[\mathbf{u}_2] \leftarrow \mathbf{0}$;

8 $\mathbf{T} \leftarrow \begin{pmatrix} [\mathbf{A}] & [\beta \mathbf{1}] & [\mathbf{b}] \\ [\mathbf{0}] & [1] & [0] \end{pmatrix}$;

9 $[\mathbf{r}] \leftarrow \mathbf{1} - [\beta]\mathbf{k} + (1 - [\beta])\mathbf{e}_{m+1}$; // 1 rnd, m inv.

VAR = RP :

10a $[\mathbf{w}] \leftarrow [\mathbf{k}][\mathbf{T}] + 2^f \mathbf{e}_{n+1}$; // 1 rnd, $n+1$ inv.

VAR = IP :

10b $[\mathbf{w}] \leftarrow [\mathbf{k}][\mathbf{T}] + \mathbf{e}_{n+1}$; // 1 rnd, $n+1$ inv.

11 **foreach** $i \in \{1, \ldots, m\}$ **do**

12 **foreach** $i \in \{1, \ldots, m\}$ **do**

13 $[t'_{ij}] \leftarrow [t_{ij}] - [r_i][w_j]$; // 1 rnd, $(m+1)(n+m+1)$ inv.

14 **return** $([\mathbf{T}'], [\mathbf{S}], [\mathbf{k}], [\mathbf{U}], [q])$

Protocol A.35: $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{ST},\mathsf{RP}}([\mathbf{T}], [\mathbf{S}], [\mathbf{k}], [\mathbf{c}], [\mathbf{U}])$

Input: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+2)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^n$, $[\mathbf{u}] \in \{1, \ldots, n+m+1\}^{n+1}$.

Output: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[\mathbf{u}] \in \{1, \ldots, n+m\}^n$.

1 $[s] \leftarrow [\mathbf{s}_1][\mathbf{k}]$; // 1 rnd, 1 inv.

2 $[\gamma] \leftarrow [s] = n+m+1$; // $\log^*(n+m+1)$ rnd, $\log^*(n+m+1)\log(n+m+1)$ inv.

3 $[\mathbf{r}] \leftarrow [\mathbf{k}][\mathbf{T}]$; // $n+m+1$ inv.

4 $[\boldsymbol{\ell}] \leftarrow \mathsf{FindFirst}([\mathbf{r}], 1 - \mathsf{EQZ}(\cdot))$;

5 $[\boldsymbol{\ell}'] \leftarrow [\gamma][\mathbf{U}][\boldsymbol{\ell}] + (1 - [\gamma])[\mathbf{S}][\mathbf{k}]$; // 2 rnd, 6 inv.

6 $[\mathbf{k}'] \leftarrow [\gamma][\mathbf{S}][\mathbf{k}] + (1 - [\gamma])[\mathbf{U}][\boldsymbol{\ell}]$; // 6 inv

7 $[\mathbf{s}_1] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}_1], [\mathbf{k}], [\ell'_1])$; // 1 rnd, m inv.

8 $[\mathbf{u}_1] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}_1], [\boldsymbol{\ell}], [k'_1])$; // n inv.

9 $[\mathbf{u}_2] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}_2], [\boldsymbol{\ell}], [k'_2])$; // n inv.

10 $[p'] \leftarrow \mathsf{Rec}([\mathbf{r}][\boldsymbol{\ell}], k)$;

11 $[v] \leftarrow 1$;

12 $[\mathbf{w}] \leftarrow [p']([\mathbf{T}][\boldsymbol{\ell}] - 2^f[\mathbf{k}])$;

13 $[\mathbf{w}] \leftarrow [\gamma][\mathbf{w}]$; // 1 rnd, $m+1$ inv.

14 $[\mathbf{r}] \leftarrow [\mathbf{r}] + 2^f[\boldsymbol{\ell}]$;

15 **foreach** $i \in \{1, \ldots, m+1\}$ **do**

16 **foreach** $j \in \{1, \ldots, n+m+1\}$ **do**

17 $[t'_{ij}] \leftarrow \mathsf{TruncPr}([t_{ij}][v] - [w_i][r_j], 3k, 2k)$; // 1 rnd, $(m+1)(n+m+1)$ inv.

18 $[\mathbf{T}] \leftarrow \mathsf{DelCol}([\mathbf{T}], [\mathbf{u}_2])$;

19 $[\mathbf{u}_1] \leftarrow \mathsf{DelCol}([\mathbf{u}_1], [\mathbf{u}_2])$;

20 $[\mathbf{T}'] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{ST},\mathsf{RP}}([\mathbf{T}'], [\mathbf{c}], [\mathbf{s}_1], [\mathbf{u}_1])$; // Prot. A.24

return $([\mathbf{T}'], [\mathbf{s}_1], [\mathbf{u}_1])$

Protocol A.36: $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{ST},\mathsf{IP}}([\mathbf{T}], [\mathbf{S}], [\mathbf{k}], [\mathbf{c}], [\mathbf{U}], [q])$

Input: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+2)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^n$, $[\mathbf{u}] \in \{1, \ldots, n+m+1\}^{n+1}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[\mathbf{u}] \in \{1, \ldots, n+m\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $[s] \leftarrow [\mathbf{s}_1][\mathbf{k}]$; // 1 rnd, 1 inv.

2 $[\gamma] \leftarrow [s] = n+m+1$; // $\log^*(n+m+1)$ rnd, $\log^*(n+m+1)\log(n+m+1)$ inv.

3 $[\mathbf{r}] \leftarrow [\mathbf{k}][\mathbf{T}]$; // $n+m+1$ inv.

4 $[\boldsymbol{\ell}] \leftarrow \mathsf{FindFirst}([\mathbf{r}], 1 - \mathsf{EQZ}(\cdot))$;

5 $[\boldsymbol{\ell}'] \leftarrow [\gamma][\mathbf{U}][\boldsymbol{\ell}] + (1 - [\gamma])[\mathbf{S}][\mathbf{k}]$; // 2 rnd, 6 inv.

6 $[\mathbf{k}'] \leftarrow [\gamma][\mathbf{S}][\mathbf{k}] + (1 - [\gamma])[\mathbf{U}][\boldsymbol{\ell}]$;
; // 6 inv

7 $[\mathbf{s}_1] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}_1], [\mathbf{k}], [\ell'_1])$; // 1 rnd, m inv.

8 $[\mathbf{u}_1] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}_1], [\boldsymbol{\ell}], [k'_1])$; // n inv.

9 $[\mathbf{u}_2] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}_2], [\boldsymbol{\ell}], [k'_2])$; // n inv.

10 $[q'] \leftarrow \mathsf{Invert}([q])$;

11 $[p] \leftarrow [\mathbf{r}][\boldsymbol{\ell}]$;

12 $[\alpha] \leftarrow 1 - 2([p] \leq 0)$;

13 $[v] \leftarrow [\alpha][p][q']$;

14 $[\mathbf{w}] \leftarrow [\alpha]\,([p'][\mathbf{T}][\boldsymbol{\ell}] - [\mathbf{k}])$;

15 $[v] \leftarrow [\gamma][v] + (1 + [\gamma])$;

16 $[\mathbf{w}] \leftarrow [\gamma][\mathbf{w}]$;

17 $[\mathbf{r}] \leftarrow [\mathbf{r}] + [q][\boldsymbol{\ell}]$; // 1 rnd, n inv.

18 **foreach** $i \in \{1, \ldots, m+1\}$ **do**

19 **foreach** $j \in \{1, \ldots, n+m+1\}$ **do**

20 $[t'_{ij}] \leftarrow [t_{ij}][v] - [w_i][r_j]$; // 1 rnd, $(m+1)(n+m+1)$ inv.

21 $([\mathbf{T}], [\mathbf{u}]) \leftarrow \mathsf{DelCol}([\mathbf{T}], [\mathbf{u}_2])$;

22 $([\mathbf{u}_1]) \leftarrow \mathsf{DelCol}([\mathbf{u}_1], [\mathbf{u}_2])$;

23 $[q] \leftarrow [\gamma][\alpha][p] + (1 - [\gamma])[q]$; // 2 rnd, 3 inv.

24 $[\mathbf{T}'] \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{ST},\mathsf{IP}}([\mathbf{T}'], [\mathbf{c}], [\mathbf{s}], [\mathbf{u}], [q])$; // Prot. A.24

return $([\mathbf{T}'], [\mathbf{s}_1], [\mathbf{u}_1], [q])$

A.2.2.3 Revised Simplex

Protocol A.37: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{TwoPhaseSimplex}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$

Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

$([\mathbf{D}], [\mathbf{T}^0], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; // Prot. A.38

$([\mathbf{D}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.1

$[T] \leftarrow [\mathbf{d}_{m+1}][\mathbf{T}^0_{n+m+1}]$; // 1 rnd, 1 inv

$t \leftarrow \mathsf{Open}([t])$; // 1 rnd, 1 inv.

if $t < 0$ **then**

  return $(\mathbf{0}, \text{pred})$;

$([\mathbf{D}], [\mathbf{T}^0], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{RS}}([\mathbf{D}], [\mathbf{s}], [\mathbf{c}], [\mathbf{T}^0], [q])$;
// Prot. A.39 or Prot. A.40

$([\mathbf{D}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.1

$[\mathbf{x}] \leftarrow \mathsf{GetSolution}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.51

return $([\mathbf{x}], \text{pred})$

Protocol A.38: $([\mathbf{D}], [\mathbf{s}], [\mathbf{k}], [\mathbf{T}^0], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$.

Output: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$

$\mathsf{VAR} = \mathsf{IP}$:

  $[q] = [1]$;

$([b], [\mathbf{k}]) \leftarrow \mathsf{FindMin}([\mathbf{b}], \mathsf{LTZ})$; // $\lceil \log m \rceil (6+2)$ rnd, $(m-1)((4k+2)+2)$ inv.

$[\beta] \leftarrow 2([b] \geq 0) - 1$; // 6 rnd, $(4k+2)$ inv.

$[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}\,((n+1, \ldots, n+m), [\mathbf{k}], n+1)$;

$[\mathbf{T}^0] \leftarrow \begin{pmatrix} [\mathbf{A}] & [\mathbf{I}_m] & [\mathbf{b}] \\ [\mathbf{0}] & [\mathbf{0}] & [0] \end{pmatrix}$;

$[\mathbf{D}] \leftarrow \begin{pmatrix} [\mathbf{I}_m] & [\mathbf{0}] \\ [\mathbf{0}] & [1] \end{pmatrix}$;

$[\mathbf{r}] \leftarrow \mathbf{1} - [\beta]\mathbf{k} + (1 - [\beta])\mathbf{e}_{m+1}$; // 1 rnd, m inv.

foreach $i \in \{1, \ldots, m+1\}$ **do**

  foreach $j \in \{1, \ldots, n+m+1\}$ **do**

    $[d_{ij}] \leftarrow [d_{ij}] - [r_i][k_j]$; // 1 rnd, $(m+1)(m+1)$ inv.

return $([\mathbf{D}], [\mathbf{s}], [\mathbf{k}], [\mathbf{T}^0], [q])$

Protocol A.39: $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{RS,RP}}([\mathbf{D}], [\mathbf{s}], [\mathbf{k}], [\mathbf{c}], [\mathbf{T}^0])$

Input: $[\mathbf{D}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^n$, $[\mathbf{T}^0] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$.

Output: $[\mathbf{D}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[\mathbf{T}^0] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$.

1 $[s] \leftarrow [\mathbf{s}][\mathbf{k}]$; // 1 rnd, 1 inv.

2 $[\gamma] \leftarrow [s] = n+m+1$; // $\log^*(n+m+1)$ rnd, $\log^*(n+m+1)\log(n+m+1)$ inv.

3 $[\mathbf{r}] \leftarrow [\mathbf{k}][\mathbf{D}]$; // $m+1$ inv.

4 $[\mathbf{r}'] \leftarrow [\mathbf{r}][\mathbf{T}^0_{(n+1,\ldots,n+m)}]$; // m inv.

5 $[\boldsymbol{\ell}'] \leftarrow \mathsf{FindFirst}([\mathbf{r}'], 1 - \mathsf{EQZ}(\cdot))$;

6 $[\boldsymbol{\ell}] \leftarrow (\mathbf{0}, [\boldsymbol{\ell}'])$; // Add n zero's to ℓ.

7 $[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], [\gamma](\sum_{i=1}^{n+m}[\ell_i]) + (1 - [\gamma])[s])$; // 1 rnd, m inv.

8 $[p'] \leftarrow \mathsf{Rec}([\mathbf{r}][\boldsymbol{\ell}], k)$;

9 $[v] \leftarrow 1$;

10 $[\mathbf{w}] \leftarrow [p']([\mathbf{D}][\mathbf{T}^0][\boldsymbol{\ell}] - 2^f[\mathbf{k}])$;

11 $[\mathbf{w}] \leftarrow [\gamma][\mathbf{w}]$; // 1 rnd, $m+1$ inv.

12 **foreach** $i \in \{1, \ldots, m+1\}$ **do**

13 **foreach** $j \in \{1, \ldots, n+m+1\}$ **do**

14 $[t'_{ij}] \leftarrow \mathsf{TruncPr}([t_{ij}][v] - [w_i][r_j], 3k, 2k)$; // 1 rnd, $(m+1)(n+m+1)$ inv.

15 $([\mathbf{D}'], [\mathbf{T}^0]) \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{LT,RP}}([\mathbf{D}'], [\mathbf{c}], [\mathbf{s}], [\mathbf{T}^0])$; // Prot. A.28

16 **return** $([\mathbf{D}'], [\mathbf{s}])$

Protocol A.40: $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q]) \leftarrow \mathsf{InitializePhaseII}_{\mathsf{RS},\mathsf{IP}}([\mathbf{D}], [\mathbf{s}], [\mathbf{k}], [\mathbf{c}], [\mathbf{T}^0], [q])$

Input: $[\mathbf{D}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+1\}^m$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^n$,$[\mathbf{T}^0] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $[\mathbf{D}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[\mathbf{T}^0] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $[s] \leftarrow [\mathbf{s}][\mathbf{k}]$; // 1 rnd, 1 inv.

2 $[\gamma] \leftarrow [s] = n+m+1$; // $\log^*(n+m+1)$ rnd, $\log^*(n+m+1)\log(n+m+1)$ inv.

3 $[\mathbf{r}] \leftarrow [\mathbf{k}][\mathbf{D}]$; // $m+1$ inv.

4 $[\mathbf{r}'] \leftarrow [\mathbf{r}][\mathbf{T}^0_{(n+1,\ldots,n+m)}]$; // m inv.

5 $[\boldsymbol{\ell}'] \leftarrow \mathsf{FindFirst}([\mathbf{r}'], 1 - \mathsf{EQZ}(\cdot))$;

6 $[\boldsymbol{\ell}] \leftarrow (\mathbf{0}, [\boldsymbol{\ell}'])$; // Add n zero's to ℓ.

7 $[\mathbf{s}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}], [\mathbf{k}], \sum_{i=1}^{n+m}[\ell_i])$; // 1 rnd, m inv.

8 $[q'] \leftarrow \mathsf{Invert}([q])$;

9 $[p] \leftarrow [\mathbf{r}'][\boldsymbol{\ell}']$;

10 $[\alpha] \leftarrow 1 - 2([p] \leq 0)$;

11 $[v] \leftarrow [\alpha][p][q']$;

12 $[\mathbf{w}] \leftarrow [\alpha]\left([p'][\mathbf{D}][\mathbf{T}^0][\boldsymbol{\ell}] - [\mathbf{k}]\right)$;

13 $[v] \leftarrow [\gamma][v] + (1 - [\gamma])$;

14 $[\mathbf{w}] \leftarrow [\mathbf{w}]$;

15 **foreach** $i \in \{1, \ldots, m+1\}$ **do**

16 **foreach** $j \in \{1, \ldots, m+1\}$ **do**

17 $[d'_{ij}] \leftarrow [d_{ij}][v] - [w_i][r_j]$; // 1 rnd, $(m+1)^2$ inv.

18 $[q] \leftarrow [\gamma][\alpha][p] + (1 - [\gamma])[q]$;

19 $([\mathbf{D}'], [\mathbf{T}^0]) \leftarrow \mathsf{ChangeCostReducedRow}_{\mathsf{LT},\mathsf{RP}}([\mathbf{D}'], [\mathbf{c}], [\mathbf{s}], [\mathbf{T}^0], [q])$; // Prot. A.28

20 **return** $([\mathbf{D}'], [\mathbf{s}], [q])$

A.2.3 Big-M Method

A.2.3.1 Large valued M

Protocol A.41: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{BigMSimplex}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$
Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

$([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializeBigM}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$ `// Prot. A.42, Prot. A.43 or Prot. A.44`
$([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{Iterate}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [q])$; `// Prot. A.1`
$t \leftarrow \mathsf{Open}(-[t_{m+1,n+m+1}] > 2^L)$; `// 1 rnd, 1 inv.`
if $t = 1$ **then**
    return $(\mathbf{0}, \mathsf{Infeasible})$;
$([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [q])$; `// Prot. A.49, A.50, or A.51`
return $([\mathbf{x}], \text{pred})$

Protocol A.42: $([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializeBigM}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.
Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

$([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$ `// Prot. A.18 or Prot. A.30`
$[\mathbf{t}_{m+1}] \leftarrow M[\mathbf{t}_{m+1}] + [\mathbf{c}]$;
return $([\mathbf{T}], [\mathbf{s}], [q])$

Protocol A.43: $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{InitializeBigM}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.
Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[\mathbf{u}] \in \{1, \ldots, n+2m\}^{n+m}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

$([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; `// Prot. A.18 or Prot. A.30`
$[\mathbf{t}_{m+1}] \leftarrow M[\mathbf{t}_{m+1}] + [\mathbf{c}]$;
return $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q])$

Protocol A.44: $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q]) \leftarrow \mathsf{InitializeBigM}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.
Output: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

$([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; `// Prot. A.18 or Prot. A.30`
$[\mathbf{t}^0_{m+1}] \leftarrow [\mathbf{c}]$;
$[\mathbf{d}_{m+1}] \leftarrow M[\mathbf{d}_{m+1}]$;
return $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$

A.2.3.2 Alternative

Protocol A.45: $([\mathbf{x}], \text{pred}) \leftarrow \mathsf{BigMSimplexAlt}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in (\mathbb{Z})_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$

Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, pred $\in \{\mathsf{InfeasibleLP}, \mathsf{UnboundedLP}, \mathsf{Optimal}\}$

1 $([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializeBigMAlt}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$ `// Prot. A.46, Prot. A.47 or Prot. A.48`

2 $([\mathbf{T}], [\mathbf{s}], \text{pred}, [q]) \leftarrow \mathsf{IterateBigMAlt}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [q])$; `// Prot. A.55`

3 $t \leftarrow \mathsf{Open}([t_{m+2,n+m+1}] = 0)$; `// 1 rnd, 1 inv.`

4 **if** $t = 0$ **then**

5 **return** $(\mathbf{0}, \mathsf{Infeasible})$;

$([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [q])$; `// Prot. A.49, A.50, or A.51`

6 **return** $([\mathbf{x}], \text{pred})$

Protocol A.46: $([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializeBigMAlt}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.

Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^{m}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $([\mathbf{T}], [\mathbf{s}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; `// Prot. A.18 or Prot. A.30`

2 $[\mathbf{t}_{m+2}] \leftarrow [\mathbf{t}_{m+1}]$;

3 $[\mathbf{t}_{m+1}] \leftarrow [\mathbf{c}]$;

4 **return** $([\mathbf{T}], [\mathbf{s}], [q])$

Protocol A.47: $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{InitializeBigMAlt}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.

Output: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^{m}$, $[\mathbf{u}] \in \{1, \ldots, n+2m\}^{n+m}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; `// Prot. A.18 or Prot. A.30`

2 $[\mathbf{t}_{m+2}] \leftarrow [\mathbf{t}_{m+1}]$;

3 $[\mathbf{t}_{m+1}] \leftarrow [\mathbf{c}]$;

4 **return** $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q])$

Protocol A.48: $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q]) \leftarrow \mathsf{InitializeBigMAlt}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.
Output: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+2m\}^m$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

1 $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q]) \leftarrow \mathsf{InitializePhaseI}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}])$; `// Prot. A.18 or Prot. A.30`
2 $[\mathbf{t}^0_{m+2}] \leftarrow [\mathbf{t}^0_{m+1}]$;
3 $[\mathbf{t}^0_{m+1}] \leftarrow [\mathbf{c}]$;
4 $[\mathbf{d}_{m+2}] \leftarrow \mathbf{d}_{m+1}$;
5 $[\mathbf{d}_{m+1}] \leftarrow \mathbf{0}$;
6 **return** $([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$

A.2.4 Output Solution

Protocol A.49: $([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
$[\mathbf{x}] \leftarrow \mathbf{0}$;
foreach $i \in \{1, \ldots, m\}$ **do**
1 $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n)$;
2 $[\mathbf{x}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{x}], [\boldsymbol{\sigma}_i], [t_{i(n+m+1)}])$;
3 **return** $([\mathbf{x}], [q])$

Protocol A.50: $([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
1 **return** $\mathsf{GetSolution}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [q])$

Protocol A.51: $([\mathbf{x}], [q]) \leftarrow \mathsf{GetSolution}_{\mathsf{RS},\mathsf{VAR}}([\mathbf{D}], [\mathbf{s}], [\mathbf{T}^0], [q])$

Input: $[\mathbf{D}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m\}^m$, $[\mathbf{T}^0] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
Output: $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
$[\mathbf{x}] \leftarrow \mathbf{0}$;
$[\mathbf{t}] \leftarrow [\mathbf{D}][\mathbf{T}^0_{n+m+1}]$;
foreach $i \in \{1, \ldots, m\}$ **do**
1 $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n)$;
2 $[\mathbf{x}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{x}], [\boldsymbol{\sigma}_i], [t_i])$;
3 **return** $([\mathbf{x}], [q])$

A.2.5 Alternative Iterations

A.2.5.1 Phase I Small Tableau Simplex

Protocol A.52:
$([\mathbf{T}], [\mathbf{s}], \text{pred}, [\mathbf{u}], [q]) \leftarrow \mathsf{IteratePhaseIST}_{\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\mathbf{T}^0], [\mathbf{u}], [q])$

Input: $\mathbf{T} \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$, $\mathbf{S} \in \{1, \ldots, n+2m\}^m \times \{0,1\}^m$,
$\mathbf{U} \in \{1, \ldots, n+2m\}^{n+m} \times \{0,1\}^{n+m}$, $q \in \mathbb{Z}_{\langle k \rangle}$.
Output: $\mathbf{T} \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $\mathbf{S} \in \{1, \ldots, n+2m\}^m \times \{0,1\}^m$,
$\text{pred} \in \{\mathsf{UnboundedLP}, \mathsf{Optimal}\}$, $\mathbf{U} \in \{1, \ldots, n+2m\}^{n+m} \times \{0,1\}^{n+m}$,
$q \in \mathbb{Z}_{\langle k \rangle}$.

1 $(d, [\boldsymbol{\ell}], [\mathbf{p}^c]) \leftarrow \mathsf{GetPivotColumnPhaseIST}([\mathbf{T}], \mathbf{1} - [\mathbf{u}_2])$; // Protocol A.53
2 **if** $d = 0$ **then**
3 **return** $([\mathbf{T}], [\mathbf{S}], \mathsf{Optimal}, [\mathbf{U}], [q])$;
4 $(d, [\boldsymbol{k}], [\mathbf{p}^r]) \leftarrow \mathsf{GetPivotRow}_{\mathsf{ST}}(\mathbf{T}, \mathbf{p}^c)$; // Protocol A.7
5 **if** $d = 0$ **then**
6 **return** $(\mathbf{T}, \mathbf{S}, \mathsf{UnboundedLP}, [\mathbf{U}], [q])$;
7 $([\mathbf{T}], [\mathbf{S}], [\mathbf{U}], [q]) \leftarrow \mathsf{UpdatePhaseIST}_{\mathsf{VAR}}([\mathbf{T}], [\mathbf{S}], [\boldsymbol{k}], [\mathbf{p}^c], [\mathbf{p}^r], [\mathbf{U}], [q])$;
// Protocol A.54
8 **return** $\mathsf{IteratePhaseIST}_{\mathsf{VAR}}([\mathbf{T}], [\mathbf{S}], [\mathbf{U}], [q])$;

Protocol A.53: $(d, [\ell], [\mathbf{p^c}]) \leftarrow \mathsf{GetPivotColumnPhaseIST}([\mathbf{T}], [\mathbf{v}])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$
Output: $d \in \{0,1\}$, $[\boldsymbol{\ell}] \in \{0,1\}^n$, $[\mathbf{p^c}] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$

1 $[\mathbf{t}] = \big([v_1][t_{(m+1)1}], \ldots, [v_{n+m}][t_{(m+1)(n+m)}]\big)$;

PIVOTRULE = DANTZIG :
2a $([\boldsymbol{\ell}], [min]) \leftarrow \mathsf{FindMin}([\mathbf{t}], \mathsf{LTZ})$; // $\lceil \log n \rceil (6+2)$ rnd, $(n-1)((4k+2)+2)$ inv
3a $[d] \leftarrow [min] < 0$; // 6 rnd, $(4k+2)$ inv

PIVOTRULE = BLAND :
2b $[\boldsymbol{\ell}] \leftarrow \mathsf{FirstNeg}([\mathbf{t}])$; // $(6+3)$ rnd, $n((4k+2)+5)-1$ inv
3b $[d] \leftarrow \sum_{i=1}^{n} [\ell_i]$;

4 $d \leftarrow \mathsf{Open}([d])$; // 1 rnd, 1 inv.
5 **if** $d = 0$ **then return** $(0, [\boldsymbol{\ell}], \mathbf{0})$;
6 $[\mathbf{p^c}] = [\mathbf{T}][\ell]$; // 1 rnd, $m+1$ inv.
7 **return** $(1, [\boldsymbol{\ell}], [\mathbf{p^c}])$

Protocol A.54:
$([\mathbf{T}'], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{UpdatePhaseIST}_{\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [\mathbf{u}], [q])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\boldsymbol{\ell}] \in \{0,1\}^n$, $[\mathbf{k}] \in \{0,1\}^m$, $[\mathbf{p}^c] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$, $[\mathbf{p}^r] \in \mathbb{Z}_{\langle k \rangle}^{n+1}$, $[\mathbf{u}] \in \{1, \ldots, n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $[\mathbf{T}'] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+1)}$, $[\mathbf{s}] \in \{1, \ldots, n\}^m$, $[\mathbf{u}] \in \{1, \ldots, n\}^n$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

$([\mathbf{T}'], [\mathbf{s}_1], [\mathbf{u}_1], [q]) \leftarrow \mathsf{Update}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{T}], [\mathbf{s}_1], [\boldsymbol{\ell}], [\mathbf{k}], [\mathbf{p}^c], [\mathbf{p}^r], [\mathbf{u}_1], [q])$; `// Prot. A.8`

1 $[\ell'] \leftarrow [\mathbf{u}_2][\boldsymbol{\ell}]$; `// 1 rnd, 1 inv.`

2 $[k'] \leftarrow [\mathbf{s}_2][\mathbf{k}]$; `// 1 inv`

3 $[\mathbf{s}_2] \leftarrow \mathsf{WriteAtPosition}([\mathbf{s}_2], [\mathbf{k}], [\ell'])$; `// 1 rnd,` m `inv.`

4 $[\mathbf{u}_2] \leftarrow \mathsf{WriteAtPosition}([\mathbf{u}_2], [\boldsymbol{\ell}], [k'])$; `//` n `inv.`

5 **return** $([\mathbf{T}'], [\mathbf{S}], [\mathbf{U}], [p])$;

A.2.5.2 Big-M Alternative

Protocol A.55:
$([\mathbf{T}], [\mathbf{s}], \mathrm{pred}, [\mathbf{u}], [q]) \leftarrow \mathsf{IterateBigMAlt}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [\mathbf{T}^0], [\mathbf{u}], [q])$

Input: $\mathbf{T} \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n'+1)}, \mathbf{s} \in \{1, \ldots, n+m\}^m$, $\mathbf{T^0} \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}, \mathbf{u} \in \{1, \ldots, n\}^n$, $q \in \mathbb{Z}_{\langle k \rangle}$.

Output: $\mathbf{T} \in \mathbb{R}^{(m+1)\times(n+1)}, \mathbf{s} \in \{1, \ldots, n\}^m$,pred $\in \{\mathsf{UnboundedLP}, \mathsf{Optimal}\}$, $\mathbf{u} \in \{1, \ldots, n\}^n$, $q \in \mathbb{Z}_{\langle k \rangle}$.

1 $(d, [\boldsymbol{\ell}], [\mathbf{p}^c]) \leftarrow \mathsf{GetPivotColumnBigMAlt}_{\mathsf{VAR}_1}([\mathbf{T}], [\mathbf{T}^0])$; `// Prot. A.56`

2 **if** $d = 0$ **then**

3 **return** $([\mathbf{T}], [\mathbf{s}], \mathsf{Optimal}, [\mathbf{u}], [q])$;

4 $(d, [\boldsymbol{k}], [\mathbf{p}^r]) \leftarrow \mathsf{GetPivotRow}_{\mathsf{VAR}_1}(\mathbf{T}, \mathbf{p}^c, \mathbf{T}^0)$; `// Prot. A.3, A.7, or A.11.`

5 **if** $d = 0$ **then**

6 **return** $(\mathbf{T}, \mathbf{s}, \mathsf{UnboundedLP}, [\mathbf{u}], [q])$;

7 $([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q]) \leftarrow \mathsf{Update}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [\boldsymbol{k}], [\mathbf{p}^c], [\mathbf{p}^r], [\mathbf{T}^0], [\mathbf{u}], [q])$; `// Prot. A.4, A.5,A.8,A.9,A.12,A.13`

8 **return** $\mathsf{Iterate}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [q])$;

Protocol A.56: $(d, [\ell], [\mathbf{p^c}]) \leftarrow \mathsf{GetPivotColumnBigMAlt}_{\mathsf{VAR}}([\mathbf{T}])$

Input: $[\mathbf{T}] \in \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+1)}$

Output: $d \in \{0,1\}$, $[\boldsymbol{\ell}] \in \{0,1\}^{n+m}$, $[\mathbf{p^c}] \in \mathbb{Z}_{\langle k \rangle}^{m+1}$

$\mathsf{VAR} = \mathsf{LT}, \mathsf{ST}$

1a $[\mathbf{t}_1] = ([t_{(m+1)1}], \ldots, [t_{(m+1)(n+m)}])$;

2a $[\mathbf{t}_2] = ([t_{(m+2)1}], \ldots, [t_{(m+2)(n+m)}])$;

$\mathsf{VAR} = \mathsf{RS}$

1b $[\mathbf{t}_1] = [\mathbf{d}_{m+1}]([\mathbf{T}^0_1], \ldots, [\mathbf{T}^0_{(n+m)}])$;

2b $[\mathbf{t}_2] = [\mathbf{d}_{m+2}]([\mathbf{T}^0_1], \ldots, [\mathbf{T}^0_{(n+m)}])$;

$\mathsf{PIVOTRULE} = \mathsf{DANTZIG}$:

3a $([\boldsymbol{\ell}], [min]) \leftarrow \mathsf{FindMin}(([\mathbf{t}_1], [\mathbf{t}_2]), \mathsf{BigMLT})$; // $\lceil \log n + m \rceil (6+2)$ rnd, $3(n+m-1)((4k+2)+2)$ inv

4a $[d] \leftarrow [min] < 0$; // 6 rnd, $(4k+2)$ inv

$\mathsf{PIVOTRULE} = \mathsf{BLAND}$:

3b $[\boldsymbol{\ell}] \leftarrow \mathsf{FindFirst}(([\mathbf{t}_1], [\mathbf{t}_2]), \mathsf{BigMLTZ})$; // $(6+3)$ rnd, $(n+m)((4k+2)+5)-1$ inv

4b $[d] \leftarrow \sum_{i=1}^{n} [\ell_i]$;

5 $d \leftarrow \mathsf{Open}([d])$; // 1 rnd, 1 inv.

6 **if** $d = 0$ **then return** $(0, [\ell], \mathbf{0})$;

7 $[\mathbf{p^c}] = [\mathbf{T}][\ell]$; // 1 rnd, $m+2$ inv.

8 **return** $(1, [\boldsymbol{\ell}], [\mathbf{p^c}])$

Protocol A.57: $[b] \leftarrow \mathsf{BigMLTZ}([x], [y])$

1 $[\alpha] \leftarrow [y] < 0$;

2 $[\beta] \leftarrow [y] = 0$;

3 $[\gamma] \leftarrow [x] \leq 0$;

4 $[b] \leftarrow [\alpha] + (1-\alpha)\beta\gamma$; // 2 rnd, 2 inv.

5 **return** $[b]$

Protocol A.58: $[b] \leftarrow \mathsf{BigMLT}(([x_1], [x_2]), ([y_1], [y_2]))$

1 $[d_1] \leftarrow [x_1] - [x_2]$;

2 $[d_2] \leftarrow [y_1] - [y_2]$;

3 $[b] \leftarrow \mathsf{BigMLTZ}([d_1], [d_2])$;

4 **return** $[b]$

A.3 Simplex Verification

Protocol A.59: $\delta \leftarrow \mathsf{VerifySolution}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [\mathbf{x}], [q], \mathsf{pred}, [\mathbf{i}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+p+1)}$,
$[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$,
$\mathsf{pred} \in \{\mathsf{Optimal}, \mathsf{Infeasible}, \mathsf{Unbounded}\}$, $[\mathbf{i}] \in \{0,1\}^{n+m}$.

Output: $\delta \in \{0,1\}$.

1 **if** $\mathsf{pred} = \mathsf{Optimal}$ **then**
2 **return** $\mathsf{VerifyOptimal}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [\mathbf{x}], [q], \mathbf{u}, \mathbf{T}^0)$.
3 **if** $\mathsf{pred} = \mathsf{Infeasible}$ **then**
4 **return** $\mathsf{VerifyInfeasible}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{T}], [\mathbf{s}], [\mathbf{u}], [\mathbf{T}^0])$.
5 **if** $\mathsf{pred} = \mathsf{Unbounded}$ **then**
6 **return** $\mathsf{VerifyUnbounded}_{\mathsf{VAR}_1,\mathsf{VAR}_2}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [\mathbf{x}], [q], [\mathbf{i}])$.

A.3.1 Large Tableau Simplex

Protocol A.60: $\delta \leftarrow \mathsf{VerifyOptimal}_{\mathsf{LT},\mathsf{RP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [\mathbf{x}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+p+1)}$,
$[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.

Output: $\delta \in \{0,1\}$.

1 $[\mathbf{p}] \leftarrow -([t_{(m+1)(n+1)}], \ldots, [t_{(m+1)(n+m)}])$;
2 $[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
3 $[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq \mathbf{0})$;
4 $[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] = [\mathbf{c}][\mathbf{x}])$;
5 $[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [\mathbf{b}])$;
6 $[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [\mathbf{c}])$;
7 $\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta_i']) + \sum_{j=1}^{n}\left([\alpha_j'] + [\beta_j]\right)\right)$;
8 **return** $[\delta]$

Protocol A.61: $\delta \leftarrow \mathsf{VerifyInfeasible}_{\mathsf{LT}}([\mathbf{A}], [\mathbf{b}], [\mathbf{T}], [\mathbf{s}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+p+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$.

Output: $\delta \in \{0,1\}$.

1 $[\mathbf{p}] \leftarrow -([t_{(m+1)(n+1)}], \ldots, [t_{(m+1)(n+m)}])$;
2 $[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] > 0)$;
3 $[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq [\mathbf{0}])$;
4 $[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [\mathbf{0}])$;
5 $\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}[\alpha_i] \sum_{j=1}^{n}[\beta_j]\right)$;
6 $[\delta]$

Protocol A.62: $\delta \leftarrow \mathsf{VerifyOptimal}_{\mathsf{LT},\mathsf{IP}}([\mathbf{A}],[\mathbf{b}],[\mathbf{c}],[\mathbf{T}],[\mathbf{s}],[\mathbf{x}],[q])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k\rangle}^{m\times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k\rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k\rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k\rangle}^{(m+1)\times(n+m+p+1)}$, $[\mathbf{s}] \in \{1,\ldots,n+m+p\}^m$, $[\mathbf{x}] \in \mathbb{Z}_{\langle k\rangle}^{n}$, $[q] \in \mathbb{Z}_{\langle k\rangle}$.

Output: $\delta \in \{0,1\}$.

$[\mathbf{p}] \leftarrow -([t_{(m+1)(n+1)}],\ldots,[t_{(m+1)(n+m)}])$;
$[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
$[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq \mathbf{0})$;
$[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] = [\mathbf{c}][\mathbf{x}])$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [q][\mathbf{b}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [q][\mathbf{c}])$;
$[\chi] \leftarrow 1 - ([q] > 0)$;
$\delta \leftarrow \mathsf{EQZ}\left([\chi] + [\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta_i']) + \sum_{j=1}^{n}\left([\alpha_j'] + [\beta_j]\right)\right)$;
return $[\delta]$

Protocol A.63: $\delta \leftarrow \mathsf{VerifyUnbounded}_{\mathsf{LT},\mathsf{RP}}([\mathbf{A}],[\mathbf{b}],[\mathbf{c}],[\mathbf{T}],[\mathbf{s}],[\mathbf{i}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k\rangle}^{m\times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k\rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k\rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k\rangle}^{(m+1)\times(n+m+p+1)}$, $[\mathbf{s}] \in \{1,\ldots,n+m+p\}^m$, $[\mathbf{i}] \in \{0,1\}^{n+m}$.

Output: $\delta \in \{0,1\}$.

$[\mathbf{t}] \leftarrow [\mathbf{T}][\mathbf{i}]$;
$[\mathbf{d}] \leftarrow \mathbf{0}$;
foreach $j \in \{1,\ldots,m\}$ **do**
    $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n+m)$;
    $[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}],[\boldsymbol{\sigma}_i],-[t_j])$;
$[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}],[\mathbf{i}],1)$;
$[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
$[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{d}] \geq \mathbf{0})$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [\mathbf{b}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{A}\mathbf{d}] = \mathbf{0})$;
$[\gamma] \leftarrow [\mathbf{c}]([\mathbf{x}] + [\mathbf{d}]) < [\mathbf{c}][\mathbf{x}]$;
$\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta_i]) + \sum_{j=1}^{n}\left([\alpha_j'] + [\beta_j']\right)\right)$;
return $[\delta]$

Protocol A.64: $\delta \leftarrow \mathsf{VerifyUnbounded}_{\mathsf{LT},\mathsf{IP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [\mathbf{i}], [q])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+p+1)}$,
$[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[\mathbf{i}] \in \{0,1\}^{n+m}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
Output: $\delta \in \{0,1\}$.

$[\mathbf{t}] \leftarrow [\mathbf{T}][\mathbf{i}]$;
$[\mathbf{d}] \leftarrow \mathbf{0}$;
foreach $j \in \{1, \ldots, m\}$ **do**
  $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n+m)$;
  $[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}], [\boldsymbol{\sigma}_i], -[t_j])$;
$[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}], [\mathbf{i}], 1)$;
$[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
$[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{d}] \geq \mathbf{0})$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [q][\mathbf{b}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{A}]([\mathbf{x}] + [\mathbf{d}]) \leq [q][\mathbf{b}])$;
$[\chi] \leftarrow 1 - ([q] > 0)$;
$[\gamma] \leftarrow [\mathbf{c}]([\mathbf{x}] + [\mathbf{d}]) < [\mathbf{c}][\mathbf{x}]$;
$\delta \leftarrow \mathsf{EQZ}\left([\gamma] + [\chi] + \sum_{i=1}^{m}([\alpha_i] + [\beta_i]) + \sum_{j=1}^{n}\left([\alpha'_j] + [\beta'_j]\right)\right)$;
return $[\delta]$

A.3.2 Small Tableau Simplex

Protocol A.65: $\delta \leftarrow \mathsf{VerifyOptimal}_{\mathsf{ST},\mathsf{RP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [\mathbf{x}], [\mathbf{u}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+p+1)}$,
$[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{u}] \in \{1, \ldots, n+m+p\}^{n+p}$.
Output: $\delta \in \{0,1\}$.

$[t] \leftarrow \mathbf{0}$;
foreach $i = 1, \ldots, n$ **do**
  $[\boldsymbol{\upsilon_i}] \leftarrow \mathsf{ConvertUnary}([u_i], n+m)$;
  $[t] \leftarrow \mathsf{WriteAtPosition}([\mathbf{t}], [\boldsymbol{\upsilon_i}], -[t_{(m+1)i}])$;
$[p] \leftarrow ([t_{n+1}], \ldots, [t_{n+m}])$;
$[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
$[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq \mathbf{0})$;
$[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] = [\mathbf{c}][\mathbf{x}])$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [\mathbf{b}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [\mathbf{c}])$;
$\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta'_i]) + \sum_{j=1}^{n}\left([\alpha'_j] + [\beta_j]\right)\right)$;
return $[\delta]$

Protocol A.66: $\delta \leftarrow \mathsf{VerifyOptimal}_{\mathsf{ST},\mathsf{IP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [\mathbf{x}], [\mathbf{u}], [q])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+p+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^{m}$, $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{u}] \in \{1, \ldots, n+m+p\}^{n+p}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.

Output: $\delta \in \{0, 1\}$.

$[t] \leftarrow \mathbf{0}$;
foreach $i = 1, \ldots, n$ **do**
  $[\boldsymbol{v_i}] \leftarrow \mathsf{ConvertUnary}([u_i], n+m)$;
  $[t] \leftarrow \mathsf{WriteAtPosition}([\mathbf{t}], [\boldsymbol{v_i}], -[t_{(m+1)i}])$;
$[p] \leftarrow ([t_{n+1}], \ldots, [t_{n+m}])$;
$[\boldsymbol{\alpha'}] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
$[\boldsymbol{\beta'}] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq \mathbf{0})$;
$[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] = [\mathbf{c}][\mathbf{x}])$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [q][\mathbf{b}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [q][\mathbf{c}])$;
$[\chi] \leftarrow 1 - ([q] > 0)$;
$\delta \leftarrow \mathsf{EQZ}\left([\chi] + [\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta'_i]) + \sum_{j=1}^{n}\left([\alpha'_j] + [\beta_j]\right)\right)$;
return $[\delta]$

Protocol A.67: $\delta \leftarrow \mathsf{VerifyInfeasible}_{\mathsf{ST}}([\mathbf{A}], [\mathbf{b}], [\mathbf{T}], [\mathbf{s}], [\mathbf{u}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+p+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^{m}$, $[\mathbf{u}] \in \{1, \ldots, n+m+p\}^{n+p}$.

Output: $\delta \in \{0, 1\}$.

$[t] \leftarrow \mathbf{0}$;
foreach $i = 1, \ldots, n$ **do**
  $[\boldsymbol{v_i}] \leftarrow \mathsf{ConvertUnary}([u_i], n+m)$;
  $[t] \leftarrow \mathsf{WriteAtPosition}([\mathbf{t}], [\boldsymbol{v}_i], -[t_{(m+1)i}])$;
  $[p] \leftarrow ([t_{n+1}], \ldots, [t_{n+m}])$;
$[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] > 0)$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq [\mathbf{0}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [\mathbf{0}])$;
$\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}[\alpha_i] \sum_{j=1}^{n}[\beta_j]\right)$;
return $[\delta]$

Protocol A.68: $\delta \leftarrow \mathsf{VerifyUnbounded}_{\mathsf{ST},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{T}], [\mathbf{s}], [q], [\mathbf{i}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+p+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$, $[\mathbf{i}] \in \{0,1\}^{n+m}$.
Output: $\delta \in \{0,1\}$.
return $\mathsf{VerifyUnbounded}_{\mathsf{LT},\mathsf{VAR}}([\mathbf{A}], [\mathbf{b}], [\mathbf{T}], [\mathbf{s}], [q], [\mathbf{i}])$

A.3.3 Revised Simplex

Protocol A.69: $\delta \leftarrow \mathsf{VerifyOptimal}_{\mathsf{RS},\mathsf{RP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{D}], [\mathbf{s}], [\mathbf{x}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{D}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$.
Output: $\delta \in \{0,1\}$.
$[\mathbf{p}] \leftarrow -([d_{(m+1)1}], \ldots, [d_{(m+1)n}])$;
$[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
$[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq \mathbf{0})$;
$[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] = [\mathbf{c}][\mathbf{x}])$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [\mathbf{b}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [\mathbf{c}])$;
$\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta'_i]) + \sum_{j=1}^{n}\left([\alpha'_j] + [\beta_j]\right)\right)$;
return $\mathsf{Open}([\delta])$

Protocol A.70: $\delta \leftarrow \mathsf{VerifyOptimal}_{\mathsf{RS},\mathsf{IP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{D}], [\mathbf{s}], [\mathbf{x}], [q])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{D}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[\mathbf{x}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[q] \in \mathbb{Z}_{\langle k \rangle}$.
Output: $\delta \in \{0,1\}$.
$[\mathbf{p}] \leftarrow -([d_{(m+1)1}], \ldots, [d_{(m+1)n}])$;
$[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
$[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq \mathbf{0})$;
$[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] = [\mathbf{c}][\mathbf{x}])$;
$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [q][\mathbf{b}])$;
$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [q][\mathbf{c}])$;
$[\chi] \leftarrow 1 - ([q] > 0)$;
$\delta \leftarrow \mathsf{EQZ}\left([\chi] + [\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta'_i]) + \sum_{j=1}^{n}\left([\alpha'_j] + [\beta_j]\right)\right)$;
return $\mathsf{Open}([\delta])$

Protocol A.71: $\delta \leftarrow \mathsf{VerifyInfeasible}_{\mathsf{RS}}([\mathbf{A}], [\mathbf{b}], [\mathbf{D}], [\mathbf{s}], [] T^0)$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{D}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(m+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[\mathbf{T}^0] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+p+m+1)}$.

Output: $\delta \in \{0, 1\}$.

1 $[\mathbf{p}] \leftarrow -[\mathbf{d}_{m+1}]([T^0_{n+1}], \ldots, [T^0_{n+m}])$;
2 $[\gamma] \leftarrow 1 - ([\mathbf{p}][\mathbf{b}] > 0)$;
3 $[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{p}] \leq [\mathbf{0}])$;
4 $[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{p}][\mathbf{A}] \leq [\mathbf{0}])$;
5 $\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}[\alpha_i] \sum_{j=1}^{n}[\beta_j]\right)$;
6 $[\delta]$

Protocol A.72: $\delta \leftarrow \mathsf{VerifyUnbounded}_{\mathsf{RS},\mathsf{RP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [q], [\mathbf{i}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+p+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$, $[\mathbf{i}] \in \{0, 1\}^{n+m}$.

Output: $\delta \in \{0, 1\}$.

1 $[\mathbf{t}] \leftarrow [\mathbf{D}][\mathbf{T}^0_i]$;
2 $[\mathbf{d}] \leftarrow \mathbf{0}$;
foreach $j \in \{1, \ldots, m\}$ **do**
3 $\quad[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n+m)$;
4 $\quad[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}], [\boldsymbol{\sigma}_i], -[t_j])$;
5 $[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}], [\mathbf{i}], 1)$;
6 $[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;
7 $[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{d}] \geq \mathbf{0})$;
8 $[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [\mathbf{b}])$;
9 $[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{A}]([\mathbf{x}] + [\mathbf{d}]) \leq [\mathbf{b}])$;
10 $[\gamma] \leftarrow [\mathbf{c}]([\mathbf{x}] + [\mathbf{d}]) < [\mathbf{c}][\mathbf{x}]$;
11 $\delta \leftarrow \mathsf{EQZ}\left([\gamma] + \sum_{i=1}^{m}([\alpha_i] + [\beta_i]) + \sum_{j=1}^{n}\left([\alpha'_j] + [\beta'_j]\right)\right)$;
12 **return** $[\delta]$

Protocol A.73: $\delta \leftarrow \mathsf{VerifyUnbounded}_{\mathsf{RS},\mathsf{RP}}([\mathbf{A}], [\mathbf{b}], [\mathbf{c}], [\mathbf{T}], [\mathbf{s}], [q], [\mathbf{i}])$

Input: $[\mathbf{A}] \in \mathbb{Z}_{\langle k \rangle}^{m \times n}$, $[\mathbf{b}] \in \mathbb{Z}_{\langle k \rangle}^{m}$, $[\mathbf{c}] \in \mathbb{Z}_{\langle k \rangle}^{n}$, $[\mathbf{T}] \leftarrow \mathbb{Z}_{\langle k \rangle}^{(m+1)\times(n+m+p+1)}$, $[\mathbf{s}] \in \{1, \ldots, n+m+p\}^m$, $[q] \in \mathbb{Z}_{\langle k \rangle}$, $[\mathbf{i}] \in \{0,1\}^{n+m}$.

Output: $\delta \in \{0, 1\}$.

$[\mathbf{t}] \leftarrow [\mathbf{D}][\mathbf{T}_i^0]$;

$[\mathbf{d}] \leftarrow \mathbf{0}$;

foreach $j \in \{1, \ldots, m\}$ **do**

  $[\boldsymbol{\sigma}_i] \leftarrow \mathsf{ConvertUnary}([s_i], n+m)$;

  $[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}], [\boldsymbol{\sigma}_i], -[t_j])$;

$[\mathbf{d}] \leftarrow \mathsf{WriteAtPosition}([\mathbf{d}], [\mathbf{i}], 1)$;

$[\boldsymbol{\alpha}'] \leftarrow \mathbf{1} - ([\mathbf{x}] \geq \mathbf{0})$;

$[\boldsymbol{\beta}'] \leftarrow \mathbf{1} - ([\mathbf{d}] \geq \mathbf{0})$;

$[\boldsymbol{\alpha}] \leftarrow \mathbf{1} - ([\mathbf{A}][\mathbf{x}] \leq [q][\mathbf{b}])$;

$[\boldsymbol{\beta}] \leftarrow \mathbf{1} - ([\mathbf{A}]([\mathbf{x}] + [\mathbf{d}]) \leq [q][\mathbf{b}])$;

$[\gamma] \leftarrow [\mathbf{c}]([\mathbf{x}] + [\mathbf{d}]) < [\mathbf{c}][\mathbf{x}]$;

$[\chi] \leftarrow 1 - ([q] > 0)$;

$\delta \leftarrow \mathsf{EQZ}\left([\chi] + [\gamma] + \sum_{i=1}^{m} ([\alpha_i] + [\beta_i]) + \sum_{j=1}^{n} \left([\alpha'_j] + [\beta'_j]\right)\right)$;

return $[\delta]$

Index

List of Symbols

$\mathbf{v}$	vector, (defined on page 9)
v_i	i-th entry of $\mathbf{v}$, (defined on page 9)
$\mathbf{M}$	two dimensional matrix, (defined on page 9)
$\mathbf{M}_j$	j-th row of $\mathbf{M}$, (defined on page 9)
$\mathbf{m}_i$	i-th row of $\mathbf{M}$, (defined on page 9)
m_{ij}	entry in row i and column j of $\mathbf{M}$, (defined on page 9)
$\mathbf{V_s}$	$(\mathbf{V}_{s_i} \dots \mathbf{V}_{s_m})$, where $\mathbf{s} = (s_1, \dots, s_m)$, (defined on page 29)
$\mathbf{e}_i$	i-th unity vector, (defined on page 44)
$\mathbf{I}_m$	$m \times m$ identity matrix, (defined on page 44)
$\parallel \mathbf{v} \parallel$	Euclidian norm of $\mathbf{v}$, (defined on page 50)
$\text{diag}(\mathbf{x})$	diagonal matrix where column i equals $x_i\mathbf{e}_i$, (defined on page 122)
$\text{argmin}(\mathbf{x})$	index of a minimum of $\mathbf{x}$, (defined on page 34)
$\bar{b}$	negation of the bit b, (defined on page 137)
$\lvert x \rvert_b$	boolean evaluation of x, (defined on page 40)
$x \in_R \mathcal{X}$	a uniformly random draw from the set $\mathcal{X}$ resulting in x, (defined on page 10)
$[s]_k$	Shamir share of s held by party P_k, (defined on page 10)
$[s]$	collection of all Shamir shares of s, (defined on page 10)
$[s]_i^A$	additive share of s held by party P_i, (defined on page 11)
$[s]^A$	collection of all additive shares of s, (defined on page 11)
$[s]_k^R$	replicated share of s held by each party P_i not in the k-th unqualified set, (defined on page 12)
$[s]^R$	collection of all replicated shares of s, (defined on page 12)

$x = x_{k-1} \dots x_0$	bit-decomposition of x starting with the most significant bit, (defined on page 90)
$[x]_B$	bitwise sharing of x, (defined on page 90)
$[\![x]\!]$	probabilistic homomorphic encryption of x, (defined on page 152)
$G = (V, A)$	directed hypergraph on vertices V and arcs A, (defined on page 156)
$\mathbb{Z}$	set of all integers, (defined on page 9)
$\mathbb{Z}_p$	set of integers modulo p, (defined on page 9)
$\mathbb{Q}$	set of all rationals, (defined on page 47)
$\mathbb{F}_q$	a finite field of order q, (defined on page 10)
$\mathbb{Z}_{\langle k \rangle}$	set of k-bit signed integers, (defined on page 78)
$\mathbb{Q}_{\langle k,f \rangle}$	set of k bit signed fixed point numbers with range f, (defined on page 99)
S_k	the group of all permutations acting on k elements, (defined on page 156)

Linear Programming

m	number of constraints, excluding nonnegativity, (defined on page 28)
n	number of unknowns, (defined on page 28)
$\mathbf{A}$	coefficients of the constraints, (defined on page 28)
$\mathbf{c}$	coefficients of the objective, (defined on page 28)
$\mathbf{b}$	constants of the constraints, (defined on page 28)
$\mathbf{x}$	primal solution, (defined on page 28)
$\mathbf{p}$	dual solution, (defined on page 41)
$\mathbf{s}$	basis, (defined on page 29)
$\mathbf{B}$	basis matrix, (defined on page 29)
$\mathbf{u}$	co-basis, (defined on page 29)
$\mathbf{d}$	direction, (defined on page 30)
$\mathbf{d}^\ell$	ℓ-th basic direction, (defined on page 30)
$\overline{c}$	cost-reduced vector, (defined on page 32)
$\mathbf{y}$	artificial variables, (defined on page 38)
$\mathbf{T}$	simplex tableau, (defined on page 44)

List of Protocols

Acknowledgements

This thesis is the result of four years of study and research at the Crypto and Coding group at the Eindhoven University of Technology. It would not have been possible without the help of many people.

First of all, I wish to thank my supervisor Berry Schoenmakers for creating the possibility to start as a Ph.D. student. Once started, due to his enthusiasm, perseverance and many useful insights, he turned out to be a very stimulating supervisor. I'm also very grateful for his sense of humor that created a nice working atmosphere. Together with my promotor Henk van Tilborg he improved my writing skills significantly. Also, I wish to thank Henk van Tilborg for being my promotor.

My Ph.D. studies were part of the EU-funded project SecureSCM. Eindhoven University of Technology completed an enthusiastic consortium working on Secure Supply Chain Management together with SAP, University of Mannheim, University of Milan, European Business School, Zaragoza Logistics Center, and DHITech Technological District in Puglia. I wish to thank Octavian Catrina, Florian Kerschbaum, Richard Pibernik, Ernesto Damiani, Axel Schroepfer, and the other members of the consortium for the nice discussions that gave direction to the research. Most of all, I wish to thank Octavian Catrina for the nice cooperation during the project. I very much enjoyed the many technical and personal discussions. Chapters 4 and 5 are the result of our intense cooperation.

I wish to express my gratitude to Onno Boxma, Andries Brouwer, Ronald Cramer, Jesper Buus-Nielsen, Milan Petkovic, and Alexander Schrijver for agreeing to be a member of my Ph.D committee and reading my thesis. Special thanks go to Andries Brouwer for his useful comments and help that improved Chapter 7 significantly.

I thank the members of the Crypto Club for the nice discussions on various interesting cryptographic topics. Many topics in this thesis have been discussed in those club sessions. Special thanks go to Berry Schoenmakers, Boris Škorić, José Villegas, Peter van Liesdonk, Tomas Toft, and Mikkel Krøigaard for the many club sessions spent on my research problems.

One of the properties of the Crypto and Coding group at the Eindhoven University of Technology is a very nice working atmosphere. Coming to the office was very enjoyable due to my office mates Peter van Liesdonk, Christiane Peters, José Villegas, Jing Pan, Peter Schwabe, Mehmet Kiraz, Reza Farashahi, Peter Birkner, Michael Naehrig, Antonino Somine, Mayla Bruso, Elisa Costante, Bruno Pontes Soares, Daniel Trivellato, Gaëtan Bisson, Relinde Jurrius, Jan-Jaap van Oosterwijk, Thijs Laarhoven, Dion Boesten, and Meilof Veeningen. I'm very grateful for the nice after-lunch coffee breaks with Peter L, Meilof and Relinde. Special thanks go to Henk van Tilborg for being a very accessible leader of the group and for organizing the daily tea-breaks in which all Ph.D students of the group could enjoy nice social discussions. Last, but certainly not least in this respect,

I wish to thank our secretary Anita Klooster and our administrator Floortje Haasen for their patience, support and very nice social conversations.

I couldn't complete my Ph.D. studies without the support of my family and friends. I am very grateful for their support in good times and in bad times. Especially, I wish to thank Marionne van de Camp, who has proofread my thesis and has given me valuable comments on style, language and formatting and Brigitte Gedike for designing the cover of this thesis.

Finally, I wish to thank my dear wife Yvonne for her unconditional love and support. She makes me feel confident and strong.

Sebastiaan de Hoogh
Oosterhout, July 2012

Curriculum Vitae

Sebastiaan de Hoogh was born on July 11, 1981 in Dongen, the Netherlands. He finished his pre-university education at the Sint-Oelbertgymnasium in Oosterhout, and started his studies in mathematics at the Eindhoven University of Technology in September 2001.

During the second year of his studies in mathematics he was selected with three other students to take part in a project about queueing theory. In this project a problem of the travel agency TUI was solved. In 2005 he did an internship at the Coding and Crypto group at the Eindhoven University of Technology under supervision of dr. Benne de Weger. The internship was to study the differential cryptanalysis which has been used to find collisions for some well known cryptographic hash functions. He received his Bachelor's of Science degree in 2006.

In 2007 he spent six months in Sydney Australia for an internship at the Macquarie University. Together with dr. Scott Contini he worked on improving the running time of the cryptographic hash function VSH, the Very Smooth Hash function. In 2008, after writing his Master's thesis *On the speed of VSH*, he received his Master's of Science degree in Industrial and Applied Mathematics. The degree was awarded cum laude.

He started as a Ph.D. student at the Crypto and Coding group at the Eindhoven University of Technology under supervision of dr. Berry Schoenmakers. This research was part of the EU-FP7 project SecureSCM which is about secure supply chain management and has led to various solutions to secure supply chain management using multiparty computation. In 2012 he works on the Commit project THeCS, which is about thrusted health-care services. Here, the results of SecureSCM and this thesis can be applied.

STATEMENTS

accompanying the dissertation

Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming

by

Sebastiaan Jacobus Antonius de Hoogh

1. Let $[x]$ and $[y]$ be secretly shared (or homomorphic encrypted) k bit numbers and let $\log^*(k) := \min\{i | \log^i(k) \leq 1\}$. There exists a protocol that, after a preprocessing stage, runs in $\log^*(k)$ interactive rounds and requires $\log(k)$ openings (or decryptions) for securely computing the secret bit $[x = y]$.

2. Universally Verifiable Multiparty Computation is achieved by applying the Fiat-Shamir heuristic, or Generalized Fiat-Shamir heuristic [AABN02], to the multiparty Σ-protocol of [CDN01].

3. Let p be prime. Let $\mathcal{P} \subset S_p$ be a permutation group, which is represented by either rotation or affine transformation. There exists a u-uniform directed hypergraph on p vertices such that $\mathcal{P}$ is precisely its group of automorphisms, where $u = 2$ if $\mathcal{P}$ is represented by rotations and $u = 3$ if $\mathcal{P}$ is represented by affine transformations.

   Furthermore, if $\mathcal{P} \subset S_{p+1}$ is represented by Möbius transformations, then there exists a 4-uniform directed hypergraph on $p + 1$ vertices such that $\mathcal{P}$ is precisely its group of automorphisms.

4. **Conjecture** There exists a 5-uniform directed hypergraph on 11 vertices such that the Mathieu group M_{11} is precisely its group of automorphisms. Also, there exists a 6-uniform directed hypergraph on 12 vertices such that the Mathieu group M_{12} is precisely its group of automorphisms.

5. Universally Verifiable Multiparty Computation that is also Universally Composable is achieved by applying the Fiat-Shamir heuristic, or Generalized Fiat-Shamir heuristic [AABN02], to the multiparty Σ-protocol of [DN03].

6. The protocols for linear programming by [LA06] are inefficient and incorrect: in every iteration the size of the values in the tableau doubles; this leads to exponentially large values. Furthermore, the solution extraction does not check independency of the columns; hence the result may be infeasible.

7. Let $\mathbf{A}$ be an $m \times n$ matrix and π be a function that rotates the columns and rows of $\mathbf{A}$, i.e., there exists $0 \leq r_1 < m$ and $0 \leq r_2 < n$ such that the j-th row of $\pi(\mathbf{A})$ is equal to the $(j + r_1 \bmod m)$-th row of $\mathbf{A}$ for all $0 \leq j < m$ and the i-th column of $\pi(\mathbf{A})$ is equal to the $(i + r_2 \bmod n)$-th column of $\mathbf{A}$ for all $0 \leq i < n$. To prove knowledge of π the protocol of [TW10] with respect to the Cayley graph of the group $(\mathbb{Z}_m \times \mathbb{Z}_n, +)$ can be used.

8. The running times of the protocols in this thesis can be improved by applying packed secret sharing if there are more than 5 parties.

9. It is only a matter of time before multiparty computation will be used in practice to solve real life problems.

10. Having a degree at a technical university does not at all guarantee enough technical insight to become a skilled handyman.

11. If a cryptographer says that a system is secure, then he means that under certain circumstances (the model) the system satisfies certain properties (the corresponding security definition).

12. Security in practice is not a matter of mathematics or computer science; it is a matter of psychology.

13. Personal experience shows that carnival is the strongest form of team building:

We zijn er weer bij en dat is prima
Viva, Eilandia
We houden van de crypto en koffie is een must
We werken door tot 's avonds laat nog lang niet uitgeblust

References

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the fiat-shamir transform: Minimizing assumptions for security and forward-security. In *EUROCRYPT*, pages 418–433, 2002.

[CDN01] R. Cramer, I. Damgård, and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–300. Springer-Verlag, 2001.
Full version (Oct 2000): http://eprint.iacr.org/2000/055.

[DN03] I. Damgård and J.B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer-Verlag, 2003.

[LA06] J. Li and M. Atallah. Secure and Private Collaborative Linear Programming. In *Proc. 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (ColaborateCom 2006)*, pages 19–26, Atlanta, USA, 2006.

[TW10] Björn Terelius and Douglas Wikström. Proofs of Restricted Shuffles. In *AFRICACRYPT*, pages 100–113, 2010.